A picture containing text

Description automatically generated

**UNIVERSITY OF HERTFORDSHIRE**

School of physics, Engineering and Computer Science

**Final Project report (FPR)**



**Project Title**

**Air Quality Index prediction using multiclass classifications machine learning algorithms.**

Name: Muhammad Mahfuzur Rahman

Student ID: 20015658

Course title: Advance Computer Science Masters Project

Course code: 7COM1039-0501-2021

Supervisor: Dr. Diego Resende Faria



# Abstract

Air pollution is becoming pressing issue nowadays and people are becoming sufferer for that. It actually relies on some ingredients like sulphar-di-oxide(So2), carbon monoxide(CO), nitrogen di oxide(NO2), respirable suspended particulate matter(rspm), suspended particulate matter,temperature maximum, temperature minimum, atmospheric pressure at sea level, average humidity, total rainfall, average visibility, average wind speed which made based on these characteristics the pm2.5 level will be classified.A range of machine learning classification algorithms will be used to classify the AQI levels correctly and the accuracy of those models as well as making comparison amidst the all ML models as Logistic regression, bagging and boosting technique like Decision tree classfier, Random forest classifier, AdaBoost classifier and be used support vector classifier , bayessian algorithm multinomial naïve bayes which is probability based algorithm and finally GradientBoosting classifier . To be compared all models which model gives comparatively good performance that is the predominant purpose to be used many algorithms.

At the very beginning, the project was tried to complete by regression algorithm but that was thought not worth for the project, so decision has been changed.

This paper gives the model for Air Quality Index (AQI) labelling data.

# Acknowledgement

I convey my heartiest congratulations to my honourable supervisor is Dr. Diego Resende Faria who has provided support me relentlessly to complete my project eventually.

It would be hugely ticklish for me if he couldn’t root for me from beginning to end. I intervened him invariably where there were no time bindings and straightforwardly, he has responded to me as well as delivered precise guidance what need to do. I acquired plenty of things as to data science by doing this project and having constructive instructions from my respectable supervisor.

Contents

[Abstract 2](#_Toc121486628)

[Acknowledgement 2](#_Toc121486629)

[List of Figures 7](#_Toc121486630)

[List of tables 10](#_Toc121486631)

[1.Introduction 11](#_Toc121486632)

[1.1 Problem definition 12](#_Toc121486633)

[1.2 Keywords 13](#_Toc121486634)

[1.3 Research Aim 13](#_Toc121486635)

[1.4 Research Question 13](#_Toc121486636)

[1.5 To Research 13](#_Toc121486637)

[1.6 Research Methodology 14](#_Toc121486638)

[1.7 Checking feasibility 15](#_Toc121486639)

[1.8 Contributions 15](#_Toc121486640)

[2 Literature Review 15](#_Toc121486641)

[2.1 Overview 16](#_Toc121486642)

[**2.2 Air quality prediction techniques** 16](#_Toc121486643)

[3 Methodology 17](#_Toc121486644)

[3.1 Methods Choice 17](#_Toc121486645)

[3.2 Validation of chosen models 18](#_Toc121486646)

[3.3 Project Design 18](#_Toc121486647)

[3.4 Profound insights of models 20](#_Toc121486648)

[3.4.1 Logistic Regression 20](#_Toc121486649)

[3.4.2 How logistic regression works 20](#_Toc121486650)

[3.4.3 Decision Tree Classifier 22](#_Toc121486651)

[3.4.4 Tree nodes 23](#_Toc121486652)

[3.4.5 Pruning 23](#_Toc121486653)

[3.4.6 Entropy 23](#_Toc121486654)

[3.4.7 Information gain 24](#_Toc121486655)

[3.4.8 Random Forest 24](#_Toc121486656)

[3.4.9 Pros and Cons for RF: 25](#_Toc121486657)

[3.4.10 K Nearest Neighbor 25](#_Toc121486658)

[3.4.11 Advantages and Disadvantages for KNN was given below: 26](#_Toc121486659)

[3.4.12 Naïve Bayes 26](#_Toc121486660)

[3.4.13 Pros and Cons for Multinomial Naïve bayes 27](#_Toc121486661)

[3.4.14 Support Vector Classifier 27](#_Toc121486662)

[3.4.15 Pros and Cons for SVM 28](#_Toc121486663)

[3.4.16 Gradient Boosting 28](#_Toc121486664)

[3.4.17 Advantage and Disadvantage of Gradient Boosting 28](#_Toc121486665)

[3.4.18 AdaBoost Classifier 29](#_Toc121486666)

[3.4.19 Ethical issue 30](#_Toc121486667)

[3.4.20 Legal issue 30](#_Toc121486668)

[3.4.21 Professional issue 31](#_Toc121486669)

[3.4.22 Commercial issue 31](#_Toc121486670)

[3 Implementation details 31](#_Toc121486671)

[1.1 Dataset collection 31](#_Toc121486672)

[4.2 Imported necessary libraries 31](#_Toc121486673)

[4.3 Datasets loading 31](#_Toc121486674)

[4.4 About data 32](#_Toc121486675)

[4.5 Columns adding to df 32](#_Toc121486676)

[4.6 Final df 33](#_Toc121486677)

[4.7 Data cleaning 33](#_Toc121486678)

[**4.7.1** **Columns renaming** 34](#_Toc121486679)

[**4.7.2** **Target column labelling** 34](#_Toc121486680)

[**4.7.3** **Ordering target column** 35](#_Toc121486681)

[**4.7.4** **Describing dataframe** 36](#_Toc121486682)

[**4.7.5** **Unique values** 37](#_Toc121486683)

[**4.7.6** **Checking duplicate values** 37](#_Toc121486684)

[**4.7.7 Columns present in dataset** 37](#_Toc121486685)

[**4.7.7** **Checking NaN** 38](#_Toc121486686)

[**4.7.8** **Percentage of NaN** 38](#_Toc121486687)

[**4.7.9** **Percentages of missing data** 39](#_Toc121486688)

[**4.7.10** **Removing redundant columns** 39](#_Toc121486689)

[**4.7.11** **Feature guidance details** 40](#_Toc121486690)

[**4.8** **Data analysis and Visualization** 41](#_Toc121486691)

**[4.8.1](#_Toc121486692)****[Plotted state column](#_Toc121486692)** [42](#_Toc121486692)

[**4.8.2** **Types of areas** 43](#_Toc121486693)

[**4.8.3** **Pollution controlling agencies** 44](#_Toc121486694)

[**4.8.4** **States vs so2** 45](#_Toc121486695)

[**4.8.5** **No2 vs state** 46](#_Toc121486696)

[**4.8.6** **Rspm vs states** 47](#_Toc121486697)

[**4.8.7** **Spm vs states** 48](#_Toc121486698)

[**4.8.8** **State vs PM2.5** 48](#_Toc121486699)

[**4.8.9** **Type vs PM2.5** 49](#_Toc121486700)

[**4.8.10** **So2 vs Pm2.5** 50](#_Toc121486701)

[**4.8.11** **No2 vs Pm2.5** 51](#_Toc121486702)

[4.8.12 Rspm vs PM2.5 52](#_Toc121486703)

[**4.8.13** **Spm vs PM2.5** 53](#_Toc121486704)

[4.8.14 CO vs PM2.5 54](#_Toc121486705)

[**4.8.15** **Ozone vs PM2.5** 55](#_Toc121486706)

[**4.9** **Feature engineering** 55](#_Toc121486707)

[**4.9.1** **NaN values handling** 56](#_Toc121486708)

[**4.9.2** **For so2 feature:** 57](#_Toc121486709)

[**4.9.3** **For no2 column:** 58](#_Toc121486710)

[4.9.4 For Rspm column: 59](#_Toc121486711)

[**4.9.5** **For spm column:** 60](#_Toc121486712)

[**4.9.6** **For CO column:** 61](#_Toc121486713)

[**4.9.7 For Ozone feature:** 62](#_Toc121486714)

[**4.10** **Data Splitting** 64](#_Toc121486715)

[**4.11 Data Scaling** 65](#_Toc121486716)

[**4.11.1 Scaling for train data** 65](#_Toc121486717)

[**4.11.2 Scaling for test data** 69](#_Toc121486718)

[**4.12 Data transformation** 70](#_Toc121486719)

[**4.12.1 For training data** 70](#_Toc121486720)

[**4.12.2 Test data Transformation** 80](#_Toc121486721)

[**4.13 Data Resampling** 81](#_Toc121486722)

[**4.13.1 Data discrepancies plot** 81](#_Toc121486723)

[**4.13.2 Data balanced plot** 82](#_Toc121486724)

[**4.13.3 Accuracies for LR with balanced data** 83](#_Toc121486725)

[4.13.4 Accuracies for LR with not balanced data 84](#_Toc121486726)

[4.13.5 For decision tree the accuracies by balanced data 84](#_Toc121486727)

[4.13.6 Accuracies without balancing for DT 85](#_Toc121486728)

[**4.14 Outlier detection** 85](#_Toc121486729)

[**4.14.1 Outlier ranges for so2 column** 87](#_Toc121486730)

[**4.14.2 Outlier ranges for no2 column** 88](#_Toc121486731)

[**4.14.3 Outlier ranges for rspm column** 89](#_Toc121486732)

[**4.14.4 Outlier ranges for spm column** 90](#_Toc121486733)

[**4.15 Features selection** 91](#_Toc121486734)

[**4.15.1 Features selection by Pearson** 91](#_Toc121486735)

[**4.15.2 Features selection by ExtraTreeClassifier** 92](#_Toc121486736)

[**4.16 Building Models** 94](#_Toc121486737)

[**4.16.1 Normalization** 94](#_Toc121486738)

[**4.16.2 Fitting Models** 94](#_Toc121486739)

[**4.16.3 Logistic Regression** 95](#_Toc121486740)

[**4.16.4 Performance reports evaluation** 95](#_Toc121486741)

[**4.16.5 Precision for LR** 96](#_Toc121486742)

[**4.16.6 Recall for LR** 96](#_Toc121486743)

[**4.16.7 F1 scores for LR:** 97](#_Toc121486744)

[**4.16.8 Hyperparameter Tuning for logistic regression** 97](#_Toc121486745)

[**4.16.9 Decision Tree Classifier** 97](#_Toc121486746)

[**4.16.10 Performance reports evaluation for DT:** 98](#_Toc121486747)

[**4.16.11 Hyperparameter Tuning for DT Classifier** 98](#_Toc121486748)

[**4.16.12 Random Forest Classifier** 98](#_Toc121486749)

[**4.16.13 Performance report evaluation for RF:** 99](#_Toc121486750)

[**4.16.14 Hyperparameter Tuning for RF Classifier** 99](#_Toc121486751)

[**4.16.15 K Nearest Neighbor Classifier** 99](#_Toc121486752)

[**4.16.16 Performance report evaluation for KNN:** 100](#_Toc121486753)

[**4.16.17 Hyperparameter Tuning for KNN Classifier** 100](#_Toc121486754)

[**4.16.18 Multinomial Naïve Bayes** 101](#_Toc121486755)

[**4.16.19 Performance report evaluation for MNB** 101](#_Toc121486756)

[**4.16.20 Hyperparameter Tuning for MNB Classifier** 102](#_Toc121486757)

[**4.16.21 Support Vector Classifier** 102](#_Toc121486758)

[**4.16.22 Performance reports evaluation for SVC** 103](#_Toc121486759)

[**4.16.23 Tuning for SVC** 103](#_Toc121486760)

[**4.16.24 Gradient Boosting Classifier** 104](#_Toc121486761)

[**4.16.25 Hyperparameter tuning for GBC** 104](#_Toc121486762)

[**4.16.26 AdaBoost Classifier** 106](#_Toc121486763)

[5 Results 107](#_Toc121486764)

[**5.1** **Comparison amidst all models’ accuracies** 108](#_Toc121486765)

[**5.2** **Roc Auc Scores** 108](#_Toc121486766)

[**5.3** **Classification report details before and after tuning** 109](#_Toc121486767)

[6 Conclusion and Future works 110](#_Toc121486768)

[**6.1 Conclusions** 110](#_Toc121486769)

[**6.2 Future works** 111](#_Toc121486770)

[Reference 111](#_Toc121486771)

[Appendix and Code 112](#_Toc121486772)

[**Appendix A Data Loading** 112](#_Toc121486773)

[**Appendix B Data visualization** 121](#_Toc121486774)

[**Appendix C Feature Engineering** 128](#_Toc121486775)

[**DETECTING AND REMOVING OUTLIERS BY GAUSSIAN RANGE AND IQR** 143](#_Toc121486776)

[**FEATURE SELECTION** 146](#_Toc121486777)

[**selection by correlation** 146](#_Toc121486778)

[**selection by extratree classifier** 147](#_Toc121486779)

[**Appendix D Models Building** 149](#_Toc121486780)

[**MODELS BUILDING** 149](#_Toc121486781)

[**LOGISTIC REGRESSION** 149](#_Toc121486782)

[**Decision Tree Classifier** 152](#_Toc121486783)

[**Random Forest** 154](#_Toc121486784)

[**K-Nearest Neighbours** 156](#_Toc121486785)

[**MULTINOMIAL NAIVE BAYES** 159](#_Toc121486786)

[**SUPPORT VECTOR MACHINE** 161](#_Toc121486787)

[**GRADIENT BOOSTING CLASSIFIER** 163](#_Toc121486788)

[**ROC\_AUC\_Score** 165](#_Toc121486789)

# List of Figures

Fig 1: Project architecture…………………………………………………………………19

Fig 2: OVR …………………………………………………………………………………21

Fig 3: Decision tree………………………………………………………… …………….22

Fig 4: KNN ………………………………………………………………………………….25

Fig5: Adaboost classifiers………………………………………………. ……………….30

Fig 6: Datasets loading……………………………………………………………………31

Fig 7: Columns adding ……………………………………………………………………32

Fig 8: Final dataframe …………………………….......................................................33

Fig 9: New dataset volume ……………………………………………………………….33

Fig 10: Renaming columns……………………………………………………………….34

Fig 11: Categories target column……………………………………. …………………35

Fig12: Mapped target column ……………………………………………………………36

Fig 13: Describing datarame …………………………………………………………….36

Fig 14: Unique values ……………………………………………………………………37

Fig 15: Duplicate values…………………………………………………………………..37

Fig 16: Columns of df ……………………………………………………………………..37

Fig 17: Null values………………………………………………………………………..38

Fig 18: Percentages columns NaN …………………………………………………….38

Fig 19: Columns missing values ……………………………………………………….39

Fig 20: Dropped unnecessary columns ……………………………………………….39

Fig 21: Values of state column ………………………………………………………….41

Fig22: State column visualization ………………………………………………………42

Fig 23**:** Areas of provinces ……………………………………………………………….43

Fig 24: Name of agencies…………………………………………………………………44

Fig 25: SO2 measurement in states…………………………………. …………………45

Fig 26**:** NO2 measurement in states……………………………………………………. 46

Fig 27**:** RSPM measurement in states……………………………… ………………….47

Fig 28: SPM measurement in states ……………………………………………………48

Fig 29**:** PM2.5 level in states……………………………………………………………..48

Fig30: PM2.5 level in different areas …………………………………………………..49

Fig 31: Relation between so2 and PM2.5 ………………………………………………50

Fig 32: relation between no2 and PM2.5 ………………………………………………51

Fig 33: Relation between rspm and PM2.5…………………………………………….52

Fig 34: Relation between spm and PM2.5 …………………………………………….53

Fig 35: Relation between CO and PM2.5 ………………………… ………………….54

Fig 36 : Relation between Ozone and PM2.5 ………………………………………….55

Fig 37: Interpretation data ………………………………………………………………. 56

Fig 38: Histogram for so2 ………………………………………………………………...57

Fig 39: Right skewed ……………………………………………………………………..57

Fig 40: Box plot for so2 …………………………………………………………………..58

Fig 41: Histogram for no2 ………………………………………………………………..58

Fig 42: Box plot for no2…………………………………………………….. ……………59

Fig 43: Histogram for rspm ………………………………………………………………59

Fig 44: Box plot for rspm …………………………………………………………………60

Fig 45: Histogram for spm ……………………………………………………………….60

Fig 46: Boxplot for spm …………………………………………………………………..61

Fig 47**:** Histogram for CO ……………………………………………………………….61

Fig 48: Boxplot for CO………………………………………………………. ………….62

Fig 49: Histogram for Ozone……………………………………………… ……………62

Fig 50: Boxplot for Ozone…………………………………………………. ……………63

Fig 51: Null values handling by median……………………………………………….. 63

Fig 52: Checking null values …………………………………………………………….64

Fig 53: Data partitioning ………………………………………………………………….65

Fig 54: Train data scaling ………………………………………………………………..66

Fig 55: After scaling so2 histogram …………………………………………………….66

Fig 56: after scaling no2 histogram……………………………………………………..67

Fig 57: After scaling rspm histogram …………………………………………………..67

Fig 58: After scaling spm histogram ……………………………………………………68

Fig 59: After scaling CO histogram……………………………………………………. 68

Fig 60: After scaling no2 histogram ……………………………………………………69

Fig 61**:** Scaled test data ………………………………………………………………….69

Fig 62: Before transformation so2 col. ………………………………………………….70

Fig 63: after transformation so2 col. …………………………………………………….71

Fig 64: Before transformation no2 col. ………………………………………………….72

Fig 65: After transformation no2 col. …………………………………………………….73

Fig 66: Before transformation rspm col. ………………………………………………...74

Fig 67: After transformation rspm col. …………………………………………………..75

Fig 68: Before transformation spm col. …………………………………………………76

Fig 69: After transformation spm col. ……………………………………………………77

Fig 70: Before transformation CO ……………………………………………………….78

Fig 71: After transformation CO ………………………………………………………….79

Fig 72: Before transformation Ozone ……………………………………………………79

Fig 73: After transformation Ozone ……………………………………………………...80

Fig 74: Data discrepancies histogram …………………………………………………..81

Fig 75: Data balanced histogram………………………………………. ……………….82

Fig 76: With resampling scores for LR ………………………………………………….83

Fig 77: Without resampling score for LR ………………………………………………..84

Fig 78: With balancing for DT ……………………………………………………………84

Fig 79: Without balancing for DT ………………………………………………………...85

Fig 80: Conception about computing outliers with 3 std. ……………………………...86

Fig 81**:** Outlier detection for so2 …………………………………………………………87

Fig 82**:** Outlier detection for no2 …………………………………………………………88

Fig 83**:** Outlier detection for rspm ………………………………………………………..89

Fig 84**:** Outlier detection for spm ………………………………………………………...90

Fig 85**:** Correlation metrics heatmap ……………………………………………………91

Fig 86**:** Function for checking correlation………………………………………………..92

Fig 87: Feature importance by extra tree ……………………………………………….93

Fig 88: Sigmoid function ………………………………………………………………….95

Fig 89: Classification report for LR ………………………………………………………96

Fig 90: Classification report for DT ………………………………………………………98

Fig 91: Classification report for RF ………………………………………………………99

Fig 92: Classification report for KNN …………………………………………………..100

Fig 93: K values vs error rate …………………………………………………………..101

Fig 94**:** Classification report for MNB ………………………………………………….101

Fig 95: Classification report for SVC …………………………………………………..103

Fig 96: Classification with accuracy for GBC …………………………………………104

Fig 97: Learning rates vs train&test scores for GBC ……………………………….. 105

Fig 98: Classification report with accuracies for AdaBoost…………………………..106

# List of tables

Table 1: Overall accuracy information …………………………………………………108

Table 2**:** Classification report for before and after tuning…………………………….110

# 1.Introduction

In modern years, the Internet of Things (IoT) technology, data compilation methods have been uninterruptedly proliferated, and a vast amount of data has been compiled. Underlying on this,

ML algorithms are being used extensively in every sector either classify or predict accurately which is helping every sector to be taken proper decision.

The data science project has been taken owing to that nowadays everything is becoming digitalised so thereby on every day in every organization is generated huge amount of data which mainly carries the real scenarios of any company or any other organization, but question arisen that how any organization get benefited by analysing data correctly and help them to take proper steps to boost their business so that they can evade their business loss.

Hence can be given an instance How business loss can be evaded to analyse data that is suppose a multi-national company generates abundant products and sell to across the world so they have various kinds of data about selling how much for single day to year but the company doesn’t know what is demand for specific product and what are the numbers they need to generate against demand from public because of that , they need to analyse data which helps them to be sure what would be the quantity of products according to customers demand so that they produce certain amount of products as well customer review is most vital data which also important to be analysed for that what customers anticipate on specific product. Aiding precisely on these points the data science comes up with to be taken proper decision and these tasks can be done dexterously by machine learning algorithm which has mainly three categories as supervised machine learning which label is known and can be regression type or classification type as well as unsupervised machine learning which is generally done by clustering and finally reinforcement learning.

Since data science is a topic which has lots of captions to pursue anything into deep dive that’s why it was opted in to research on the topic that is air quality index prediction using multi classification algorithms.

It is worth mentioning that nothing throughout the world is perfect so machine learning is not exception in this regard that means it can’t give result which would be spot on utterly just keeping the matter the project is going to be implemented.

## 1.1 Problem definition

Presently, it is seen climate changing is the most burning issue across the world for which global environment is affecting in many ways as temperature is rising day by day, on the other hand the precipitation of rain is decreasing gradually as a result some metropolitan cities are facing crux as to surface water and people lives have become deplorable. A range of problems have also seen for some years that due to deforestation the climate changed his formation as his own accord.

In this project the researcher is predominantly trying to figure out that classifying the label of air quality index whether it is good, bad, or moderate underlying on the criterions as SO2, NO2, RSPM, SPM, CO and O3(Ozone) which are mainly particles of air.

In medical science, it is proved that above mentioned particles of air which are pernicious for human respiratory system and responsible for many chronic diseases like asthma, brochities, heart disease, copd (chronic obstructive pulmonary disease), lung cancer and so on.

The research was endeavoured doing mainly some of the states of India’s air quality. India’s air pollution is deemed the fifth most polluted country across the world as a result the country is suffering to tackle air pollution in many ways albeit India’s government has taken some steps to tackle but the demarcation of air pollution intensity has stood at culmination level which is incredibly cumbersome to curb the pollution for the government.

Severe air pollution in India hits predominantly in every sector as education, health, national economy, industry sector.

Industries are confronting many problems as for unexpected weather those who are employees in industry, they can’t join to their workstation due to inevitable air pollution so that company loses their productivity on time and labours lose their valuable time as well govt. loses revenue.

The pollution of air scenario contains so dire situation thereby Indian govt. must scrap all primary education sometimes which affects hugely on children education on the other hand, govt. is losing huge amount of foreign currencies now that foreign industries don’t want to make investment to this country which puts immediate bad impact on country’s national GDP (gross development product)

The development of India’s economy the predominant

challenging task is to improve the air quality to rise the economy. It is seen that for the bad Air quality, India spends a hefty amount to improve air quality in every year, but the real reason remains elusive so it may be possible to

unearth what is the prime reason to rise of PM2.5 level and on which factors it depends mainly that means what’s the correlation between independent variables and dependent variables.

Using above mentioned machine learning algorithms will be tried to classify AQI levels and as well be used hyperparameter technique for getting an utmost accuracy for learning algorithms.

## 1.2 Keywords

Air quality, Machine learning algorithms, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, KNearestNeighbors Classfier, Multinomial Naïve bayes, Support Vector Classifier, Gradient Boosting Classifier , AdaBoost Classifier.

## 1.3 Research Aim

The purpose of the research is to predict the Air Quality Index impeccably using various machine learning algorithms depending on some characteristics factor and a decision factor and will be quested for the relationship between independent variables and dependent variable. What is more, there is any improvement of accuracy to be used hyperparameter tuning technique on learning models also will be scrutinized.

## 1.4 Research Question

In this research the below qs. answers will be pursued:

1. Is there any coefficient correlation between independent variables and dependent

variables?

2. Can feature importance keep any substantial effect to boost accuracy?

3. Can hyperparameter technique give any advantageous for improving prediction duly with a view to predicting PM2.5?

4. Does bagging classifier give better prediction performance than boosting classifier technique?

algorithms?

5. Does Random Forest give better accuracy than Gradient boost algorithms?

## 1.5 To Research

The following steps needs to be completed to complete the research:

1. Importing the necessary libraries
2. Doing some analysis
3. Completing all feature engineering
4. Doing data splitting
5. Data scaling
6. Data transformation if data is not normally distributed
7. Checking the dataset whether it is balanced or any discrepancies
8. Feature selection
9. Applying machine learning algorithms
10. Using hyperparameter tuning
11. Performance measurement

12.Computing Roc\_Auc\_Score

## 1.6 Research Methodology

Steps were given below for following research methodology

|  |  |
| --- | --- |
| Steps | Process |
| Step1 | Data compilation from Kaggle |
| Step2 | Data pre-processing will be done to follow some techniques as EDA, feature importance,  , missing values handling and encoding. |

|  |  |
| --- | --- |
| Step3 | Bagging techniques will be used for building ML models as Random Forest (RF),  Decision Tree (DT) and boosting technique like AdaBoost and GradientBoost will be  used though they are complex for any model, so it needs to be checked. Others ML models as support vector classifier, multinomial naïve bayes, logistic regression also will be implemented. |
| Step4 | Performance evaluation will be done by computing accuracy score as well precision, recall and f1 score. |
| Ste5 | Hyperparameter optimization will be done by trial and error and computed performance metrics after applying this technique. |
| Step6 | Will be made comparison amidst all of models to be assumed which one delivers good from others. |
| Step7 | AUC ROC score will be figured out for all ML algorithms eventually which gives glimpse as to model performance as well how much model is capable of distinguishing class. |

## 1.7 Checking feasibility

In this project the feasibility has been checked for the dataset whether it would fit or not. There were 18 variables but some columns are redundant so those was dropped and rest of the were kept fitting the model. As a tool, python programming language was used to check the feasibility study.

## 1.8 Contributions

The project was completed utterly without any contributions by other people and wherever any necessary insights were needed all had been taken those which were pertinent. No human contributions involved with this project. That is spot on that all necessary information, literature review and profound ideas those were pertained with this project has been taken from different sources to be made more informative this project.

# 2 Literature Review

## 2.1 Overview

The proposed project is air quality index prediction using multi classification machine learning algorithms. The project predominantly proposed this topic because of that nowadays the across of world confronting the environmental issues as like weather prediction, air pollution measurement, carbon emission reduction, greenhouse effect, flood risk management all are pertained with the environment.

It is discerned that air quality is being worsened rapidly in India’s provinces by significant pollutants, including So2,No2,CO,Ozone,RSPM(respirable suspended particulate matter) and SPM(suspended particulate) and Industry growth, urbanization and rapid infrastructure development are preponderantly liable for air pollution as a consequence the particulate matter PM2.5 increasing rate is upward and people lives are affecting severely in many ways as they are suffering diseases like many chronic diseases, including asthma, breathe difficulties, lung cancer, heart disease and allergic problem. The problem hits in every sector like education, health, employment. To curtain such problems the PM2.5 level should be categorised underlying on some criterions as mentioned above and predicted correctly whether the level of PM2.5 is good, bad, moderate, or unhealthy.

To be acquired more insights about this problem many research papers was gone through.

## **2.2** **Air quality prediction techniques**

Since air quality index prediction is a burning issue nowadays so it has focused many researchers over the universe and were researched many more on air quality index prediction.

The authors used ensembles methods like decision tree, random forest, svm and ANN for predicting air quality index prediction([Kekulanadara](https://ieeexplore.ieee.org/author/37089256487), [Banujan Kuhaneswaran](https://ieeexplore.ieee.org/author/37088639736),2020) and using machine learning classification algorithms they classified six categories depending on target column PM2.5. The researcher([Madan](https://ieeexplore.ieee.org/author/37088803218), [Sagar](https://ieeexplore.ieee.org/author/37086873872), [Virmani](https://ieeexplore.ieee.org/author/38190416900),20) used discrete data for this research using Decision tree classier and support vector classifier, ANN,Linear Regression for the air quality index prediction and dataset was collected from Kaggle.

The another research was done by the researchers([Meris](https://ieeexplore.ieee.org/author/37088535698), [Dimaunahan](https://ieeexplore.ieee.org/author/37086298477),  [Fadchar](https://ieeexplore.ieee.org/author/37088373275),20) on the topic IoT based automated indoor air quality by using support vector machine. They researched that why the indoor air quality can be more risky than outdoor air quality. The ( [Permai](https://ieeexplore.ieee.org/author/37089004498), [Alfi , Zakiyyah](https://ieeexplore.ieee.org/author/37089573342), [Tanty](https://ieeexplore.ieee.org/author/37085635320), 21) have researched on multiclass classification for air quality in Jakarta using support vector machine and multi-layer perceptron classifier. First the authors believed both algorithms generate high accuracy and they just wanted to be sure which one gives better then after having results they seen the svm has given better accuracy from the multi-layer perceptron.

Forecasting daily ambient air pollution relying on least square support vector machine and for this research the data was compiled from the daily monitoring system data. They stated that multi-layer perceptron has drawbacks and by using the proposed machine learning technique they have been able to tide over the imperfections of MLP algorithm. The authors were ( [Ip](https://ieeexplore.ieee.org/author/37541235900), [Vong](https://ieeexplore.ieee.org/author/37529302200),  [Yang](https://ieeexplore.ieee.org/author/37538617100), 2010).

# 3 Methodology

To be completed the project, lots of experiments had to be done and they were carried out a computer which configurations were windows 10 home, Intel(R) Core (TM) i3-3110M CPU @ 2.40GHz and 12 GB RAM and as IDE the jupyter notebook, google colab,Python programming language had been used for implementation.

In this chapter the project researcher would like to provide throughout the scenario of this project elaborately what has been used as machine learning classification algorithms in this project, including as Logistic Regression, Decision Tree Classifier, Random Forest Classifier, K-Nearest Neighbors, Multinomial Naïve Bayes classifier, Support Vector Classifier, Gradient Boosting classifier, AdaBoost classifier and SGDClassifier (stochastic gradient descent algorithm).

## 3.1 Methods Choice

The project is preponderantly focused to be classified air quality index data using multi class classification machine learning algorithms which is prediction basis work because of that, keeping the purpose of project the datasets have been compiled from the Kaggle which is known as open-source dataset for data science projects. Those who are enthusiastic to do work with data science projects thein main preference are Kaggle which is more resourceful. Having been completed all data pre-processing techniques some ML algorithms were applied to create model for the proposed project, including as Logistic Regression, Decision Tree Classifier, Random Forest Classifier, K-Nearest Neighbors, Multinomial Naïve Bayes classifier, Support Vector Classifier, Gradient Boosting classifier and AdaBoost classifier. These above referred algorithms had been used successfully to build models and had accuracy and there is also a reason specially for bagging and boosting technique were used as Decision Tree Classifier, Random Forest Classifier, gradient boost, AdaBoost which are mainly robust to avoid overfitting and bias and variance. It is needed to mention that sometimes to be seen solely the train and test accuracy of model that can’t be said that the model is handy or performance not dismal However, the precision, recall and F1-score must be checked besides owing to if any dataset is imbalanced then it must have to be checked all classification reports for evaluation purpose.

## 3.2 Validation of chosen models

Some studies have pretended that the bagging and boosting technique specially give good performance for both train and test score and behind the reason it is that the above referred two techniques are mainly come from ensemble technique which predominantly combines multiple estimates to a single estimate to expedite model to have better accuracy on train and test data also the ensemble technique is incredibly robust to lessen the bias and variance which are really significant for the better accuracy owing to the fact that on the one hand, high bias is not handy for model , on the other hand, high variance is also substandard for the generalised model. The rest of the algorithms as SVC is good for large scale dataset and accuracy can be increased by parameter tuning so it’s discreet to be chosen the algorithm. The logistic regression also commonly used for classification, but issue is that it is sensitive to outliers which may impact on model performance because it aids increasing standard deviation as a result, some specific data do stance far from most data. In this project, the researcher mainly endeavoured to apply six algorithms, including all bagging, and boosting techniques, SVC, multinomial naïve bayes and rest of the models were applied just to check how actually they treat to performance.

## 3.3 Project Design

As it has been mentioned earlier in the above that the dataset was collected from the most reliable source is Kaggle. Following that, the exploratory data analysis as Univariate, Bivariate and Multivariate analysis was done. After that, having been completed the feature engineering as columns imputation for null values, data splitting, data scaling, data transformation, data resampling, feature selection and eventually created the ML models using all classification algorithms. As project has been done by following some steps that was drawn below:

Dataset

Train data scaling

Train(70%)

EDA

Data splitting

Test data scaling

Test(30%)

Data transformation

Data resampling

Model building using ML classification algorithms

Feature selection

Model evaluation

Confusion metrics

End

Fig 1: Project architecture

## 

## 3.4 Profound insights of models

3.4.1 Logistic Regression:

Logistic regression mainly solves classification problem which predominantly does work with categorical data unlike linear regression does work with continuous data specifically for the outcome of prediction.

The algorithm can be used for both classification binary and multi class classification that means if it solves binary classification problem then there will be two categorical data as 1 or 0 or as instance it can be said that the cancer cell is malignant or benign that is incredibly simple on the other hand, in multi class classification there can be two or more categorical data as weather can be bad , good, moderate, unhealthy, hazardous so, by applying logistic regression the multi class problem can be solved simply however, while creating model there is a parameter called ‘multi\_class’ which normally has ‘auto’ as a value so , that can be changed value by ‘ovr’(one vs rest) both are the identical for giving accuracy the researcher had that.

### 3.4.2 How logistic regression works

It mainly works in two pages one is learning page where data fitting to model and another page is prediction page where some fraction of data to be fed to model so that when the new data comes, the model easily can predict relying on the trained data earlier.

As it is mentioned earlier that for multi class classification the algorithm work basically ovr method which means one class vs rest of all classes.

In this project, there are 5 classes, including good, moderate, bad, unhealthy, and very unhealthy so, here we need 5 different logistic regression classifiers.

While training for the class 1 then the class 1 input label will be positive samples that means y==1and rest of the classes will be negative samples means y==0 and the same process will continue to rest of the classes. Figure 2 was drawn for having conspicuous idea as to OVR below:

Diagram

Description automatically generated



Fig 2: How OVR (one vs rest) works for multiclass classification



In the above figure 2 was shown that how OVR works for multi class classification where in the first model the class 1 classified as positively, and rest of classes classified as negative categories then in the second model class 2 classified as positive category and rest of all classified as negative categories and eventually class 3 classified as positive category and rest of them were negative categories.

Since this project is multiclass classification project so in this project the logistic regression OVR will work like above figure 2 for the five classes as good, moderate, bad, unhealthy, and very unhealthy.

In the Prediction page, after completing all training process then it can be predicted that which class carries what label.



For the prediction purpose the probability is used for all classes, after that the ‘ovr’ picks the class which has maximum probability as like if the class ‘good’ probability is 0.6 and rest of classes probabilities are below 60% then the ‘good’ class will be the output.

Logistic regression mainly follows the linear equation which is

Y=mx+c ………………..(1)

where m is slope, x is co-efficient, and c is intercept.

### 3.4.3 Decision Tree Classifier

Decision tree classifier is a supervised machine learning which is a tree-based algorithm. It can be used for both regression and classification model and used for linear and non-linear data but significantly is used for non-linear data.

The boosting and bagging ensemble technique mainly constructed based on decision tree classifier traits.

Although the algorithm can be used for both regression and classification, but principles are a bit different, and decisions comes for both relying on any feature with conditions.

The algorithm behaves like trees, and it has some characteristics as root, branch, leaf node. When it gets dataset then splits the dataset many small subsets. Tree begins from root node after turning out many branches it ends with leaf nodes.

One instance can be given for clear conception that

does the person have any risk to have loan? Diagram

Description automatically generated



Fig 3: How Decision tree works

In the above the figure 3 says that how decision tree works with condition basis. If condition meets, then the tree will expand and finally will go to the leaf node and if condition doesn’t meet with criteria, then trees will not expand as well will finish at leaf node from root node directly.

### 3.4.4 Tree nodes

**Root node**: In this node an attribute is selected

**Internal node**: The node basically non-leaf node which depicts a test on a column and each branch represents the aftermath of the test

**Leaf node:** Each leaf node parades a class label.

### 3.4.5 Pruning

Pruning carries the meaning is that to trim redundant branches from root node to direct leaf node or terminal node so simply can be said that it’s a shortening process of tree.

Now qs. is that what the selection process of root node is. the answer should be information gain or Gini index. So first it is needed to compute entropy to be figured out information gain.

### 3.4.6 Entropy

Now qs. is that what is entropy and how it works?

Entropy measures pure subset of the split. To have the leaf node promptly we should select right attribute on the root node. Suppose initially on the root node 9 yes and 5 no ten after splitting two internal nodes have 6y/3n and 3y/2n then it’s not proper subset rather it’s impure subset. Proper subset means either yes or no will have on the terminal node. After computing entropy by this cue then it could be said which split more acceptable.

H(s)=-p (+) logbase2p(+)-p(-) logbase2p(-) where p means probability and p(+) means percentage of positive class and p(-)means percentage of negative class. The entropy range should be always 0 to 1. Which node entropy is high the node should split more. Entropy 1 means utter impure subset.

### 3.4.7 Information gain

To be computed information gain that needs to calculate summation of after splitting subset divided by total subset get multiplied with the entropy of after splitting subset and substruction from the entropy of the subset.

Suppose if there are three attributes as f1, f2 and f3 then concerned is that what the root node is, so answer would be that after computing information gain like placing f1 feature in the root and rest are in the subsets and the same for the rest two attributes then for which the information gain will be maximum that will be root node.

### 3.4.8 Random Forest

The algorithm is commonly used for classification problem. It is a bagging technique which come from the ensemble method. Ensemble method is a method which combines multiple estimates to a single estimate and gives better prediction performance. It solves both kinds of problem as regression and classification.

Let it be discussed what the bagging technique is and how it does work for random forest classifier.

The algorithm also a tree-based algorithm which principles predominantly come from decision tree as its discussed elaborately. Here also be used multiple subsets of decision trees.

Bagging technique is like that suppose we have a dataset where contains some rows and features. Subsequently having some samples from the dataset and give to the DT1 with row sampling with replacement as well feature sampling. Following that, the same process which presented for the DT1 is applied to the DT2 where some row samples as well column samples get replied from DT1 to DT2 and the process will remain same for others decision tree models then the models get trained data. Having got trained by training data when teat data comes then all models will give output if it is binary then 1 or 0 and it it’s multi class then output will be more. Let the samples outputs are 1,1,0,1 for four models so it is distinctly seen that the majority voting number is 1 which will be the output and the process is known as bootstrap aggregation.

One very specific matter for random forest is that trees go max depth then it is most likely to get low bias and high variance however, since the output doesn’t come depending on only one decision tree so there is less chance to be overfitted because when output comes relying on majority vote from the multiple decision trees then high variance is converted into low variance and model becomes a generalised model.

### 3.4.9 Pros and Cons for RF:

Pros:

1. If the features are highly correlated to each other’s then random forest easily can tackle this problem owing to when multiple decision trees get trained, they trained by random sampling data.
2. Since the algorithm is a bagging ensemble technique so after training data successfully in multiple trees the outcome come from multiple decision not single decision and which number is majority percentage that is the output eventually as thereby the individual inaccuracy can avoid simply and can handle high variance.
3. Imbalanced dataset is not issue for this algorithm
4. It can handle overfitting also.
5. Random forest is not sensitive to outliers.

Cons**:**

1. Features should be robust in prediction power otherwise they don’t want to work properly.
2. An unexpected error like black box may appear that means that sometimes it can be seen the massive difference among train and test accuracy due to nobody knows what is factually taking place internally so tuning is best way to deal that.

### 3.4.10 K Nearest Neighbor

How this algorithm works the answer can be got if we have a look to this title of the algorithm that those are nearest neighbor the algorithm works with them.



Fig 4: How KNN algorithms works

In the above drawn figure 4 has been shown 2 coloured data points one is red, and another is black, and one new data point is positioning a bit top from the two-coloured data points so there is qs. that when new data point comes then how can be determined whether the new query point belongs to the red point, or it belongs to the black point so k nearest neighbor assists it to give the answer to the question.



Hence, we must deem two things that how many k nearest neighbor we must consider that means the K value selection and another one is the computation of distance or nearest points from new query data to belong to the data points by calculating Euclidean distance. Then we must think that how many categories present from category 1 and how many present from category 2 if it is said the K value is 5 suppose.



If there are maximum number of categories from category 1 (red data points) can be found for nearest neighbor means which are closest to new data point, and then the new query data point will belong to the category 1 otherwise it will belong to category 2 (black data points)

Once the model can do the process that means the model is ready.

### 3.4.11 Advantages and Disadvantages for KNN was given below:

Advantages:

1. It is easy to apprehend and implementing the algorithm.

2. Multi class problem can be solved.

3. When new data comes then depending on the short distance, it can be placed easily where it will belong to.

4. Hyperparameter can be done by Selecting best K value by the elbow method.

Disadvantages**:**

1. It is slow for the large datasets.

2. It has curse of dimensionality means if there are lots of feature so it can’t work well on that time.

3. Data must have to be done scaling.

4. It is very sensitive to outliers.

5. It can’t good deal when there are missing values in dataset.

### 3.4.12 Naïve Bayes

The naïve bayes algorithm works with conditional probability underlying on bayes theorem. Three main factors are involved in the bayes theorem as conditional probability, independent events, and dependent events.

The way the conditional probability can be computed is

P(A|B) =P (A intersection B)/P(B) where P denotes probability and A, and B are events.

Suppose we have a coin which has head and tail then if we toss it either it would be head or tail, so probability hence is fifty percentages of each phenomenon.

If we toss two coins so any possibility can take place as like these samples {hh,ht,th,tt} , both are outputs independent two each other not dependent. That is called independent probability.

An instance can be given for dependent probability is that let we have 5 marbles where 2 is black and rest of are red colour. Now if we say the probability of picking up 1 black marble that would be 2/5. After that, if we want to pick up again 1 black marble from the 4 marbles (1 is black and rest 3 are red) not 5 marbles then the probability would be 1/5 so it is seen distinctly that last probability depends on the earlier probability and it’s decreasing, thus we can say it is dependent probability.

### 3.4.13 Pros and Cons for Multinomial Naïve bayes

**Pros naïve bayes:**

1. prediction performance is seen in real time.

2. very efficient in large datasets.

3. it is not sensitive to irrelevant feature.

4.multi class classification can be done

**Cons naïve bayes:**

1. it assumes all features are independent so it’s a limitation

2.estimation is not always trustworthy.

### 3.4.14 Support Vector Classifier

Support vector machine is a supervised machine learning algorithms which can be used for both as regression and classification problem.

It works on those data are linearly separable that means data can be categorized by the straight line even it can work non-linear data. It can good perform on mix data and classifies impeccably multi class data.

In svm, data points are linearly separable by hyper plane, and it also creates two marginal lines, and the lines will be having some distance then it will be easy for the classification of data points.

Two parallel planes go through the nearest data point where one goes towards the positive point, and another goes towards the negative point.

### 3.4.15 Pros and Cons for SVM

**Pros of svm:**

1. It works on high dimension data as 2D or 3D

2. It is best for classification algorithms owing to it creates a hyperplane which separates data points even though besides creating hyperplane, it also creates two parallel place which help to separate data spotlessly.

3. Outliers can’t impact strongly.

**Cons of svm:**

1. It is very slow for large dataset. The researcher had it while training model.

2. It doesn’t perform good for the overlapped classes

3. lots of scrutinization for the hyperparameter tuning specially for the C and gamma values.

### 3.4.16 Gradient Boosting

Gradient boosting classifier is a group of machine learning algorithms which combines multiple weak learner’s models to a single robust prediction model. Hence also is used decision tree principles like outcome comes from multiple decisions not from a single decision tree.

Gradient boosting is boosting technique of ensemble method which is a sequential method.

### 3.4.17 Advantage and Disadvantage of Gradient Boosting

**Advantage:**

1. It can give more accurate performance

2. Not sensitive to outliers

3. Outperform on large scale dataset

4. Robust for lessening bias

5. Capable of tackling complex dataset

6. It maintains sequential order as thereby. Only one tree runs at a time and handles errors of previous trees.

**Disadvantage:**

1. It can’t be capable of handling noisy data

2. Boosting tree may overfit when noisy values have in dataset

### 3.4.18 AdaBoost Classifier

The algorithm is known also adaptive boosting technique in machine learning which is also a boosting technique one of the ensemble methods.

As it has been mentioned earlier that boosting technique works mainly follows sequential order. When the algorithm builds a model then it gives same weights to data points. After that, those which data points are wrong classified, it assigns maximum weights to them. Now in the next model which data points have maximum weights the algorithm emphasises them more.

A figure was given below which depicts how AdaBoost classifier algorithm works internally.

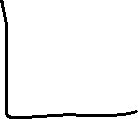
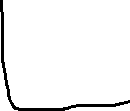


Fig 5: How Adaboost classifiers algorithm works

In the above drawn figures 5, which explain that the first model classified two positive (+) data points correctly and rest 3 positive points are wrongly classified and then the model 2 the wrongly classified data points will get higher weights and it can be seen the model boundary was shifted to the right but still the model is not perfect owing to 3 negative (-) data points are wrongly classified. Now the next model the wrongly classified data points will give more importance means higher weights then in model 3 the 3 negative (-) data points and the iteration will be going on until purely classified.

That’s why it is said that a single decision model is weak learners now we must take all weak learners weighted average and build a final model which will be the strong model with the mean of weighted.

### 3.4.19 Ethical issue

Containing data has no human issue thereby if it is expressed by any means then someone personally won’t be harassed or faced any deleterious issue. This dataset is merely pertained to environmental so there is no possibility that people will be sufferer or loser will somehow. So, it is evidently said that there is no legal issue by any chance.

### 3.4.20 Legal issue

From where the datasets have been taken in which there are no stereotypes to import data or any copyright issues. It can be said it is an open-source data website sono needs to be taken any consent and there is no legal issue existence.

### 3.4.21 Professional issue

The research work was done solely by researcher’s own contribution so, there is no caption to be plagiarism and copied from someone resource and no copyright issue apart from. To be researched, the insights had been taken many sources which were merely free of sources that’s why it is thought that there is no professional issue.

### 3.4.22 Commercial issue

The project was done merely for research purpose so there is no commercial issue involved However, it is fully accessed for any research purpose or study purpose.

# Implementation details

## Dataset collection

To be implemented a data science project the foremost requirement is that to be compiled a quality dataset that means the majority percent of data will be meaningful so, according to the definition the researcher has compiled two datasets from the most resourceful source is Kaggle which are pertinent with the environment.

## 4.2 Imported necessary libraries

Those libraries are necessary these were imported from scikit learn including NumPy, Pandas, Seaborn, matplotlib which is most efficient tool for data science.

## 4.3 Datasets loading

By utilizing pandas, the datasets were loaded with setting options to see all columns and rows by displaying.

Graphical user interface, text, application, email

Description automatically generated

Fig 6: Datasets loading

In the above figure 6 says that datasets loaded by using pandas.read\_csv() function and to be seen whole columns and rows a function namely is set\_option was used.

## 4.4 About data

Due to deficiency the arget column’s data for the first dataframe as data.csv that’s why the second dataset was used in this project however, both datasets are incredibly similar as to information.

The first dataset has 435742 data points and 13 attributes and the second one has 34186 data points and 21 columns. Both datasets have one dependent column as pm2.5 which are categorical data.

## Columns adding to df

Five columns have been added from df1 to df to be compared which dataframe columns have more meaningful data as reason was mentioned earlier that in the df.csv dataset has the identical columns, but problem is the dependent column where contains 9314 data points only not null scale of 435742 so there is huge difference. Figure 7 was given below which says adding columns.

Graphical user interface

Description automatically generated with medium confidence

Fig 7: Columns adding to first dataset

## Final df

Having been added five new columns to the df.csv now there are two dependent columns however, they contain same information, and both are categorical data. We know that as data more meaningful that means less numbers of zeros and null values, the models will get trained properly as well as accuracy will be incredibly high.

To be kept the dataset more sufficient the researcher dropped all null values obtaining PM2.5 as a subset where contains roughly values as 34186 rows and was prepared a final dataset to analyse data and to be trained data.

Graphical user interface, text, application

Description automatically generated Fig 8: Final dataframe head

In the above figure 8 says about final dataframe head.

## Data cleaning

In the final dataframe there are 34186 data points, and 17 rows contain. The figure 9 was given below shown the volume of dataset.

A picture containing rectangle

Description automatically generated

Fig 9: New dataset volume

From the dataset, three columns as sampling\_date, state, agency, and type were used predominantly for the data analysis and visualization purposes. What’s more, the six features like so2, no2, rspm (respirable suspended particulate matter), spm(suspended particulate matter), CO(carbon monoxide) and eventually Ozone had been used for the independent features and PM2.5 attribute used as a dependent column. Even if it shows 17 features now but those which were redundant features had been dropped them later.

### **Columns renaming**

Hence above-mentioned CO, Ozone and PM2.5 which columns name were renamed from the Days\_CO, Days\_Ozone and Days\_PM2.5.

Graphical user interface, text, application, email

Description automatically generated

Fig 10: Renaming columns

In the above the figure 10 says that those columns were added to first dataset, they had been renamed.

### **Target column labelling**

At the very beginning, the data type of dependent column was discrete data means that is numerical type of data which is fully concreted data and no fractional number data. Subsequently, underlying on dependent feature PM2.5 a function was built with conditions applying as if x<=50 then the AQI label will be **good**. Here x means PM2.5. Then if x>50 and x<=100 it returns **moderate** after that if x>100 and x<=200 it returns **poor** air quality. Subsequently, if x>200 and x<=300 then it returns **unhealthy label** and if x>300 and x<=400 it classifies as **very unhealthy** and eventually, if air quality range be x>400 then it returns **hazardous** label. A new column was added in dataframe named is AQI\_Range.

A picture containing text

Description automatically generated

Fig 11: Categorised target column

In the above figure 11 depicts that the target column was categorised as good, moderate, bad, unhealthy, very unhealthy.

### **Ordering target column**

The AQI\_Range column was ordered to a new column named AQI\_RANKING later to be apprehended easily.

At first, using the python built-in function dictionary () was made a dictionary in which the keys were numbered 1 to 6, and values obtained as Good, Moderate, Poor, Unhealthy, very unhealthy, and hazardous. Having been made a dictionary that was mapped with the AQI\_Range column.

Graphical user interface, text, application, email

Description automatically generated

Fig12: Mapped target column

In the above figure 12 which depicts that after being categorised target column then they were ordered by 1,2,3,4,5 for good, moderate, bad, unhealthy, very unhealthy accordingly.

### **Describing dataframe**

Text, table

Description automatically generated

Fig 13: Describing datarame

In the above figure 13, we can have a look that for all columns after 75% percentile to maximum range there is huge difference and among them spm feature is on the pinnacle stance and it also noteworthy that for all of the attributes the standard deviation is so high which means that the values is so far from average values as well as it can be said data is utterly scattered.so it is said that there are huge outliers existing in each column.

### **Unique values**

**Text

Description automatically generated with medium confidence**

Fig 14: Unique values

Above figure 14 tells us that out of the 34186 data points how many unique values have in each column.

### **Checking duplicate values**

Chart

Description automatically generated with medium confidence

Fig 15: Duplicate values

It is seen there are

no duplicated values in all the attributes.

### **4.7.7 Columns present in dataset**

**Graphical user interface, text, application

Description automatically generated**

Fig 16: Columns of df

In the above figure 16 tells us that these columns have now in the dataset.

### **Checking NaN**

**Table

Description automatically generated with medium confidence**

**Fig 17**: Null values contain each column

It is distinctly seen from figure 17 that the significant percentage null values have in three features including, spm, ageancy, stn\_code and substantial null values have in rspm, location\_monitoring\_station, type, so2, no2. Albeit those columns are redundant so that these were dropped thereupon.

### **Percentage of NaN**

**Graphical user interface, table

Description automatically generated**

Fig 18: Percentages columns NaN

In the above figure 18 means that the percentage of each column’s null values.

### **Percentages of missing data**

Graphical user interface, application

Description automatically generated

Fig 19: each Column’s missing value

From the above figure 19 says the identical of null values percentages, here also the majority percentages missing values in three columns as spm, agency and stn\_code which were near to 50% percentage.

### **Removing redundant columns**

**Text

Description automatically generated**

Fig 20: Dropped unnecessary columns

The above figure 20 depicts that the above-mentioned columns were redundant owing to the same information contained by other columns in dataset, so these had been eliminated from dataset to abstain from curse of dimensionalities.

### **Feature guidance details**

Having obtained meaningful data after that, the 5 redundant columns as stn\_code, samplng\_date, location\_monitoring\_station, date, agency they were dropped from dataset as well it is effective for having good accuracy and to stay away the curse of dimensionality.

In **state** column mainly depicts that from which states of **India’s** data was compiled that are obtained now in dataset for analysis and visualization purpose.

In Type variable contains which type of area it is, as it is rural area, residential area, or industrial area.

To be polluted air the main contribution keeps by **SO2**(sulphar di oxide), **NO2**(nitrogen di oxide), **RSPM** (**respirable suspended particulate matter)** which is very pernicious for people health, and it can penetrate deep into lungs and damage easily and **SPM (Suspended Particulate Matter)** it is mainly tiny solid particle can penetrate through nose and damage respiratory system and mainly generated by the industry combustion process.

The utmost two features as CO gas and Ozone(O3) which are mainly gas and generated by combustion of vehicles fuels and industrial combustion. Furthermore, CO is a toxic gas which is very familiar as self-killing gas, and the Ozone gas is also grim for children breathe system when they breathe it then their lung system gets attacked mortally.

Depending on six attributes the PM2.5 values has placed with range. By the PM2.5 values that can be measured air quality for a specific region that by which particle in the air is being polluted so government or any other organizations can take strict measurements to lessen the amount of pollution for air.

The rest of variables are Index ([ 'state', 'type', 'so2', 'no2', 'rspm', 'spm', 'pm2\_5', 'date'], dtype='object')

The pm2.5 preponderantly dependent column of dataset.

The details insights as to pollution contain in six significant variables as So2, no2, spm, rspm,CO,Ozone and AQI\_range column created based on PM2.5 level applying a function with conditions which was mentioned in the above. After that, feature AQI\_RANKING had been created which is considered the ultimate column for dependent column and others like PM2.5 and AQI\_range columns had been dropped.

Mainly above six columns have been prepared for being trained ML models as independent variables and AQI\_RAKNING was obtained for dependent variable.

## **Data analysis and Visualization**

In this part how data behaves that was done for the project.

The two parts, data analysis and visualization were completed elaborately to be obtained the conspicuous idea regarding of data. Both parts were completed together for better understanding where data analysis had been analysed by using univariate analysis which mainly works with a single feature and bivariate as well as multivariate analysis. The bivariate analysis gives about two variables insights and multivariate analysis predominantly gives the throughout connotations of data. All are discussed below with plotted images.

Graphical user interface, text

Description automatically generated

Fig 21: values of state column

It is seen from the above figure 21 that there were three states as Andhra Pradesh, Assam, and Arunachal Pradesh where Andhra Pradesh is holding the majority.

### **Plotted state column**

**Chart, waterfall chart

Description automatically generated**

**Fig** 22: state column visualization

As we can see from above figure 22 that Andhra Pradesh frequency is highest and Arunachal Pradesh frequency is lowest.

### **Types of areas**

**Text

Description automatically generated**

**Chart, bar chart

Description automatically generated**

**Fig 23:** Areas of provinces

We can have a look from above figure 23 that residential and rural areas numbers are high and industrial areas are less than residential areas also sensitive areas.

### **Pollution controlling agencies**

Text

Description automatically generated

Fig 24: Name of agencies

We can see from above figure 24 that there are plenty of agencies work for tackling air pollution where highest number of agencies situated at the Andhra Pradesh then Assam and a smaller number of controlling agencies situated at Arunachal Pradesh.

### **States vs so2**

**Chart, bar chart, box and whisker chart

Description automatically generated**

Fig 25: SO2 measurement in states

Hence, we can notice that the quantities of so2 in air is highest in Andhra Pradesh than rest of states and Assam also near to Andhra Pradesh. Furthermore, a small amount so2 is in Arunachal Pradesh.

### **No2 vs state**

**Chart, bar chart, box and whisker chart

Description automatically generated**

**Fig 26:** NO2 measurement in states

Hence, we can have a look from figure 26 that the quantities of no2 in air is the highest in Andhra pradesh than rest of states and Assam also near to Andhra Pradesh. Furthermore, a small amount no2 is in Arunachal pradesh.

### **Rspm vs states**

**Chart, bar chart

Description automatically generated**

**Fig 27:** RSPM measurement in states

Hence, we can see from figure 27 that the quantities of RSPM in air is highest in Assam than rest of states as Andhra Pradesh and Arunachal Pradesh are the identical levels but small amount from Assam state.

### **Spm vs states**

**Chart, bar chart

Description automatically generated**

Fig 28: SPM measurement in states

From the above figure 28, we can see that the quantities of SPM in air is highest level in Andhra Pradesh then Assam has the second highest level spm. Furthermore, Arunachal Pradesh has no level of suspended particulate matter in air.

### **State vs PM2.5**

**Chart

Description automatically generated**

**Fig 29:** PM2.5 level in states

From the above figure 29 it can be said that the particulate matter 2.5 level is in the higher amount in both states, including Assam and Arunachal Pradesh and a small amount level is in the Andhra Pradesh.

### **Type vs PM2.5**

**Chart, bar chart

Description automatically generated**

**Fig** 30: PM2.5 level in different areas

From the above figure 30 it can be said that the particulate matter 2.5 level is in the highest amount in three states, including residential, rural and others, industrial areas, and sensitive area.

### **So2 vs Pm2.5**

**Chart, scatter chart

Description automatically generated**

Fig 31: Relation between so2 and PM2.5

From the above figure 31, we can see one scatter plot was plotted where in x axis is so2 which is independent feature and in y axis is PM2.5 which dependent feature. The plot depicts how two variables are correlated to each other, so it is vividly said that the plot doesn’t indicate normally distributed data.

### **No2 vs Pm2.5**

**Chart, scatter chart

Description automatically generated**

Fig 32: Relation between no2 and PM2.5

From the above figure 32, we can see a scatter plot was plotted where in x axis is no2 which is independent feature and in y axis is PM2.5 which dependent feature. The plot depicts how two variables are correlated to each other, so it is vividly said that the plot doesn’t indicate normally distributed data.

### Rspm vs PM2.5

**Chart, scatter chart

Description automatically generated**

Fig 33: Relation between rspm and PM2.5

From the above figure 33, we can see one scatter plot was plotted where in x axis is rspm which is independent feature and in y axis is PM2.5 which dependent feature. The plot depicts how two variables are correlated to each other, so it is vividly said that the plot doesn’t indicate normally distributed data rather it is skewed data can be said.

### **Spm vs PM2.5**

Chart, scatter chart

Description automatically generated

Fig 34: Relation between spm and PM2.5

From the above figure 34, we can notice a scatter plot was plotted where in x axis is spm which is independent feature and in y axis is PM2.5 which dependent feature. The plot depicts how two variables are correlated to each other, so it is clearly said that the plot doesn’t indicate normally distributed data rather it is somewhat scattered and has outliers.

### CO vs PM2.5

**Chart, scatter chart

Description automatically generated**

Fig 35: Relation between CO and PM2.5

From the above figure 35, we can notice a scatter plot was plotted where in x axis is CO which is independent feature and in y axis is PM2.5 which dependent feature. The plot depicts how two variables are correlated to each other, so it is clearly said that the plot doesn’t indicate normally distributed data. There is a negative correlation between CO and PM2.5 which is seen distinctly in the above plotted figure.

### **Ozone vs PM2.5**

**Chart

Description automatically generated**

Fig 36: Relation between Ozone and PM2.5

From the above figure 36, we can notice a scatter plot was plotted where in x axis is Ozone which is independent variable and in y axis is PM2.5 which dependent variable. The plot presents how two variables are correlated to each other, so it can be clearly said that the plot doesn’t indicate normally distributed data rather it carries the meaning that there is a negative correlation between Ozone and PM2.5 variables which is seen distinctly in the above plotted figure.

## **Feature engineering**

In this feature engineering there are lots of thing worth doing as columns imputation for null values, data scaling, data transformation if needs, outliers’ detection and if needs, to be removed, data resampling if data is not balanced, feature selection. All steps were implemented for this project and will be elucidated below. The researcher reckons it needs to mention that all steps maintained for solely proper implementations, however all processes are not necessary for this project just being confirmed all procedures were done.

### **NaN values handling**

As we have seen earlier that there are numerous null values in four columns as so2, no2, rspm and spm among them the spm has the highest number of null values so we need to handle that by column imputation which is fixed number or we can handle that to fill with the mean, median or most frequent value so we need to check first whether there are any outliers in columns.

We know that if there are outliers in columns and data is skewed then we need to apply median or mode technique to fill null values. At first just have a look to description of dataframe.

Table

Description automatically generated

Fig 37: Interpretation data

We can see from the above figure 37 that from every column’s third quartile, or 75% percentile value is far from the max range of value so it can assume that there are outliers in features, and we can apply median or mode to fill null values.

Now it needs to be checked that data is skewed or not. If data is skewed, then median is preferred to handle null values and if data is not skewed rather normally distributed then be applied mean value is compatible.

Let’s check data is skewed or not by plotting distribution plot for each column.

### **For so2 feature:**

Graphical user interface

Description automatically generated

Fig 38: Histogram for so2

It is explicitly seen from above figure 38 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so it can be said indisputably that there are outliers. Right skewed looks like below

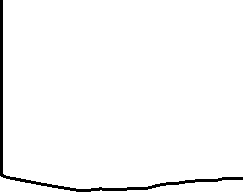
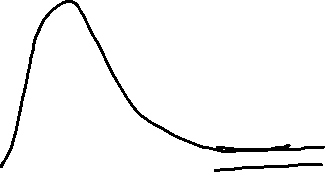
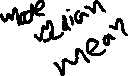


Fig 39: how looks like Right skewed data

In the above figure 39 was shown how right skewness looks like where mean stance is on the right of summit then the median is on the left and the mode is on the left pinnacle position from the mean.

Graphical user interface, text, application, table

Description automatically generated

Fig 40: Box plot for so2

From the above figure 40, it is said that there have outliers in that column where 75% percentile is 8 and max range of value is above 200.

### **For no2 column:**

Graphical user interface

Description automatically generated with medium confidence

Fig 41: Histogram for no2

It is explicitly seen from above figure 41 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so it can be said indisputably that there are outliers.

Graphical user interface, text, application, email

Description automatically generated

Fig 42: Box plot for no2

From the above figure 42, it is said that there have outliers in that column where 75% percentile is 26 and max range of value is above 334.

### For Rspm column:

Chart

Description automatically generated with medium confidence

Fig 43: Histogram for rspm

It is explicitly seen from the above figure 43 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so it can be said indisputably that there have outliers in rspm column.

Table

Description automatically generated with low confidence

Fig 44: Box plot for rspm

By using the boxplot from seaborn, above figure 44 was drawn. It can be said that there have outliers in that column where 75% percentile is 100 and max range of value is above 790.

### **For spm column:**

**Chart

Description automatically generated**

Fig 45: Histogram for spm

It is explicitly seen from above figure 45 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so it can be said indisputably that there have outliers in that column.

Chart, box and whisker chart

Description automatically generated

Fig 46: Boxplot for spm

By using the boxplot from seaborn, above figure 46 was drawn. It can be said that there have outliers in that column where 75% percentile is 236 and max range of value is above 1300, so it needs to be handled.

### **For CO column:**

**Graphical user interface

Description automatically generated with medium confidence**

**Fig 47:** Histogram for CO

It is explicitly seen from above figure 47 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so it can be said indisputably that there have extreme outliers on that column.

Chart

Description automatically generated with medium confidence

Fig 48: Boxplot for CO

By using the boxplot from seaborn, above figure 48 was drawn. It can be said that there have outliers in that column where 75% percentile is 0 and max range of value is above 366, so it needs to be handled. In this feature that there are no null values but have numerous zeros on that column which are deemed actual values so no needs to be handled null values or outliers.

### **4.9.7 For Ozone feature:**

**Chart

Description automatically generated with medium confidence**

Fig 49: Histogram for Ozone

It is explicitly seen from above figure 49 that the data is not normally distributed and right skewed that means data is positively skewed or most data fall in the right side. It has a long tail on the right side, so seemingly it can be said that there are outliers since kurtosis is high at the beginning.

Chart, histogram

Description automatically generated

Fig 50: boxplot for Ozone

It is said from the above figure 50 that there are no outliers visible but looks like the standard deviation is extreme high between 50th percentile and 75th percentile data and they are 167 and 235 respectively so data transformation would be the most dexterous way to be tackled high deviation from that column then standard deviation incredibly close to mean values.

As we talked about regarding outliers and skewness in data and we have seen that in every column has outliers except CO and Ozone columns and data is skewed on the right side so median would be discreet way to be handled null values those which features have null values. Now null values will be handled by median.

Graphical user interface, text, application, email

Description automatically generated

Fig 51: Null values handling by median

From above figure 51 says that Having been filled null values in four attributes by median we can check that there are any null values remaining in column.

Graphical user interface, text, application

Description automatically generated

Fig 52: Checking null values

It is seen from the above figure 52 that there are no null values in any column except type column which mainly used for data analysis so either it can be dropped or remained, and it won’t affect while training owing to it will not feed to models.

## **Data Splitting**

For having better performance from model, the data splitting is significant. Normally data can be divided in two segments where one is train data, and another is test data with obtaining 80% data for train which is used for model training purpose and 20% data is kept for test data which is mainly used for prediction purpose. The ratio of data division, for train data is 80% and for the test data is 20%, however it can be 70% and 20% for tarin and test data.

From train data the machine learning model gets trained first that to learn what should be the aftermath to predict it. After that, with a small fraction of data to be given for test purpose for that whether the ML model can give right output from what it has learned before from train data.

Partitioning the data is done by using train\_test\_split() function which splits data into train and test. In the beginning, the data needs to be divided into independent features(X) and dependent feature or labelled column(Y) then the dataframe gets partitioned into four parts as X\_train, X\_test,Y\_train and Y\_test from where X\_train and Y\_train are predominantly used for fitting the model.

Text

Description automatically generated

Fig 53: Data partitioning

We can take a notice to above figure 53 that the data was partitioned into X2 and Y2 first then divided into X\_train2, X\_test2, Y\_train2 and Y\_test2 where 30% had been reserved for the test purpose and 70% data for train purpose and random\_state is 10 means data gets shuffled randomly while training data in model.

## **4.11 Data Scaling**

### **4.11.1 Scaling for train data**

We have seen earlier and interpreted elaborately that the significant number of features have outliers and data is skewed on the right side, so considering it after that, the scaling technique robust scaler was used at first to scale all the features in the same scale and to be checked how data looks like after applying the scaling method. The scaling method was applied for both train and test data individually.

Table

Description automatically generated

Fig 54: Train data scaling

We can see from above figure 54 that all columns’ ranges have become smaller. Robust scaler principle is very close to Min max Scaler, but min max scaler works with minimum and maximum range which is [0,1] or [-1,1] and effective for the small variance and if the data doesn’t imitate normal distribution. Robust scaler works with the Inter quantile Range (IQR) that means 25th percentile and 75th percentile and it is hugely robust to outliers as well as it scales data down.

After having been scaled down data by robust scaler let’s check how data looks like for each feature by drawing histogram below.

Graphical user interface

Description automatically generated

Fig 55: After scaling so2 histogram

A picture containing graphical user interface

Description automatically generated

Fig 56: After scaling no2 histogram

Chart, histogram

Description automatically generated

Fig 57: After scaling rspm histogram

Text

Description automatically generated with medium confidence

Fig 58: After scaling spm histogram

Graphical user interface

Description automatically generated

Fig 59: After scaling CO histogram

Chart, histogram

Description automatically generated

Fig 60: After scaling Ozone histogram

In the above figures as 55,56,57,58,59,60 was plotted for all independent features after data scaling, but very interestingly all columns’ data were the identical as before scaling data means as data was skewed and not normal distributed before scaling, so after scaling data characteristics were the same.

### **4.11.2 Scaling for test data**

**Text, table

Description automatically generated**

Fig 61**:** Scaled test data

In the above figure 61 shows that after scaling test data the first 5 rows and all independent columns.

## **4.12 Data transformation**

as like train data the test data was scaled by robust scaler and data looks like the identical after scaling also which plotted figure were shown in the above the same applies for test data.

Since our data doesn’t follow gaussian distribution so we need to transform our data with using logarithm transformation as it is incredibly effective for not normal distribution data to normal distribution data.

It is examined that if data is transformed after scaling, then a problem emerges that is range infinity problem that’s why without scaling train and test data were used for data transformation. Subsequently, the min max scaler had been used before creating model.

Now we will see that after applying logarithms distribution how data behaves.

### **4.12.1 For training data**

Hence will be shown before and after data transformation. By creating Q-Q plot we will check how data looks like before and after with distribution plot and probability plot for each.

Chart, line chart

Description automatically generated

Fig 62: Before transformation so2 col.

It is noticeable in the above figure 62 that the column’s data is not normally distributed before applying logarithm transformation. So, let’s check what happens after applying logarithm transformation.

Chart, histogram

Description automatically generated

Fig 63: After transformation so2 col.

In the above figure 63 it looks like gaussian distributed data after applying logarithm transformation the data has changed to normal distribution.

Chart, line chart

Description automatically generated

Fig 64: Before transformation no2 col.

In the above figure 64 says that the data doesn’t follow gaussian distribution as we can see from above histogram and plot.

Chart, histogram

Description automatically generated

Fig 65: After transformation no2 col.

From the above figure 65 it looks like after applying logarithm transformation the data has changed to normal distribution.

Chart

Description automatically generated

Fig 66: Before transformation rspm col.

Before transformation the rspm column’s data doesn’t follow gaussian distribution we can see from the above figure 66.

Chart, histogram

Description automatically generated

Fig 67: After transformation rspm col.

Data has transformed outstandingly after having been applied logarithm transformation for the feature rspm which we can see from above figure 67.

Chart

Description automatically generated

Fig 68: Before transformation spm col.

Before transformation the column of spm data does not follow gaussian distribution. If we see to the probability plot from above figure 68, we can apprehend it.

Chart, histogram

Description automatically generated

Fig 69: After transformation spm col.

We can see from above figure 69 that data is roughly following gaussian distribution, but data kurtosis is substantial which means still likely to have lots of outliers.

Chart, line chart

Description automatically generated

Fig 70: Before transformation CO

From above figure 70, it looks data doesn’t follow normal distribution rather it has lots of kurtosis with heavy right tail. Let’s see what happens after applying logarithm transformation.

Chart, line chart

Description automatically generated

Fig 71: After transformation CO

We can look at the above figure 71 that data has not shifted after applying logarithm transformation. It is identical as same before.

Chart, histogram

Description automatically generated

Fig 72: Before transformation Ozone

From above figure 72, we can say it that at the beginning the feature has long kurtosis. After that, data follows roughly gaussian distribution and at last data has a small kurtosis we cansee from figure.

Chart, histogram

Description automatically generated

Fig 73: After transformation Ozone

We can notice from above figure 73 that data has not shifted after applying logarithm transformation. It is identical as same before. It is seen that the data somewhat gaussian distributed.

### **4.12.2 Test data Transformation**

As like the train data transformation by using logarithm transformation, for the test data the logarithm transformation was used, and all figures were identical as like train data.

## **4.13 Data Resampling**

Checking target feature if can be got that all classes’ data are not the same ad there is huge difference one class to other class then the data needs to be resampled. It is compatible for classification problem when there is no equality among classes then the problem emerges.

The project dataset has a data imbalanced issue.

### **4.13.1 Data discrepancies plot**

Text, table

Description automatically generated

Chart, bar chart

Description automatically generated

Fig 74: Data discrepancies histogram

We can see from the above figure 74 that in the target column there are five classes as class 1 has 61.78% data, class 2 has 15.39% data, class 3 has 12.31% data, class 4 has 6.38% data and class 5 has only 4.12% data so it can be that is clearly imbalanced dataset and that needs to be handled.

As we know that imbalanced dataset data is extremely skewed data and caused for poor performance.

Imbalanced dataset can be handled many ways as applying oversampling which works with majority percent data by randomoversmpler() , undersampling which loosens data as a result data is lost so that’s not suggested often and SMOTE also very known technique to handle data discrepancy problem.

Hence was applied SMOTE (synthetic minority oversampling technique) method which predominantly works with minority class data and increasing them to even out to majority class.

In this project we have experienced a very interesting thing that after balancing data to use SMOTE method and trained model with balanced data our accuracy has dwindled a small fraction of than without balanced data that’s why all models were implemented without using balanced data.

Just to be confirmed the imbalanced data was balanced by SMOTE and had been trained in two models as Linear regression and Decision Tree classifier and had a bit dismal performance than without resampling data.

### **4.13.2 Data balanced plot**

Chart, bar chart

Description automatically generated

Fig 75: Data balanced histogram

We can it from above figure 75 that after having been balanced all classes data points which are now the same for all classes and the target column Y\_test2 data points shape have shifted from 10256 to 31455.

After having been balanced data after that, the two models as Linear regression and Random Forest have got trained and predicted for making comparison that before balancing and after balancing accuracy, however with balancing data the accuracy had for Linear Regression they were 43% for both train and test which is deemed not good performance for linear regression and without balancing the scores were 61% for train and 60% for test which was better performance comparatively.

### **4.13.3 Accuracies for LR with balanced data**

Graphical user interface, text, application, email

Description automatically generated

Fig 76: With resampling scores for LR

In the above figure 76 says that with balanced data the accuracies for both train and test were got 43%.

### 4.13.4 Accuracies for LR with not balanced data

Text

Description automatically generated

Fig 77: Without resampling score for LR

In the above figure 77 says with fitting imbalanced data the accuracies for both tarin and test were got 61% and 60% respectively which are more high scores than with balanced data.

### 4.13.5 For decision tree the accuracies by balanced data

Text, letter

Description automatically generated

Fig 78: With balancing for DT

In the above the figure 78 says that after balancing data the scores were 65% and 62% for both train and test respectively for decision tree classifier.

### 4.13.6 Accuracies without balancing for DT

Text, letter

Description automatically generated

Fig 79: Without balancing for DT

From above figure 79 says that without balancing data the scores for train and test data were 66% and 65% accordingly which are a bit better from with balanced data scores.

So, it can be said just for this project it may prudent decision to train model without data resampling, however it’s not suggested in the long run for any project without balancing data to train model otherwise it may be the cause for the dismal performance of model.

## **4.14 Outlier detection**

At the very beginning we had lots of outliers in each column. To curtail these, we have applied logarithm transformation to transfer data from skewed to gaussian distribution and we have been able to do it.

For outlier detection the most used technique is IQR or Inter Quantile Range which preponderantly works with 25th percentile and 75th percentile.

If data follows gaussian distribution, then we can apply gaussian range that we can compute positive 3 standard deviation from mean as well as negative 3 standard deviation from mean which are also known as upper limit and lower limit respectively.

Which values are more than upper limit that are outliers, and which are less than lower than that are considered also outliers.

It basically follows the central limit theorem (CLT) that is mean+1st standard deviation, mean+2nd standard deviation and mean+3rd standard deviation which covers 99.7% data as the same for the negative side like mean-1st standard deviation, mean-2nd standard deviation and mean-3rd standard deviation. We can draw a pattern below for clear conception

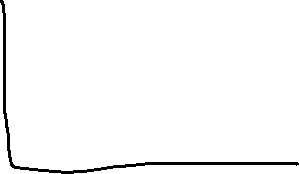
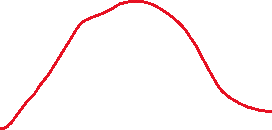
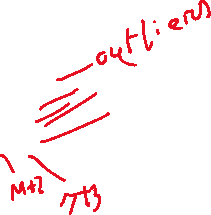
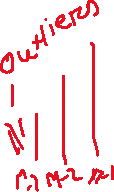


Fig 80: Conception about computing outliers with 3 std.

In the above the figure 80 says from the mean value to up to 3 standard positive deviation the data are not outliers and on the negative side from mean to up to 3 standard negative deviation the data are not outliers.

Having been Computed 3 standard deviation for upper limit and lower limit the outliers were figured out for each feature.

### **4.14.1 Outlier ranges for so2 column**

**Chart, box and whisker chart

Description automatically generated** Graphical user interface

Description automatically generated with low confidence

Fig 81**:** Outlier detection for so2

We can see from the above figure 81 that the upper limit of feature so2 is 3.34 and the lower limit is 0.25, so after 3.34 any values will be considered as outliers and as the same below 0.25 will be outliers.

Now we will consider that we should remove outliers or keep owing that outlier that doesn’t mean always has to be removed sometimes outliers are very significant for statistical analysis so we can check the range between 75th percentile and maximum range from boxplot. After that, we can reach a conclusive decision whether they should remove or not. If difference between third quarter data and maximum range of data is very small, then we should be kept otherwise should be eliminated.

For so2 column the 75th percentile is seen that is 2.07 and maximum range of column is 5.42 so we can keep it easily.

### **4.14.2 Outlier ranges for no2 column**

**Chart

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Fig 82:** Outlier detection for no2

We can see from the above figure 82 that the upper limit of feature no2 is 4.35 and the lower limit is 1.39, so after 4.35 any values will be considered as outlier and as the same below 1.39 will be outlier.

we can check the range between 75th percentile and maximum range from boxplot. After that, we can reach a conclusive decision whether they should remove or not.

For no2 column the 75th percentile is seen that is 3.25 and maximum range of column is 5.58 so we can keep it easily since very small difference between them and it won’t affect to model performance.

### **4.14.3 Outlier ranges for rspm column**

**Chart

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Fig 83:** Outlier detection for rspm

We can see from the above figure 83 that the upper limit of feature rspm is 5.81 and the lower limit is 2.72, so after 5.51 any values will be considered as outlier and as the same below 2.72 will be outlier.

For rspm column the 75th percentile is seen that is 4.58 and maximum range of column is 6.67 so we can keep it easily since very small difference between them and it won’t be impediment to model performance.

### **4.14.4 Outlier ranges for spm column**

**Chart, box and whisker chart

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Fig 84:** Outlier detection for spm

We can see from the above figure 84 that the upper limit of feature spm is 6.32 and the lower limit is 3.88, so after 6.32 any values will be considered as outlier and as the same below 3.88 will be outlier.

For spm column the 75th percentile is seen that is 5.14 and maximum range of column is 7.17 so we can keep it easily.

Due to infinity values the two features as CO and Ozone have not been able to be shifted data transformation for that why the outlier detection was not taken place for them.

## **4.15 Features selection**

Feature selection is method through which we can select those which are important feature for model.

Many feature techniques are used to select important features, including Pearson correlation coefficient, mutual information gain, chi square test, extra tree classifier.

Hence was used two feature selection techniques as they are Pearson correlation coefficient and Extra tree classifier.

### **4.15.1 Features selection by Pearson**

Let’s explain what their principles are. Pearson correlation coefficient works with independent features to give information that how features are correlated to each other or how many percentages they are dependent to each other. As percentage is high for a feature means it is incredibly correlated to others. depending on threshold value the Pearson correlation coefficient works as like if threshold value is selected 0.6 or 60% and if any feature correlation coefficient is not above 60% then it is considered the feature is not highly correlated to others. We can have a look from figure below

Chart, treemap chart

Description automatically generated

Fig 85: Correlation metrics heatmap

We can see from the above figure 85 that where was shown what is the correlation between two features and colour bar explains that by what colour means what is the percentage.

We can see that CO and Ozone features have negative correlation to other features means if independent feature increases, then target feature decreases.

We can select those which are highly correlated feature by creating a function which iterates through each column by looping and values contain in a variable which is a set () because set () doesn’t contain any duplicate values.

Graphical user interface, text, application, email

Description automatically generated

Fig 86: Function for checking correlation

We can say from above figure 86 that a function was built for selecting highly correlated features relying on threshold value then as parameters had been passed the training data X\_train2 and threshold value is 0.6 so we can see that no feature here which highly correlated.

### **4.15.2 Features selection by ExtraTreeClassifier**

Being used another method the feature selection technique was implemented that is ExtraTreeClassifier. It is a tree-based algorithm like random forest classifier and follows principles of random forest, however sampling without replacement for every tree. It works with feature importance by giving value for each column. The higher the score means the feature is more pertinent. We can see a figure below

Graphical user interface, text, application, email

Description automatically generated Chart, bar chart

Description automatically generated

Fig 87: Feature importance by extra tree

We can see from the above figure 87 that the value of feature importance is high for the Ozone then rest of the are rspm, no2, so2, spm and CO.

Since the feature importance of CO is around 7 so we can drop that to avoid noisy data.

## **4.16 Building Models**

### **4.16.1 Normalization**

Doing data scaling should be done always after splitting data into train and test which keeps abstain train data sharing from test data that is called data leakageing.

We did also that by robust scaler in the above and endeavoured to use scaled data for data transformation, however we couldn’t do that due to infinity range error that’s why before creating models we did normalization by MinMaxScaler and scaled all data range 0 to 1 which also the identical to the robust scaler technique that was examined.

Since at the beginning, all features’ data were positively skewed and every feature contained extreme outliers that’s why the robust scaler tried to be used to handle large skewness and outliers, however after data transformation we have seen majority columns have transferred to gaussian distribution as well the variance has lessened from columns so that we used minmax scaler now for the small outliers and variance.

### **4.16.2 Fitting Models**

Since it is a classification-based project so there were used lots of classification machine learning algorithms, including Logistic Regression, Random Forest classifier, Decision Tree classifier, Multinomial Naïve Bayes, Support Vector Classifier, Gradient Boosting Classifier, AdaBoost classifier.

The predominant purpose of using many algorithms is to check that which model gives better performance comparatively from other.

As the details as to models were mentioned in the above from that we have acquired insights of them.

After having been completed all the pre-processing the train and test data ready to be trained models.

Now will be shown how models have given accuracy for both on train and test data.

### **4.16.3 Logistic Regression**

Logistic regression gets trained y training data and generates prediction on this basis of test data. It can categories data and the eventual output would be labelled data.

The algorithm uses sigmoid function to map the prediction values of probabilities and the function range is 0 to 1 within the range it can give the prediction result.

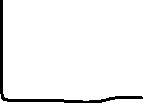
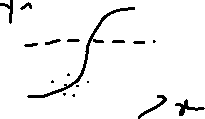


Fig 88: Sigmoid function

In the above the figure 88 says that how sigmoid function works for logistic regression. It maps prediction probabilities between 0 and 1 whatever input is big positive value or big negative value the output will be 0 to 1 always.

Fitting the X\_train2 and Y\_train2 data the model was trained and to be obtained prediction result the X\_test2 data had been used for this algorithm.

We had train and train accuracy by using accuracy\_score function which was imported form sklearn.metrics module. The accuracies are 0.61 for both train and test data that means 60% data classified correctly. To have more idea about classification result we need to see recall, precision, and f1-score from classification report.

First needs to be given a short brief about recall, precision and f-1 score.

### **4.16.4 Performance reports evaluation**

The performance metrics has four parts, including True positive rate (TPR), True negative rate (TNR), False positive rate (FPR) and False negative rate (FNR)

TPR is that out of predictive positive how many actual positive values and TNR is that out of the total predictive negative how many negative values, however the FPR is that false positive divided by total negative and FNR is that out of the total negative how many actual negative values lie. FPR and FNR are also knows as type 1 error and type2 error which are predominantly concerned to be handled.

By computing TPR, TNR, FPR, FNR we can figure out precision, recall and f1 score for better understanding as to model performance.

### **4.16.5 Precision for LR**

Precision is that out of total predictive positive how many actual positive values. We use precision normally to know about negative data information.

Table

Description automatically generated with low confidence

Fig 89: Classification report for LR

In the above the figure 89 says that our model has given precision for class 1 is 0.6 means 60% data could correctly classified by model, class 2 is 0 means no data is correctly classified, class 3 is classified 21% data and class 4 and 5 precisions are also 0 classified no values totally and overall accuracy is 61%.

### **4.16.6 Recall for LR**

From the above figure 89 we can say that recall is that out of the total actual positive how many positive are predicted which is known also TPR.

We can see for the 1st category the recall is 0.98 that means 98% data labelled correctly for this class and then rest of the classes are 0 predicted except class 3 which was predicted 3%.

### **4.16.7 F1 scores for LR:**

From above figure 89 says that F1-score is the harmonic mean of precision and recall. It is a metric which considers both precision and recall for quantifying harmonic accuracy because if precision and recall are 1 then f1 score will be 1.

Since true positive value is 0 for the three classes 2,4 and 5 so their f1 score have become 0. For class 1 the score is 0.76 means 24% data misclassified and class 3, 95% data has misclassified.

### **4.16.8 Hyperparameter Tuning for logistic regression**

Hyperparameter constructs with the utmost parameters values to train algorithm with different values which can help the model to boost the accuracy.

Every model has different kinds of parameters. To regularise model the tuning technique can be used for having better accuracy, however it may be same sometimes.

Hence used GridSearchCV as a hyperparameter technique which selected the best optimal values which were given as **penalty= [l1, l2]** and C=np.logspace(-3,3,7) and 'solver’: ['newton-cg', 'lbfgs', 'liblinear'] and after assigning them into GridSearchCV it returned the best optimal values as Tuned Hyperparameters: {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}. We know that as C value smaller the model would be more regularised.

### **4.16.9 Decision Tree Classifier**

being used the classifier we had the accuracy for both train and test are 98% and 61% respectively which seems as overfitted data, so we need lessen this.

`

### **4.16.10 Performance reports evaluation for DT:**

Table

Description automatically generated

Fig 90: Classification report for DT

In the above the figure 90 depicts that the precision, recall and f1-scores for the decision tree classifier where the highest precision score was for class 1 and lowest for class 5 and the recall and f1-scores were the same for the highest and lowest.

### **4.16.11 Hyperparameter Tuning for DT Classifier**

After having been done tuning the overfitted problem has solved and it has given the accuracy for both train and test were 67% and 66% accordingly. They are considered moderate scores for both.

We know that as max\_depth deeper the model is most likely to get overfitted. The

best optimal values were: {'min\_samples\_leaf': 22, 'max\_features': 14, 'max\_depth': 9, 'criterion': 'entropy'} and best score was 65% so it can be said indisputably that by tuning the high bias converted into low bias successfully.

### **4.16.12 Random Forest Classifier**

For random forest classifier we had the accuracy for both the train and test were 98% and 66% respectively which seems highly overfitted.

In generally when noisy data gets trained to the model and if it is tree-based

algorithm then it depends in the max depth. As max depth deeper for the algorithm

then the overfitting most likely to take place. To tackle we need pruning the trees

and it can be done by selecting the optimal values by hyperparameter tuning.

### **4.16.13 Performance report evaluation for RF:**

Table

Description automatically generated

Fig 91: Classification report for RF

From above figure 91 says the highest correctly classified class is class 1 is 78% and lowest for the class 5 is 28 percent in precision and in recall out of the total actual value the highest classified correctly positive value for the class 1 is 87 and lowest is 26% for class 5. Precision and recall should be high invariably. As f1 score goes to the score 1 then it’s best and worst score is 0.

### **4.16.14 Hyperparameter Tuning for RF Classifier**

Because of the highly overfitting value in this model before tuning we need to lessen this by tuning.

Having been tuned the accuracy was got 61% for both train and test accuracy which is better comparatively to the before tuning RF model.

The best optimal values were best parameters: {'n\_estimators': 56, 'min\_samples\_split': 5, 'min\_samples\_leaf': 2, 'max\_features': 'auto', 'max\_depth': 4, 'bootstrap': False} and best score was 0.61.

### **4.16.15 K Nearest Neighbor Classifier**

After fitting model, we had the accuracy for both train and test were 97% and 58% respectively but it seems highly overfitted model and that needs to be curtailed.

### **4.16.16 Performance report evaluation for KNN:**

Table

Description automatically generated

Fig 92: Classification report for KNN

From above figure 92 says that the highest correctly classified class is class 1 which was 76% and lowest for the class 5 was 25 percent in precision and in recall out of the total actual value the highest classified correctly positive value for the class 1 is 75 and lowest is 25% for class 5. The rest of the classes’ precision, recall and f1 scores were average.

### **4.16.17 Hyperparameter Tuning for KNN Classifier**

The best K value selection is considered tuning for K nearest neighbor. By applying elbow method, we can select best K value for this algorithm. Each value of K, we can compute the WCSS (within cluster sum of square) value and a range of K values the process is done. To select best K values a function was built by looping K range values from 1 to 40 and it returned an array of outputs based on K values. For the K=30 we had better accuracy for this model, and it has been possible to reduce overfitting problem. A figure had been given below about K selection method.

Chart, scatter chart

Description automatically generated

Fig 93: K values vs error rate

We can see from the above figure 93 that after k value 30 the error rate tends to go upward so assigning K=30 we had the accuracy for the training was 68% and for test was 66% which were better from earlier KNN model.

### **4.16.18 Multinomial Naïve Bayes**

we had accuracy for the model was 61% for both train and test accuracy but precision and recall score were not good for each class.

### **4.16.19 Performance report evaluation for MNB**

**Table

Description automatically generated**

Fig 94**:** Classification report for MNB

To see the classification report from above figure 94 that we can say from the above figure that the precision, recall and f1 score values were not classified utterly except for class 1.

We need to see that after applying optimal value optimization whether any change comes for the model.

### **4.16.20 Hyperparameter Tuning for MNB Classifier**

Having applied tuned parameters for this model we got the identical accuracy and precision, recall and f1 score as we had before tuned. Though we had rough trained and test accuracy for this model, however the classification report says it couldn’t classify any values for class 2 to class 5 utterly only correctly classified a significant percentage for the class 1.

we know that if we select high alpha value then it is cause for overfitting (high variance) problem and if we select small alpha value then it is cause for underfitting (high bias) problem, so we need to trade-off the bias and variance problem by cross-validation using GridSearchCV and was used 10 K fold cross-validation for tuning purpose.

The best optimal parameter was **{'alpha': 1e-05}** which is very small like 0.00001.

### **4.16.21 Support Vector Classifier**

After fitting train and test data for this model the accuracies were got for train and test was 0.63 or 63% which can be said rough score.

### **4.16.22 Performance reports evaluation for SVC**

Table

Description automatically generated

Fig 95: Classification report for SVC

In the above figure 95 says that for all classes’ precision, recall and f1 scores were rough except class 2 where all values were 0.

### **4.16.23 Tuning for SVC**

Having been assigned best optimal parameters to RandomizedSearchCV the accuracies were got for train and test score were 73% and 65% respectively. The best optimal parameters were selected by RandomizedSearchCV were **{‘gamma’:100,’C’:10}** and the best score was 65%. we know that as gamma value is more increasing, the model complexity be higher. C parameter here is a penalty parameter which fixes that how much error could be endurable for model and the way can be balanced the decision boundary and misclassified data. While C is high then it is most likely to be classified significant data correctly.

The **RBF** (radio basis function) kernel used as auto selected kernel which converts data from lower dimension to higher dimension space.

For the large dataset the algorithm is not better option because to fit data it takes huge time.

### 

### **4.16.24 Gradient Boosting Classifier**

We had accuracy for this model were 69% and 68% for the train and test accuracy respectively.

Table

Description automatically generated

Fig 96: Classification with accuracy for GBC

From above figure 96 says that after tuning for GBC the classification report as precision, recall and f1 scores were rough. The best accuracy score was 68% which is highest among all models.

### **4.16.25 Hyperparameter tuning for GBC**

Having been assigned learning rates to this model the accuracies were got based on different learning rates which figure was given below

Graphical user interface, text, application, email

Description automatically generated

Fig 97: Learning rates vs train&test scores for GBC

We can see from figure 97 that after being given different learning rates to model then we have had the best score for the learning rate is 1 that is 0.67 for both train and test and lowest score is 0.62 for the learning rate 0.05.

Before giving different learning rates, we had best score at first and that was 68% for the learning rate 0.1 which is default.

### **4.16.26 AdaBoost Classifier**

For this boosting technique we had accuracy for both train and test were 63% and the classification report as precision, recall and f1 scores were good somewhat which given below

Table

Description automatically generated with low confidence

Fig 98: Classification report with accuracies for AdaBoost

In the above the figure 98 says that precision, recall and f1-score of AdaBoost classifier where for class 1 all scores were highest, and the data correctly classified a small amount for rest of the classes

# 5 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model names | Train accuracy  (%) | Test  Accuracy  (%) | Tuned  Train acc. (%) | Tuned test  Accuracy  (%) | Parameters  Used for tuning | Roc\_auc\_score  (%) |
| Logistic  regression | 0.61 | 0.61 | 0.61 | 0.61 | {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'} | Train:0.68 |
| Test :0.68 |
| Decision  Tree classifier | 0.98 | 0.61 | 0.67 | 0.66 | {'min\_samples\_leaf': 22,  'max\_features': 14, 'max\_depth': 9, 'criterion': 'entropy'} | Train:0.99 |
| Test :0.85 |
| Random  Forest classifier | 0.98 | 0.66 | 0.61 | 0.61 | {'n\_estimators': 56, 'min\_samples\_split': 5, 'min\_samples\_leaf': 2, 'max\_features': 'auto', 'max\_depth': 4, 'bootstrap': False} | Train:0.99 |
| Test :0.85 |
| K nearest neighbor | 0.97 | 0.58 | 0.68 | 0.66 | K=30 | Train:0.91 |
| Test :0.79 |
| Multinomial  Naïve bayes | 0.61 | 0.61 | 0.61 | 0.61 | {'alpha': 1e-05},  Cv=10 | Train:0.61 |
| Test :0.61 |
| Support vector machine | 0.63 | 0.63 | 0.61 | 0.61 | {'kernel': 'rbf', 'gamma': 0.01, 'C': 0.1} | Train :0.79  Test :0.79 |
|  |
| Gradient  Boost classifier | 0.69 | 0.68 | 0.67 | 0.67 | n\_estimators=20, learning\_rate=[0.05, 0.075, 0.1, 0.25,0.3, 0.5, 0.75,0.8,0.9,1], max\_features=2, max\_depth=2, random\_state=5 | Train:0.87 |
| Test :0.86 |
| AdaBoost classifier | 0.63 | 0.63 | **……** | **…….** | **…………………..** | Train:0.67 |
| Test :0.67 |

Table 1: Overall accuracy information

In the above table 1 all of the models’ information like train and test accuracy, after tuning train and test accuracy, what parameter used for hyperparameter and eventually auc\_roc-score were mentioned.

## **Comparison amidst all models’ accuracies**

Now we can say that among the ensemble models the Gradientboost classifier has given highest accuracy for both train and test accuracy compared to Random Forest, Gradient Boost and Decision Tree albeit all abide the principles of decision tree classifier.

The second highest accuracy given by Decision tree classifier and rest of them given rough score both train and test accuracy and a touch above 60% accuracy for both after having been applied hyperparameter tuning owing to at the beginning, the random forest and decision tree classifier models were overfitted means high bias and eventually through being applied parameters optimization the models become generalised model.

So that can be said hyperparameter tuning was paramount important to be solved overfitted problem.

## **Roc Auc Scores**

To be being quantified the model performance output quality by AUC\_ROC\_score which means area under the curve of receiver operating characteristic score which computed by TPR (true positive rate) in y axis and FPR (false positive rate) in x axis. As area under the curve AUC is closer to 1 that means the model performance is high and false positive rate is low and the models can differentiate all positive and negative points successfully.

Our all models auc\_roc\_score is high except logistic regression and multinomial naïve bayes classifier algorithms.

## **Classification report details before and after tuning**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model names | classes | Classification  Report before tuning | | | | Classification  Report after tuning | | | | | | Best score for tuning |
| precision | recall | | F1 | precision | recall | | | | F1 |
| Logistic Regression | 1  2  3  4  5 | 0.62  0.00  0.21  0.00  0.00 | | 0.98  0.00  0.03  0.00  0.00 | 0.76  0.00  0.05  0.00  0.00 | 0.62  0.00  0.00  0.00  0.00 | 1.00  0.00  0.00  0.00  0.00 | | | 0.76  0.00  0.00  0.00  0.00 | | 0.61 |
| Decision Tree  Classifier | 1  2  3  4  5 | 0.78  0.29  0.33  0.46  0.27 | | 0.78  0.30  0.32  0.45  0.25 | 0.78  0.29  0.33  0.45  0.26 | 0.73  0.50  0.42  0.48  0.30 | 0.89  0.17  0.26  0.65  0.09 | | | 0.80  0.25  0.32  0.55  0.13 | | 0.65 |
| Random Forest  Classifier | 1  2  3  4  5 | 0.78  0.40  0.41  0.55  0.28 | | 0.87  0.27  0.34  0.56  0.26 | 0.82  0.32  0.37  0.56  0.27 | 0.62  0.00  0.00  0.00  0.00 | 1.00  0.00  0.00  0.00  0.00 | | | 0.76  0.00  0.00  0.00  0.00 | | 0.61 |
| K nearest  neighbor | 1  2  3  4  5 | 0.76  0.27  0.33  0.37  0.25 | | 0.75  0.28  0.33  0.36  0.25 | 0.76  0.27  0.33  0.36  0.25 | 0.73  0.52  0.44  0.52  0.33 | 0.89  0.19  0.29  0.57  0.26 | | | 0.80  0.28  0.35  0.55  0.29 | | 0.67 |
| Support vector  machine | 1  2  3  4  5 | 0.64  0.00  0.55  0.53  0.33 | | 0.98  0.00  0.04  0.39  0.00 | 0.77  0.00  0.07  0.45  0.01 | ………. | ……… | | | ……. | | 0.61 |
| Multinomial  Naïve bayes | 1  2  3  4  5 | 0.62  0.00  0.00  0.00  0.00 | | 1.00  0.00  0.00  0.00  0.00 | 0.76  0.00  0.00  0.00  0.00 | 0.62  0.00  0.00  0.00  0.00 | | 1.00  0.00  0.00  0.00  0.00 | | 0.62  0.00  0.00  0.00  0.00 | | 0.61 |
| Gradient  Boosting  classifier | 1  2  3  4  5 | 0.75  0.52  0.46  0.55  0.31 | | 0.91  0.23  0.30  0.61  0.16 | 0.82  0.32  0.36  0.58  0.21 | …………. | | ………. | ……. | | | 0.68 |
| AdaBoost  classifier | 1  2  3  4  5 | 0.68  0.33  0.37  0.43  0.22 | | 0.93  0.00  0.06  0.71  0.06 | 0.79  0.01  0.11  0.53  0.10 | ………….. | | ………. | ……… | | | 0.64 |

**Table 2:** Classification report for before and after tuning

In the above table 2 has contained all information regarding model evaluation scores before tuning and after tuning for every class as what are the precision, recall and f1 scores were mentioned above which had been got from used different machine learning models.

# 6 Conclusion and Future works

## **6.1 Conclusions**

In this project the researcher endeavoured to predict the Air quality index based on independent features and target feature and have been able to do that successfully using bagging and boosting technique, logistic regression and probability-based classification algorithm is multinomial naïve bayes where ensemble methods have given better performance though there was overfitted problem and solved them using by tuning technique. For this project boosting techniques as AdaBoost, Gradient boost have given optimal accuracy than bagging technique is random forest.

It was advantageous in this project that there was no multicollinearity problem that means independent features were not highly correlated to each other which was defined by the Pearson correlation coefficient. If multicollinearity problem had been in independent features, then it would have undermined the statistical significance therefore the model accuracy would have fluctuated either.

Feature selection was important also to avoid curse of dimensionality problem.

It obviously must be admitted that in this project data skewness and outliers were big troubles, so data transformation has played very significant role to handle data skewness and shifted data from not normal distribution to gaussian distribution which also lessened the data variance.

And eventually it must have to be said that hyperparameter tuning was also vital important to diminish model’s overfitting and accuracy boosting.

## **6.2 Future works**

The project has been completed to keep aim like any environmental issue as carbon emission reduction, air pollution control, weather prediction and related all things can be used as a model. Take this model any software development organization implement it for their AI usages.

Though the project was developed for a specific country’s data but can be model for other countries owing to in every has the same environmental issue and due to deficiency, any methodical system a hefty amount money must be coughed up in every year so it is thought the project will aid to tackle the reluctant expense.

The researcher also admits the limitations for this project. If had been obtained more valuable data, the project prediction accuracies would have been incredibly high. A significant portion of time had to be given to mend the messy data, data transformation from non-normal distribution to normal distribution data, null values handling, handling data imbalanced.

# Reference

[1] C. R. K, N. R. K, B. P. K and P. S. Rajendran, "The Prediction of Quality of the Air Using Supervised Learning," 2021 6th International Conference on Communication and Electronics Systems (ICCES), 2021.

[2] K. M. O. V. K. Kekulanadara, B. T. G. S. Kumara and B. Kuhaneswaran, "Machine Learning Approach for Predicting Air Quality Index," 2021 International Conference on Decision Aid Sciences and Application (DASA), 2021.

[3] R. W. Gore and D. S. Deshpande, "Voting Method for AQI Prediction and Monitoring Air Pollution using Real-Time Data," 2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC), 2020.

[4] Y. Zhou, H. Deng, Y. Zeng, H. Chen and Z. Lian, "Holiday Travel Pattern Forecast Based On Machine Learning Algorithm," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2019.

[5] B. Baran, "Prediction of Air Quality Index by Extreme Learning Machines," 2019 International Artificial Intelligence and Data Processing Symposium (IDAP), 2019.

[6] P. Ajitha, N. Reddy, S. Srinivasan, A. Sivasangari, R. M. Gomathi and E. Brumancia, "Prediction of Air Quality Based on Supervised Learning," 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), 2021.

[7] Y. Huang, Q. Zhao, Q. Zhou and W. Jiang, "Air Quality Forecast Monitoring and Its Impact on Brain Health Based on Big Data and the Internet of Things," in IEEE Access, vol. 6, pp. 78678-78688, 2018.

[8] X. Xi et al., "A comprehensive evaluation of air pollution prediction improvement by a machine learning method," 2015 IEEE International Conference on Service Operations And Logistics, And Informatics (SOLI), 2015.

[9] S. Sur, R. Ghosal and R. Mondal, "Air Pollution Hotspot Identification and Pollution Level Prediction in the City of Delhi," 2020 IEEE 1st International Conference for Convergence in Engineering (ICCE), 2020.

[10] K. M. O. V. K. Kekulanadara, B. T. G. S. Kumara and B. Kuhaneswaran, "Machine Learning Approach for Predicting Air Quality Index," 2021 International Conference on Decision Aid Sciences and Application (DASA), 2021.

[11]<https://www.researchgate.net/publication/359272936_Applying_Decision_Tree_Algorithm_for_Air_Quality_Prediction_in_Bangladesh>

[12] P. R. Meris et al., "IOT Based – Automated Indoor Air Quality and LPG Leak Detection Control System using Support Vector Machine," 2020 11th IEEE Control and System Graduate Research Colloquium (ICSGRC), 2020.

[13] Z. Liu, G. Wang, L. Zhao and G. Yang, "Multi-Points Indoor Air Quality Monitoring Based on Internet of Things," in IEEE Access, vol. 9, pp. 70479-70492, 2021.

[14] A. Janarthanan, A. Paramarthalingam, A. Arivunambi and P. M. D. R. Vincent, "Real-time indoor air quality monitoring using the Internet of Things," 2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT), 2022.

[15] S. D. Permai, N. T. M. Sagala, A. Y. Zakiyyah, H. Tanty and J. Harefa, "Multiclass Classification for Air Quality In Jakarta Using Support Vector Machine and Multi-Layer Perceptron Classifier," 2022 3rd International Conference on Artificial Intelligence and Data Sciences (AiDAS), 2022, pp. 198-2022.

[16 W. F. Ip, C. Vong, J. Y. Yang and P. K. Wong, "Forecasting daily ambient air pollution based on least squares support vector machines," The 2010 IEEE International Conference on Information and Automation, 2010.]

# Appendix and Code

## **Appendix A Data Loading**

Library **importing**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import accuracy\_score,confusion\_matrix

*#from google.colab import files*

*#upload= files.upload()*

df=pd.read\_csv('data.csv',encoding='unicode\_escape')

pd.pandas.set\_option('display.max\_columns',None)

pd.pandas.set\_option('display.max\_rows',None)

df1=pd.read\_csv('dataa.csv',encoding='unicode\_escape')

pd.pandas.set\_option('display.max\_columns',None)

pd.pandas.set\_option('display.max\_rows',None)

df.shape

(435742, 13)

df1.shape

(34186, 21)

df.head()

|  | **stn\_code** | **sampling\_date** | **state** | **location** | **agency** | **type** | **so2** | **no2** | **rspm** | **spm** | **location\_monitoring\_station** | **pm2\_5** | **date** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 150.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 4.8 | 17.4 | NaN | NaN | NaN | NaN | 1990-02-01 |
| **1** | 151.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 3.1 | 7.0 | NaN | NaN | NaN | NaN | 1990-02-01 |
| **2** | 152.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.2 | 28.5 | NaN | NaN | NaN | NaN | 1990-02-01 |
| **3** | 150.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.3 | 14.7 | NaN | NaN | NaN | NaN | 1990-03-01 |
| **4** | 151.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 4.7 | 7.5 | NaN | NaN | NaN | NaN | 1990-03-01 |

df1.head()

|  | **State** | **County** | **Year** | **Days with AQI** | **Good Days** | **Moderate Days** | **Unhealthy for Sensitive Groups Days** | **Unhealthy Days** | **Very Unhealthy Days** | **Hazardous Days** | **Max AQI** | **90th Percentile AQI** | **Median AQI** | **Days CO** | **Days NO2** | **Days Ozone** | **Days SO2** | **Days PM2.5** | **Days PM10** | **Latitude** | **Longitude** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Alabama | DeKalb | 2021 | 58 | 58 | 0 | 0 | 0 | 0 | 0 | 49 | 42 | 32 | 0 | 0 | 58 | 0 | 0 | 0 | 34.289001 | -85.970065 |
| **1** | Alabama | Jefferson | 2021 | 60 | 33 | 27 | 0 | 0 | 0 | 0 | 93 | 64 | 50 | 1 | 1 | 2 | 0 | 55 | 1 | 33.565278 | -86.796389 |
| **2** | Alaska | Denali | 2021 | 59 | 59 | 0 | 0 | 0 | 0 | 0 | 43 | 41 | 39 | 0 | 0 | 59 | 0 | 0 | 0 | 63.723200 | -148.967600 |
| **3** | Arizona | Apache | 2021 | 87 | 87 | 0 | 0 | 0 | 0 | 0 | 25 | 19 | 11 | 0 | 0 | 0 | 0 | 1 | 86 | 34.058475 | -109.441259 |
| **4** | Arizona | Cochise | 2021 | 90 | 77 | 12 | 1 | 0 | 0 | 0 | 104 | 54 | 41 | 0 | 0 | 48 | 0 | 0 | 42 | 32.009410 | -109.389060 |

df1.columns

Index(['State', 'County', 'Year', 'Days with AQI', 'Good Days',

'Moderate Days', 'Unhealthy for Sensitive Groups Days',

'Unhealthy Days', 'Very Unhealthy Days', 'Hazardous Days', 'Max AQI',

'90th Percentile AQI', 'Median AQI', 'Days CO', 'Days NO2',

'Days Ozone', 'Days SO2', 'Days PM2.5', 'Days PM10', 'Latitude',

'Longitude'],

dtype='object')

COLUMNS ADDING FROM DF1

import pandas as pd

f\_column = df1[['Days SO2','Days NO2','Days CO','Days Ozone','Days PM2.5']]

df = pd.concat([df,f\_column], axis = 1)

df.head()

|  | **stn\_code** | **sampling\_date** | **state** | **location** | **agency** | **type** | **so2** | **no2** | **rspm** | **spm** | **location\_monitoring\_station** | **pm2\_5** | **date** | **Days SO2** | **Days NO2** | **Days CO** | **Days Ozone** | **Days PM2.5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 150.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 4.8 | 17.4 | NaN | NaN | NaN | NaN | 1990-02-01 | 0.0 | 0.0 | 0.0 | 58.0 | 0.0 |
| **1** | 151.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 3.1 | 7.0 | NaN | NaN | NaN | NaN | 1990-02-01 | 0.0 | 1.0 | 1.0 | 2.0 | 55.0 |
| **2** | 152.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.2 | 28.5 | NaN | NaN | NaN | NaN | 1990-02-01 | 0.0 | 0.0 | 0.0 | 59.0 | 0.0 |
| **3** | 150.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.3 | 14.7 | NaN | NaN | NaN | NaN | 1990-03-01 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **4** | 151.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 4.7 | 7.5 | NaN | NaN | NaN | NaN | 1990-03-01 | 0.0 | 0.0 | 0.0 | 48.0 | 0.0 |

df.tail()

|  | **stn\_code** | **sampling\_date** | **state** | **location** | **agency** | **type** | **so2** | **no2** | **rspm** | **spm** | **location\_monitoring\_station** | **pm2\_5** | **date** | **Days SO2** | **Days NO2** | **Days CO** | **Days Ozone** | **Days PM2.5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **435737** | SAMP | 24-12-15 | West Bengal | ULUBERIA | West Bengal State Pollution Control Board | RIRUO | 22.0 | 50.0 | 143.0 | NaN | Inside Rampal Industries,ULUBERIA | NaN | 2015-12-24 | NaN | NaN | NaN | NaN | NaN |
| **435738** | SAMP | 29-12-15 | West Bengal | ULUBERIA | West Bengal State Pollution Control Board | RIRUO | 20.0 | 46.0 | 171.0 | NaN | Inside Rampal Industries,ULUBERIA | NaN | 2015-12-29 | NaN | NaN | NaN | NaN | NaN |
| **435739** | NaN | NaN | andaman-and-nicobar-islands | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **435740** | NaN | NaN | Lakshadweep | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **435741** | NaN | NaN | Tripura | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

df.columns

Index(['stn\_code', 'sampling\_date', 'state', 'location', 'agency', 'type',

'so2', 'no2', 'rspm', 'spm', 'location\_monitoring\_station', 'pm2\_5',

'date', 'Days SO2', 'Days NO2', 'Days CO', 'Days Ozone', 'Days PM2.5'],

dtype='object')

df.shape

(435742, 18)

df.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 435742 entries, 0 to 435741

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 stn\_code 291665 non-null object

1 sampling\_date 435739 non-null object

2 state 435742 non-null object

3 location 435739 non-null object

4 agency 286261 non-null object

5 type 430349 non-null object

6 so2 401096 non-null float64

7 no2 419509 non-null float64

8 rspm 395520 non-null float64

9 spm 198355 non-null float64

10 location\_monitoring\_station 408251 non-null object

11 pm2\_5 9314 non-null float64

12 date 435735 non-null object

13 Days SO2 34186 non-null float64

14 Days NO2 34186 non-null float64

15 Days CO 34186 non-null float64

16 Days Ozone 34186 non-null float64

17 Days PM2.5 34186 non-null float64

dtypes: float64(10), object(8)

memory usage: 63.2+ MB

df.drop('pm2\_5',axis=1,inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 435742 entries, 0 to 435741

Data columns (total 17 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 stn\_code 291665 non-null object

1 sampling\_date 435739 non-null object

2 state 435742 non-null object

3 location 435739 non-null object

4 agency 286261 non-null object

5 type 430349 non-null object

6 so2 401096 non-null float64

7 no2 419509 non-null float64

8 rspm 395520 non-null float64

9 spm 198355 non-null float64

10 location\_monitoring\_station 408251 non-null object

11 date 435735 non-null object

12 Days SO2 34186 non-null float64

13 Days NO2 34186 non-null float64

14 Days CO 34186 non-null float64

15 Days Ozone 34186 non-null float64

16 Days PM2.5 34186 non-null float64

dtypes: float64(9), object(8)

memory usage: 59.8+ MB

DROPPING ALL NAN VALUES FROM DATAFRAME KEEPING SUBSET PM2.5 COLUMN

df=df.dropna(subset=['Days PM2.5'])

*#checking info after dropping NAN values from dataframe*

df.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 34186 entries, 0 to 34185

Data columns (total 17 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 stn\_code 19425 non-null object

1 sampling\_date 34186 non-null object

2 state 34186 non-null object

3 location 34186 non-null object

4 agency 18999 non-null object

5 type 33331 non-null object

6 so2 33352 non-null float64

7 no2 33539 non-null float64

8 rspm 32769 non-null float64

9 spm 17684 non-null float64

10 location\_monitoring\_station 32905 non-null object

11 date 34186 non-null object

12 Days SO2 34186 non-null float64

13 Days NO2 34186 non-null float64

14 Days CO 34186 non-null float64

15 Days Ozone 34186 non-null float64

16 Days PM2.5 34186 non-null float64

dtypes: float64(9), object(8)

memory usage: 4.7+ MB

df.shape

(34186, 17)

CHECKING NUMBER OF ZEROS(0'S) IN COLUMNS TO DROP SUCH COLUMN WHO HAS MORE ZERO VALUES

(df['Days SO2']==0).sum()

23416

(df['so2']==0).sum()

0

(df['Days NO2']==0).sum()

25979

(df['no2']==0).sum()

0

*#as we see column so2 and no2 haven't any 0 values so we can drop other two*

df.drop(['Days SO2','Days NO2'],axis=1,inplace=True)

df.head()

|  | **stn\_code** | **sampling\_date** | **state** | **location** | **agency** | **type** | **so2** | **no2** | **rspm** | **spm** | **location\_monitoring\_station** | **date** | **Days CO** | **Days Ozone** | **Days PM2.5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 150.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 4.8 | 17.4 | NaN | NaN | NaN | 1990-02-01 | 0.0 | 58.0 | 0.0 |
| **1** | 151.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 3.1 | 7.0 | NaN | NaN | NaN | 1990-02-01 | 1.0 | 2.0 | 55.0 |
| **2** | 152.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.2 | 28.5 | NaN | NaN | NaN | 1990-02-01 | 0.0 | 59.0 | 0.0 |
| **3** | 150.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.3 | 14.7 | NaN | NaN | NaN | 1990-03-01 | 0.0 | 0.0 | 1.0 |
| **4** | 151.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 4.7 | 7.5 | NaN | NaN | NaN | 1990-03-01 | 0.0 | 48.0 | 0.0 |

*#checking each column total null values*

df.isnull().sum()

stn\_code 14761

sampling\_date 0

state 0

location 0

agency 15187

type 855

so2 834

no2 647

rspm 1417

spm 16502

location\_monitoring\_station 1281

date 0

Days CO 0

Days Ozone 0

Days PM2.5 0

dtype: int64

*#RENAMING LAST 3 COLUMNS*

df = df.rename(columns={'Days CO': 'CO', 'Days Ozone': 'Ozone','Days PM2.5':'PM2.5'})

df.head()

|  | **stn\_code** | **sampling\_date** | **state** | **location** | **agency** | **type** | **so2** | **no2** | **rspm** | **spm** | **location\_monitoring\_station** | **date** | **CO** | **Ozone** | **PM2.5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 150.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 4.8 | 17.4 | NaN | NaN | NaN | 1990-02-01 | 0.0 | 58.0 | 0.0 |
| **1** | 151.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 3.1 | 7.0 | NaN | NaN | NaN | 1990-02-01 | 1.0 | 2.0 | 55.0 |
| **2** | 152.0 | February - M021990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.2 | 28.5 | NaN | NaN | NaN | 1990-02-01 | 0.0 | 59.0 | 0.0 |
| **3** | 150.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Residential, Rural and other Areas | 6.3 | 14.7 | NaN | NaN | NaN | 1990-03-01 | 0.0 | 0.0 | 1.0 |
| **4** | 151.0 | March - M031990 | Andhra Pradesh | Hyderabad | NaN | Industrial Area | 4.7 | 7.5 | NaN | NaN | NaN | 1990-03-01 | 0.0 | 48.0 | 0.0 |

df.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 33352.0 | 7.147949 | 5.946795 | 0.4 | 4.1 | 5.1 | 8.2 | 228.0 |
| **no2** | 33539.0 | 20.200883 | 11.429543 | 0.5 | 12.0 | 17.0 | 26.6 | 334.9 |
| **rspm** | 32769.0 | 81.717272 | 45.214228 | 2.0 | 53.0 | 74.0 | 100.0 | 790.0 |
| **spm** | 17684.0 | 184.624497 | 99.794442 | 5.0 | 115.0 | 170.0 | 236.0 | 1300.0 |
| **CO** | 34186.0 | 10.762242 | 42.056897 | 0.0 | 0.0 | 0.0 | 0.0 | 366.0 |
| **Ozone** | 34186.0 | 151.626982 | 119.363197 | 0.0 | 0.0 | 167.0 | 235.0 | 366.0 |
| **PM2.5** | 34186.0 | 63.112297 | 93.186920 | 0.0 | 0.0 | 0.0 | 105.0 | 366.0 |

df.nunique()

stn\_code 70

sampling\_date 3859

state 3

location 38

agency 6

type 6

so2 430

no2 747

rspm 579

spm 843

location\_monitoring\_station 105

date 3858

CO 358

Ozone 367

PM2.5 367

dtype: int64

df.duplicated().sum()

0

*#df.drop\_duplicates(inplace=True)*

df.columns

*# These are all the columns present in the dataset.*

Index(['stn\_code', 'sampling\_date', 'state', 'location', 'agency', 'type',

'so2', 'no2', 'rspm', 'spm', 'location\_monitoring\_station', 'date',

'CO', 'Ozone', 'PM2.5'],

dtype='object')

## **Appendix B Data visualization**

Data **Visualization**

sns.pairplot(data=df)

<seaborn.axisgrid.PairGrid at 0x16f9e289f10>

df['state'].value\_counts()

*# Viewing values present in the state column*

Andhra Pradesh 26368

Assam 7728

Arunachal Pradesh 90

Name: state, dtype: int64

plt.figure(figsize=(6, 3))

plt.xticks(rotation=90)

df.state.hist()

plt.xlabel('state')

plt.ylabel('Frequencies')

plt.plot()

*# states present in the dataset.*

[]

df['type'].value\_counts()

*#types of areas*

Residential, Rural and other Areas 11957

Residential and others 11204

Industrial Area 4530

Industrial Areas 2275

Sensitive Area 2083

Sensitive Areas 1282

Name: type, dtype: int64

plt.figure(figsize=(6, 3))

plt.xticks(rotation=90)

df.type.hist()

plt.xlabel('Type')

plt.ylabel('Frequencies')

plt.plot()

*# The visualization shows us the count of Types present in the dataset.*

[]

df['agency'].value\_counts()

*# Viewing the counts of values present in the agency column*

Andhra Pradesh State Pollution Control Board 15691

Assam State Pollution Control Board 1794

Andhra Pradesh Pollution Control Board 610

National Environmental Engineering Research Institute 569

ASPB 245

Arunachal Pradesh State Pollution Control Board 90

Name: agency, dtype: int64

plt.figure(figsize=(6, 3))

plt.xticks(rotation=90)

df.agency.hist()

plt.xlabel('Agency')

plt.ylabel('Frequencies')

plt.plot()

*# the count of Agency present in the dataset.*

[]

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

plt.xticks(rotation=90)

sns.barplot(x='state',y='so2',data=df);

*# This visualization shows the name of the state having higher so2 levels in the air which is andhra pradesh*

plt.rcParams['figure.figsize']=(8,6)

df[['so2','state']].groupby(["state"]).mean().sort\_values(by='so2').plot.bar(color='purple')

plt.show()

*# We can also use the groupby function to sort values in an ascending order based on the x-axis, y-axis and its keys*

*# Below we get a clear picture of the states in an increasing order based on their so2 levels.*

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

plt.xticks(rotation=90)

sns.barplot(x='state',y='no2',data=df);

*# West bengal has a higher no2 level compared to other states*

df[['no2','state']].groupby(["state"]).mean().sort\_values(by='no2').plot.bar(color='purple')

plt.show()

*# We can also use the groupby function to sort values in an ascending order based on the x-axis, y-axis and its keys*

*# Below we get a clear picture of the states in an increasing order based on their no2 levels.*

plt.figure(figsize=(5, 3))

plt.xticks(rotation=90)

sns.barplot(x='state',y='rspm',data=df);

*# Assam has higher rspm level compared to other states*

plt.figure(figsize=(5, 3))

plt.xticks(rotation=90)

sns.barplot(x='state',y='spm',data=df);

*# Delhi has higher spm level compared to other states*

plt.figure(figsize=(6,5))

sns.barplot(x='type',y='PM2.5',data=df);

plt.show()

*# Assam has higher rspm level compared to other states*

plt.figure(figsize=(5, 3))

plt.xticks(rotation=90)

sns.barplot(x='state',y='PM2.5',data=df);

*# Delhi has higher pm2\_5 level compared to other states*

plt.figure(figsize=(5, 3))

plt.xticks(rotation=90)

sns.barplot(x='type',y='PM2.5',data=df);

plt.scatter(x='so2',y='PM2.5',data=df)

plt.xlabel('so2')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

plt.scatter(x='no2',y='PM2.5',data=df)

plt.xlabel('no2')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

plt.scatter(x='rspm',y='PM2.5',data=df)

plt.xlabel('rspm')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

plt.scatter(x='spm',y='PM2.5',data=df)

plt.xlabel('spm')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

plt.scatter(x='CO',y='PM2.5',data=df)

plt.xlabel('CO')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

plt.scatter(x='Ozone',y='PM2.5',data=df)

plt.xlabel('Ozone')

plt.ylabel('PM2.5')

Text(0, 0.5, 'PM2.5')

Checking all null values and treating those null values.

null\_values = df.isnull().sum().sort\_values(ascending=False)

*# Checking all null values*

null\_values

*# higher null values present in spm followed by spm*

spm 16502

agency 15187

stn\_code 14761

rspm 1417

location\_monitoring\_station 1281

type 855

so2 834

no2 647

sampling\_date 0

state 0

location 0

date 0

CO 0

Ozone 0

PM2.5 0

dtype: int64

null\_values\_percentages = (df.isnull().sum()/df.isnull().count()\*100).sort\_values(ascending=False)

*#count(returns Non-NAN value percentages)*

null\_values\_percentages

spm 48.271222

agency 44.424618

stn\_code 43.178494

rspm 4.144972

location\_monitoring\_station 3.747148

type 2.501024

so2 2.439595

no2 1.892588

sampling\_date 0.000000

state 0.000000

location 0.000000

date 0.000000

CO 0.000000

Ozone 0.000000

PM2.5 0.000000

dtype: float64

missing\_data\_with\_percentages = pd.concat([null\_values, null\_values\_percentages], axis=1, keys=['TOTAL', 'Percentage'])

*# Concatenating total null values and their percentage of missing values for further imputation or column deletion*

missing\_data\_with\_percentages

|  | **TOTAL** | **Percentage** |
| --- | --- | --- |
| **spm** | 16502 | 48.271222 |
| **agency** | 15187 | 44.424618 |
| **stn\_code** | 14761 | 43.178494 |
| **rspm** | 1417 | 4.144972 |
| **location\_monitoring\_station** | 1281 | 3.747148 |
| **type** | 855 | 2.501024 |
| **so2** | 834 | 2.439595 |
| **no2** | 647 | 1.892588 |
| **sampling\_date** | 0 | 0.000000 |
| **state** | 0 | 0.000000 |
| **location** | 0 | 0.000000 |
| **date** | 0 | 0.000000 |
| **CO** | 0 | 0.000000 |
| **Ozone** | 0 | 0.000000 |
| **PM2.5** | 0 | 0.000000 |

df.drop(['agency'],axis=1,inplace=True)

df.drop(['stn\_code'],axis=1,inplace=True)

df.drop(['date'],axis=1,inplace=True)

df.drop(['sampling\_date'],axis=1,inplace=True)

df.drop(['location\_monitoring\_station'],axis=1,inplace=True)

*# Dropping redundant columns*

df.isnull().sum()

*# Now checking the null values*

state 0

location 0

type 855

so2 834

no2 647

rspm 1417

spm 16502

CO 0

Ozone 0

PM2.5 0

dtype: int64

df.head()

|  | **state** | **location** | **type** | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** | **PM2.5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 4.8 | 17.4 | NaN | NaN | 0.0 | 58.0 | 0.0 |
| **1** | Andhra Pradesh | Hyderabad | Industrial Area | 3.1 | 7.0 | NaN | NaN | 1.0 | 2.0 | 55.0 |
| **2** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.2 | 28.5 | NaN | NaN | 0.0 | 59.0 | 0.0 |
| **3** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.3 | 14.7 | NaN | NaN | 0.0 | 0.0 | 1.0 |
| **4** | Andhra Pradesh | Hyderabad | Industrial Area | 4.7 | 7.5 | NaN | NaN | 0.0 | 48.0 | 0.0 |

## **Appendix C Feature Engineering**

**Feature** **Engineering**

FIRST NEEDS TO BE CHECKED THERE ARE ANY OULIERS OR NOT IN COLUMN THEN SHOULD BE APPLIED TECHNIQUE

df.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 33352.0 | 7.147949 | 5.946795 | 0.4 | 4.1 | 5.1 | 8.2 | 228.0 |
| **no2** | 33539.0 | 20.200883 | 11.429543 | 0.5 | 12.0 | 17.0 | 26.6 | 334.9 |
| **rspm** | 32769.0 | 81.717272 | 45.214228 | 2.0 | 53.0 | 74.0 | 100.0 | 790.0 |
| **spm** | 17684.0 | 184.624497 | 99.794442 | 5.0 | 115.0 | 170.0 | 236.0 | 1300.0 |
| **CO** | 34186.0 | 10.762242 | 42.056897 | 0.0 | 0.0 | 0.0 | 0.0 | 366.0 |
| **Ozone** | 34186.0 | 151.626982 | 119.363197 | 0.0 | 0.0 | 167.0 | 235.0 | 366.0 |
| **PM2.5** | 34186.0 | 63.112297 | 93.186920 | 0.0 | 0.0 | 0.0 | 105.0 | 366.0 |

sns.distplot(df['so2'])

<AxesSubplot:xlabel='so2', ylabel='Density'>

sns.boxplot(df['so2'])

<AxesSubplot:xlabel='so2'>

sns.distplot(df['no2'])

<AxesSubplot:xlabel='no2', ylabel='Density'>

sns.boxplot(df['no2'])

<AxesSubplot:xlabel='no2'>

sns.distplot(df['rspm'])

<AxesSubplot:xlabel='rspm', ylabel='Density'>

sns.boxplot(df['rspm'])

<AxesSubplot:xlabel='rspm'>

sns.distplot(df['spm'])

<AxesSubplot:xlabel='spm', ylabel='Density'>

sns.boxplot(df['spm'])

<AxesSubplot:xlabel='spm'>

sns.distplot(df['CO'])

<AxesSubplot:xlabel='CO', ylabel='Density'>

sns.boxplot(df['CO'])

<AxesSubplot:xlabel='CO'>

sns.distplot(df['Ozone'])

<AxesSubplot:xlabel='Ozone', ylabel='Density'>

sns.boxplot(df['Ozone'])

<AxesSubplot:xlabel='Ozone'>

SINCE ALL COLUMNS except CO and Ozone HAVE NUMEROUS OUTLIERS SO NNEDS TO BE APPLIED MEDIAN TECHNIQUE WHICH BASICALLY ROBUST FOR OUTLIERS

**def** fillednull\_MF(dataf,vars):

MF=dataf[vars].median()

dataf[vars].fillna(MF,inplace=True)

**for** features **in** ['so2','no2','rspm','spm']:

fillednull\_MF(df,features)

df.isnull().sum()

*# Now we have successfully filled null values by median which were present in the dataset*

state 0

location 0

type 855

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

PM2.5 0

dtype: int64

**def** AQI\_Range(x):

**if** x<=50:

**return** "Good"

**elif** x>50 **and** x<=100:

**return** "Moderate"

**elif** x>100 **and** x<=200:

**return** "Poor"

**elif** x>200 **and** x<=300:

**return** "Unhealthy"

**elif** x>300 **and** x<=400:

**return** "Very unhealthy"

**elif** x>400:

**return** "Hazardous"

df['AQI\_Range'] = df['PM2.5'] .apply(AQI\_Range)

df.head()

*# Using threshold values to classify a particular values as good, moderate, poor, unhealthy, very unhealthy and Hazardous*

|  | **state** | **location** | **type** | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** | **PM2.5** | **AQI\_Range** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 4.8 | 17.4 | 74.0 | 170.0 | 0.0 | 58.0 | 0.0 | Good |
| **1** | Andhra Pradesh | Hyderabad | Industrial Area | 3.1 | 7.0 | 74.0 | 170.0 | 1.0 | 2.0 | 55.0 | Moderate |
| **2** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.2 | 28.5 | 74.0 | 170.0 | 0.0 | 59.0 | 0.0 | Good |
| **3** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.3 | 14.7 | 74.0 | 170.0 | 0.0 | 0.0 | 1.0 | Good |
| **4** | Andhra Pradesh | Hyderabad | Industrial Area | 4.7 | 7.5 | 74.0 | 170.0 | 0.0 | 48.0 | 0.0 | Good |

df['AQI\_Range'].value\_counts()

*# These are the counts of values present in the AQI\_Range column.*

Good 21076

Poor 5234

Moderate 4291

Unhealthy 2163

Very unhealthy 1422

Name: AQI\_Range, dtype: int64

ranking={'Good':1,'Moderate':2,'Poor':3,'Unhealthy':4,'Very unhealthy':5}

ranking

{'Good': 1, 'Moderate': 2, 'Poor': 3, 'Unhealthy': 4, 'Very unhealthy': 5}

df['AQI\_RANKING']=df['AQI\_Range'].map(ranking)

df.head()

|  | **state** | **location** | **type** | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** | **PM2.5** | **AQI\_Range** | **AQI\_RANKING** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 4.8 | 17.4 | 74.0 | 170.0 | 0.0 | 58.0 | 0.0 | Good | 1 |
| **1** | Andhra Pradesh | Hyderabad | Industrial Area | 3.1 | 7.0 | 74.0 | 170.0 | 1.0 | 2.0 | 55.0 | Moderate | 2 |
| **2** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.2 | 28.5 | 74.0 | 170.0 | 0.0 | 59.0 | 0.0 | Good | 1 |
| **3** | Andhra Pradesh | Hyderabad | Residential, Rural and other Areas | 6.3 | 14.7 | 74.0 | 170.0 | 0.0 | 0.0 | 1.0 | Good | 1 |
| **4** | Andhra Pradesh | Hyderabad | Industrial Area | 4.7 | 7.5 | 74.0 | 170.0 | 0.0 | 48.0 | 0.0 | Good | 1 |

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB,MultinomialNB

from sklearn.svm import SVC

DATA **SPLITTING**

X2=df[['so2','no2','rspm','spm','CO','Ozone']]

Y2=df['AQI\_RANKING']

X2.head()

*# we only select columns like so2, no2, rspm, spm,CO,Ozone*

|  | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.8 | 17.4 | 74.0 | 170.0 | 0.0 | 58.0 |
| **1** | 3.1 | 7.0 | 74.0 | 170.0 | 1.0 | 2.0 |
| **2** | 6.2 | 28.5 | 74.0 | 170.0 | 0.0 | 59.0 |
| **3** | 6.3 | 14.7 | 74.0 | 170.0 | 0.0 | 0.0 |
| **4** | 4.7 | 7.5 | 74.0 | 170.0 | 0.0 | 48.0 |

X\_train2,X\_test2,Y\_train2,Y\_test2=train\_test\_split(X2,Y2,test\_size=0.2,random\_state=0)

print(X\_train2.shape,X\_test2.shape,Y\_train2.shape,Y\_test2.shape)

*# splitting the data into training and testing data*

(27348, 6) (6838, 6) (27348,) (6838,)

X\_train2.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 27348.0 | 7.075431 | 5.680104 | 0.4 | 4.2 | 5.1 | 8.0 | 137.0 |
| **no2** | 27348.0 | 20.108450 | 11.369125 | 0.5 | 12.0 | 17.0 | 26.2 | 334.9 |
| **rspm** | 27348.0 | 81.363589 | 44.255473 | 2.0 | 54.0 | 74.0 | 98.0 | 769.0 |
| **spm** | 27348.0 | 177.700183 | 72.518709 | 8.0 | 166.0 | 170.0 | 173.0 | 1300.0 |
| **CO** | 27348.0 | 10.889571 | 42.363496 | 0.0 | 0.0 | 0.0 | 0.0 | 366.0 |
| **Ozone** | 27348.0 | 151.492979 | 119.366715 | 0.0 | 0.0 | 166.0 | 235.0 | 366.0 |

**DATA SCALING**

#FOR TRAIN

from sklearn.preprocessing import RobustScaler

scaler=RobustScaler()

xtrain\_scaled=pd.DataFrame(scaler.fit\_transform(X\_train2),columns=X\_train2.columns)

xtrain\_scaled.head()

|  | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | -0.289474 | -0.563380 | -0.272727 | 0.0 | 0.0 | 0.672340 |
| **1** | 1.552632 | -0.563380 | -1.090909 | 0.0 | 0.0 | -0.161702 |
| **2** | -0.289474 | -0.563380 | -0.590909 | 0.0 | 0.0 | -0.706383 |
| **3** | 0.236842 | 2.676056 | 1.295455 | 0.0 | 0.0 | -0.706383 |
| **4** | 0.500000 | -0.070423 | -0.909091 | 0.0 | 0.0 | -0.706383 |

xtrain\_scaled.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 27348.0 | 0.519850 | 1.494764 | -1.236842 | -0.236842 | 0.0 | 0.763158 | 34.710526 |
| **no2** | 27348.0 | 0.218905 | 0.800643 | -1.161972 | -0.352113 | 0.0 | 0.647887 | 22.387324 |
| **rspm** | 27348.0 | 0.167354 | 1.005806 | -1.636364 | -0.454545 | 0.0 | 0.545455 | 15.795455 |
| **spm** | 27348.0 | 1.100026 | 10.359816 | -23.142857 | -0.571429 | 0.0 | 0.428571 | 161.428571 |
| **CO** | 27348.0 | 10.889571 | 42.363496 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 366.000000 |
| **Ozone** | 27348.0 | -0.061732 | 0.507943 | -0.706383 | -0.706383 | 0.0 | 0.293617 | 0.851064 |

xtrain\_scaled.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

xtrain\_scaled.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

sns.distplot(xtrain\_scaled['so2'],bins=20)

<AxesSubplot:xlabel='so2', ylabel='Density'>

sns.distplot(xtrain\_scaled['no2'],bins=20)

<AxesSubplot:xlabel='no2', ylabel='Density'>

sns.distplot(xtrain\_scaled['rspm'],bins=20)

<AxesSubplot:xlabel='rspm', ylabel='Density'>

sns.distplot(xtrain\_scaled['spm'],bins=20)

<AxesSubplot:xlabel='spm', ylabel='Density'>

sns.distplot(xtrain\_scaled['CO'],bins=20)

<AxesSubplot:xlabel='CO', ylabel='Density'>

sns.distplot(xtrain\_scaled['Ozone'],bins=20)

<AxesSubplot:xlabel='Ozone', ylabel='Density'>

df.columns

Index(['state', 'location', 'type', 'so2', 'no2', 'rspm', 'spm', 'CO', 'Ozone',

'PM2.5', 'AQI\_Range', 'AQI\_RANKING'],

dtype='object')

#DATA SCALING FOR TEST

from sklearn.preprocessing import RobustScaler

scaler=RobustScaler()

xtest\_scaled=pd.DataFrame(scaler.fit\_transform(X\_test2),columns=X\_test2.columns)

xtest\_scaled.head()

|  | **so2** | **no2** | **rspm** | **spm** | **CO** | **Ozone** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.000000 | 0.486111 | -0.577778 | 0.000000 | 0.0 | -0.029787 |
| **1** | 0.487179 | 0.486111 | 0.333333 | 0.000000 | 0.0 | -0.710638 |
| **2** | 2.025641 | 0.069444 | 0.266667 | 0.000000 | 0.0 | 0.063830 |
| **3** | 0.000000 | 0.000000 | -0.911111 | -13.666667 | 2.0 | -0.395745 |
| **4** | -0.538462 | -0.347222 | 1.644444 | 0.000000 | 0.0 | 0.042553 |

xtest\_scaled.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 6838.0 | 0.535435 | 1.699815 | -1.205128 | -0.205128 | 0.0 | 0.794872 | 57.153846 |
| **no2** | 6838.0 | 0.226923 | 0.775579 | -1.041667 | -0.347222 | 0.0 | 0.652778 | 13.944444 |
| **rspm** | 6838.0 | 0.167391 | 0.987788 | -1.555556 | -0.444444 | 0.0 | 0.555556 | 15.911111 |
| **spm** | 6838.0 | 1.170786 | 11.772198 | -27.500000 | -0.666667 | 0.0 | 0.333333 | 188.333333 |
| **CO** | 6838.0 | 10.252998 | 40.806715 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 360.000000 |
| **Ozone** | 6838.0 | -0.063137 | 0.507899 | -0.710638 | -0.710638 | 0.0 | 0.289362 | 0.846809 |

xtest\_scaled.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

sns.distplot(xtest\_scaled['so2'],bins=20)

<AxesSubplot:xlabel='so2', ylabel='Density'>

sns.distplot(xtest\_scaled['no2'],bins=20)

<AxesSubplot:xlabel='no2', ylabel='Density'>

sns.distplot(xtest\_scaled['rspm'],bins=20)

<AxesSubplot:xlabel='rspm', ylabel='Density'>

sns.distplot(xtest\_scaled['spm'],bins=20)

<AxesSubplot:xlabel='spm', ylabel='Density'>

sns.distplot(xtest\_scaled['CO'],bins=20)

<AxesSubplot:xlabel='CO', ylabel='Density'>

sns.distplot(xtest\_scaled['Ozone'],bins=20)

<AxesSubplot:xlabel='Ozone', ylabel='Density'>

DATA TRANSFORMATION

import scipy.stats as stat

import pylab

*#### want to check whether feature is guassian or normal distributed*

*#### Q-Q plot*

**def** plot\_data(dataf,feature):

plt.figure(figsize=(10,6))

plt.subplot(1,2,1)

X\_train2[feature].hist()

plt.subplot(1,2,2)

stat.probplot(X\_train2[feature],dist='norm',plot=pylab)

plt.show()

*#before transformation*

plot\_data(X\_train2,'so2')

*#logaritm transformation for so2*

import numpy as np

X\_train2['so2']=np.log(X\_train2['so2'])

plot\_data(X\_train2,'so2')

*##### Square Root Transformation*

*#X\_train2['so2']=X\_train2.so2\*\*(1/2)*

*#plot\_data(X\_train2,'so2')*

*#before transformation*

plot\_data(X\_train2,'no2')

*#logaritm transformation for no2*

import numpy as np

X\_train2['no2']=np.log(X\_train2['no2'])

plot\_data(X\_train2,'no2')

*##### Square Root Transformation*

*#X\_train2['no2']=X\_train2.no2\*\*(1/2)*

*#plot\_data(X\_train2,'no2')*

*#before transformation*

plot\_data(X\_train2,'rspm')

*#logaritm transformation for rspm*

import numpy as np

X\_train2['rspm']=np.log(X\_train2['rspm'])

plot\_data(X\_train2,'rspm')

*##### Square Root Transformation*

*#X\_train2['rspm']=X\_train2.rspm\*\*(1/2)*

*#plot\_data(X\_train2,'rspm')*

*#before transformation*

plot\_data(X\_train2,'spm')

*#logaritm transformation for spm*

import numpy as np

X\_train2['spm']=np.log(X\_train2['spm'])

plot\_data(X\_train2,'spm')

*##### Square Root Transformation*

*#X\_train2['spm']=X\_train2.spm\*\*(1/2)*

*#plot\_data(X\_train2,'spm')*

*#before transformation*

plot\_data(X\_train2,'CO')

*# #logaritm transformation for spm*

*# import numpy as np*

*# X\_train2['CO']=np.log(X\_train2['CO'])*

*# plot\_data(X\_train2,'CO')*

*##### Square Root Transformation*

*#X\_train2['CO']=X\_train2.CO\*\*(1/2)*

*#plot\_data(X\_train2,'CO')*

plot\_data(X\_train2,'Ozone')

X\_train2.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

#DATA TRANSFORMATION FOR TEST DATA

*#before transformation*

plot\_data(X\_test2,'so2')

*#logaritm transformation for so2*

import numpy as np

X\_test2['so2']=np.log(X\_test2['so2'])

plot\_data(X\_test2,'so2')

*#before transformation*

plot\_data(X\_test2,'no2')

*#logaritm transformation for no2*

import numpy as np

X\_test2['no2']=np.log(X\_test2['no2'])

plot\_data(X\_test2,'no2')

*#before transformation*

plot\_data(X\_test2,'rspm')

*#logaritm transformation for rspm*

import numpy as np

X\_test2['rspm']=np.log(X\_test2['rspm'])

plot\_data(X\_test2,'rspm')

*#before transformation*

plot\_data(X\_test2,'spm')

*#logaritm transformation for spm*

import numpy as np

X\_test2['spm']=np.log(X\_test2['spm'])

plot\_data(X\_test2,'spm')

*#before transformation*

plot\_data(X\_test2,'CO')

*#logaritm transformation for Ozone*

import numpy as np

X\_test2['CO']=np.log(X\_test2['CO'])

plot\_data(X\_test2,'CO')

*#before transformation*

plot\_data(X\_test2,'Ozone')

X\_train2.shape

(27348, 6)

X\_test2.shape

(6838, 6)

**Data Balancing by SMOTE**

#for train

from pandas import read\_csv

from collections import Counter

from matplotlib import pyplot

from sklearn.preprocessing import LabelEncoder

counter = Counter(Y\_train2)

**for** k,v **in** counter.items():

per = v / len(Y\_train2) \* 100

print('Class=%d, n=%d (%.3f%%)' % (k, v, per))

*# plot the distribution*

pyplot.bar(counter.keys(), counter.values())

pyplot.show()

Class=1, n=16843 (61.588%)

Class=4, n=1707 (6.242%)

Class=5, n=1144 (4.183%)

Class=3, n=4190 (15.321%)

Class=2, n=3464 (12.666%)

X\_train2.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

*# from pandas import read\_csv*

*# from imblearn.over\_sampling import SMOTE*

*# from collections import Counter*

*# from matplotlib import pyplot*

*# oversample = SMOTE()*

*# X\_train2, Y\_train2 = oversample.fit\_resample(X\_train2,Y\_train2)*

*# # summarize distribution*

*# counter = Counter(Y\_train2)*

*# for k,v in counter.items():*

*# per = v / len(Y\_train2) \* 100*

*# print('Class=%d, n=%d (%.3f%%)' % (k, v, per))*

*# # plot the distribution*

*# pyplot.bar(counter.keys(), counter.values())*

*# pyplot.show()*

X\_train2.shape

(27348, 6)

Y\_train2.shape

(27348,)

Y\_train2.value\_counts()

1 16843

3 4190

2 3464

4 1707

5 1144

Name: AQI\_RANKING, dtype: int64

#FOR TEST DATA BALANCING

X\_test2.shape

(6838, 6)

Y\_test2.shape

(6838,)

Y\_test2.value\_counts()

1 4233

3 1044

2 827

4 456

5 278

Name: AQI\_RANKING, dtype: int64

counter = Counter(Y\_test2)

**for** k,v **in** counter.items():

per = v / len(Y\_test2) \* 100

print('Class=%d, n=%d (%.3f%%)' % (k, v, per))

*# plot the distribution*

pyplot.bar(counter.keys(), counter.values())

pyplot.show()

Class=3, n=1044 (15.268%)

Class=1, n=4233 (61.904%)

Class=2, n=827 (12.094%)

Class=4, n=456 (6.669%)

Class=5, n=278 (4.066%)

*# oversample = SMOTE()*

*# X\_test2, Y\_test2 = oversample.fit\_resample(X\_test2,Y\_test2)*

*# # summarize distribution*

*# counter = Counter(Y\_test2)*

*# for k,v in counter.items():*

*# per = v / len(Y\_test2) \* 100*

*# print('Class=%d, n=%d (%.3f%%)' % (k, v, per))*

*# # plot the distribution*

*# pyplot.bar(counter.keys(), counter.values())*

*# pyplot.show()*

X\_test2.shape

(6838, 6)

Y\_test2.shape

(6838,)

Y\_test2.value\_counts()

1 4233

3 1044

2 827

4 456

5 278

Name: AQI\_RANKING, dtype: int64

**DETECTING AND REMOVING OUTLIERS BY GAUSSIAN RANGE AND IQR**

#detection outliers for train data

X\_train2.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 27348.0 | 1.797571 | 0.516778 | -0.916291 | 1.435085 | 1.629241 | 2.079442 | 4.919981 |
| **no2** | 27348.0 | 2.874994 | 0.495177 | -0.693147 | 2.484907 | 2.833213 | 3.265759 | 5.813832 |
| **rspm** | 27348.0 | 4.269130 | 0.517819 | 0.693147 | 3.988984 | 4.304065 | 4.584967 | 6.645091 |
| **spm** | 27348.0 | 5.104048 | 0.408568 | 2.079442 | 5.111988 | 5.135798 | 5.153292 | 7.170120 |
| **CO** | 27348.0 | 10.889571 | 42.363496 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 366.000000 |
| **Ozone** | 27348.0 | 151.492979 | 119.366715 | 0.000000 | 0.000000 | 166.000000 | 235.000000 | 366.000000 |

sns.boxplot(X\_train2.so2)

<AxesSubplot:xlabel='so2'>

upper\_limit=X\_train2['so2'].mean()+3\*X\_train2['so2'].std()

upper\_limit

3.3479044265671396

lower\_limit=X\_train2['so2'].mean()-3\*X\_train2['so2'].std()

lower\_limit

0.2472366844423084

*#removing outlier*

*#X\_train2.loc[X\_train2['SOi']>=3,'SOi']=3*

*#sns.boxplot(X\_train2.SOi)*

sns.boxplot(X\_train2.no2)

<AxesSubplot:xlabel='no2'>

upper\_limit=X\_train2['no2'].mean()+3\*X\_train2['no2'].std()

upper\_limit

4.360524776539017

lower\_limit=X\_train2['no2'].mean()-3\*X\_train2['no2'].std()

lower\_limit

1.3894639400730024

*#removing outlier*

*#X\_train2.loc[X\_train2['no2']>=4,'no2']=4*

*#sns.boxplot(X\_train2.no2)*

*#outlier detection*

sns.boxplot(X\_train2.rspm)

<AxesSubplot:xlabel='rspm'>

upper\_limit=X\_train2['rspm'].mean()+3\*X\_train2['rspm'].std()

upper\_limit

5.822586192040332

lower\_limit=X\_train2['rspm'].mean()-3\*X\_train2['rspm'].std()

lower\_limit

2.71567372872508

*#removing outlier*

*#X\_train2.loc[X\_train2['SPMi']>=5.8,'SPMi']=5.8*

*#outlier detection*

*#sns.boxplot(X\_train2.rspm)*

*#outlier detection*

sns.boxplot(X\_train2.spm)

<AxesSubplot:xlabel='spm'>

upper\_limit=X\_train2['spm'].mean()+3\*X\_train2['spm'].std()

upper\_limit

6.329752078545496

lower\_limit=X\_train2['spm'].mean()-3\*X\_train2['spm'].std()

lower\_limit

3.8783446601292124

#if be necessary then can be used for balancing

from imblearn.combine import SMOTETomek

from imblearn.under\_sampling import TomekLinks

*# Define SMOTE-Tomek Links*

*#resample=SMOTETomek(tomek=TomekLinks(sampling\_strategy='not majority'))*

*#x1, y1 = resample.fit\_resample(X\_train2, Y\_train2)*

*#x1.shape*

#detection outliers for test data

X\_test2.describe().T

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **so2** | 6838.0 | 1.805085 | 0.524496 | -0.916291 | 1.458615 | 1.629241 | 2.104134 | 5.429346 |
| **no2** | 6838.0 | 2.884252 | 0.492396 | 0.693147 | 2.484907 | 2.833213 | 3.273364 | 5.383577 |
| **rspm** | 6838.0 | 4.274689 | 0.510702 | 1.386294 | 3.988984 | 4.304065 | 4.595120 | 6.672033 |
| **spm** | 6838.0 | 5.101321 | 0.409937 | 1.609438 | 5.111988 | 5.135798 | 5.147494 | 7.170120 |
| **CO** | 6838.0 | -inf | NaN | -inf | NaN | NaN | NaN | 5.886104 |
| **Ozone** | 6838.0 | 152.162913 | 119.356349 | 0.000000 | 0.000000 | 167.000000 | 235.000000 | 366.000000 |

#since there is no massive difference 75% to max for the test data so no needs to be figured out and removeing outliers

**FEATURE SELECTION**

**selection by correlation**

import seaborn as sns

*#Using Pearson Correlation*

plt.figure(figsize=(6,4))

cor = X\_train2.corr()

sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap\_r)

plt.show()

*# we can select highly correlated features by fuctin given below*

**def** correlation(dataset, threshold):

col\_corr = set()

corr\_matrix = X\_train2.corr()

**for** i **in** range(len(corr\_matrix.columns)):

**for** j **in** range(i):

**if** abs(corr\_matrix.iloc[i, j]) > threshold:

colname = corr\_matrix.columns[i]

col\_corr.add(colname)

**return** col\_corr

corr\_features = correlation(X\_train2, 0.6)

len(set(corr\_features))

0

corr\_features

set()

*#if needs drop those collumns are really highly correlated to each others*

*#X\_train.drop(corr\_features,axis=1)*

*#X\_test.drop(corr\_features,axis=1)*

**selection by extratree classifier**

X\_train2.isnull().sum()

so2 0

no2 0

rspm 0

spm 0

CO 0

Ozone 0

dtype: int64

from sklearn.ensemble import ExtraTreesClassifier

import matplotlib.pyplot as plt

model=ExtraTreesClassifier()

model.fit(X\_train2,Y\_train2)

ExtraTreesClassifier

ExtraTreesClassifier()

print(model.feature\_importances\_)

[0.14162898 0.18765814 0.24016745 0.12928851 0.0563205 0.24493643]

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

ranked\_features=pd.Series(model.feature\_importances\_,index=X\_train2.columns)

ranked\_features.nlargest(6).plot(kind='barh')

plt.show()

#since the column of CO importance is less than 5 so we can drop that

imp\_features=['so2', 'no2', 'rspm', 'spm','Ozone']

X\_train2=pd.DataFrame(data=X\_train2,columns=imp\_features)

X\_train2.head()

|  | **so2** | **no2** | **rspm** | **spm** | **Ozone** |
| --- | --- | --- | --- | --- | --- |
| **13176** | 1.386294 | 2.197225 | 4.127134 | 5.135798 | 324.0 |
| **20732** | 2.397895 | 2.197225 | 3.258097 | 5.135798 | 128.0 |
| **21422** | 1.386294 | 2.197225 | 3.871201 | 5.135798 | 0.0 |
| **24484** | 1.791759 | 4.007333 | 4.875197 | 5.135798 | 0.0 |
| **32600** | 1.945910 | 2.772589 | 3.526361 | 5.135798 | 0.0 |

X\_train2.shape

(27348, 5)

Y\_train2.shape

(27348,)

#drop CO column also from test data

X\_test2=pd.DataFrame(data=X\_test2,columns=imp\_features)

X\_test2.head()

|  | **so2** | **no2** | **rspm** | **spm** | **Ozone** |
| --- | --- | --- | --- | --- | --- |
| **26268** | 2.197225 | 3.178054 | 3.871201 | 5.135798 | 160.0 |
| **11422** | 1.945910 | 3.178054 | 4.488636 | 5.135798 | 0.0 |
| **12394** | 2.564949 | 2.890372 | 4.454347 | 5.135798 | 182.0 |
| **4897** | 1.629241 | 2.833213 | 3.496508 | 4.477337 | 74.0 |
| **12272** | 1.098612 | 2.484907 | 4.997212 | 5.135798 | 177.0 |

X\_test2.shape

(6838, 5)

Y\_test2.shape

(6838,)

## **Appendix D Models Building**

**MODELS BUILDING**

**LOGISTIC REGRESSION**

X\_train2.shape

(27348, 5)

Y\_train2.shape

(27348,)

Y\_train2.value\_counts()

1 16843

3 4190

2 3464

4 1707

5 1144

Name: AQI\_RANKING, dtype: int64

from sklearn.preprocessing import MinMaxScaler

MM = MinMaxScaler()

X\_train = MM.fit\_transform(X\_train2)

X\_test = MM.transform(X\_test2)

logis\_reg = LogisticRegression(multi\_class='ovr',n\_jobs=-1).fit(X\_train, Y\_train2)

*#prediction on train data*

train\_pred1 = logis\_reg.predict(X\_train)

*#accuracy on train data*

print("model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred1))

*#prediction on test data*

test\_pred1 = logis\_reg.predict(X\_test)

*#accuracy on test data*

print("model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred1))

model accuracy on train is: 0.611013602457218

model accuracy on test is: 0.6137759578824218

from sklearn.metrics import confusion\_matrix,classification\_report,accuracy\_score

print(confusion\_matrix(Y\_test2, test\_pred1))

[[4168 0 65 0 0]

[ 817 0 10 0 0]

[1015 0 29 0 0]

[ 447 0 9 0 0]

[ 253 0 25 0 0]]

print(classification\_report(Y\_test2, test\_pred1))

precision recall f1-score support

1 0.62 0.98 0.76 4233

2 0.00 0.00 0.00 827

3 0.21 0.03 0.05 1044

4 0.00 0.00 0.00 456

5 0.00 0.00 0.00 278

accuracy 0.61 6838

macro avg 0.17 0.20 0.16 6838

weighted avg 0.42 0.61 0.48 6838

*#logis\_reg.predict([[1.609438,3.218876,4.543295,5.043425,150.0]])*

logis\_reg.predict([[1.435085,2.740840,3.988984,4.997212,218.0]])

array([1], dtype=int64)

sns.distplot(Y\_test2-test\_pred1)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

#tuning for the logistic regression

*# params grid*

params\_grid = {

'penalty' : ['l1','l2'],

'C' : np.logspace(-3,3,7),

'solver' : ['newton-cg', 'lbfgs', 'liblinear'],

}

tuning\_LR= GridSearchCV(logis\_reg,

param\_grid = params\_grid,

scoring='accuracy',

cv=10)

tuning\_LR.fit(X\_train,Y\_train2)

GridSearchCV

estimator: LogisticRegression

LogisticRegression

print("Tuned Hyperparameters :", tuning\_LR.best\_params\_)

print("Accuracy :",tuning\_LR.best\_score\_)

Tuned Hyperparameters : {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}

Accuracy : 0.6158768517243086

prdlr\_tr=tuning\_LR.predict(X\_train)

prdlr\_te=tuning\_LR.predict(X\_test)

print(confusion\_matrix(Y\_test2,prdlr\_te))

print("Accuracy Score for train {}".format(accuracy\_score(Y\_train2,prdlr\_tr)))

print("Accuracy Score for test {}".format(accuracy\_score(Y\_test2,prdlr\_te)))

print("Classification report: {}".format(classification\_report(Y\_test2,prdlr\_te)))

[[4233 0 0 0 0]

[ 827 0 0 0 0]

[1044 0 0 0 0]

[ 456 0 0 0 0]

[ 278 0 0 0 0]]

Accuracy Score for train 0.6158768465701331

Accuracy Score for test 0.6190406551623282

Classification report: precision recall f1-score support

1 0.62 1.00 0.76 4233

2 0.00 0.00 0.00 827

3 0.00 0.00 0.00 1044

4 0.00 0.00 0.00 456

5 0.00 0.00 0.00 278

accuracy 0.62 6838

macro avg 0.12 0.20 0.15 6838

weighted avg 0.38 0.62 0.47 6838

**Decision Tree Classifier**

*#fit training data*

DT = DecisionTreeClassifier(random\_state=42).fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred2 = DT.predict(X\_train)

*#accuracy on train data*

print("model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred2))

*#prediction on test data*

test\_pred2 = DT.predict(X\_test)

*#accuracy on test data*

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred2))

model accuracy on train is: 0.9816805616498464

Model accuracy on test is: 0.610412401286926

print(confusion\_matrix(Y\_test2, test\_pred2))

[[3319 372 363 95 84]

[ 350 244 163 44 26]

[ 386 173 339 84 62]

[ 99 36 99 203 19]

[ 96 26 70 17 69]]

*#for DT before*

print(classification\_report(Y\_test2, test\_pred2))

precision recall f1-score support

1 0.78 0.78 0.78 4233

2 0.29 0.30 0.29 827

3 0.33 0.32 0.33 1044

4 0.46 0.45 0.45 456

5 0.27 0.25 0.26 278

accuracy 0.61 6838

macro avg 0.42 0.42 0.42 6838

weighted avg 0.61 0.61 0.61 6838

DT.predict([[1.609438,3.218876,4.543295,5.043425,150.0]])

array([1], dtype=int64)

DT.predict([[1.945910,2.484907,3.496508,5.043425,0.0]])

array([1], dtype=int64)

sns.distplot(Y\_test2-test\_pred2)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

*## Hyper Parameter Optimization for decision tree*

param\_dist = {"max\_depth": [1,2,3,4,5,6,7,8,9],

"max\_features": range(1, 25),

"min\_samples\_leaf": range(1, 25),

"criterion": ["gini", "entropy"]}

*## Hyperparameter optimization using GridSearchCV*

from sklearn.model\_selection import RandomizedSearchCV

rs\_dt=RandomizedSearchCV(DT, param\_dist, cv=10)

rs\_dt.fit(X\_train, Y\_train2)

RandomizedSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier

print("Tuned decision Tree Parameters: {}".format(rs\_dt.best\_params\_))

print("Best score is {}".format(rs\_dt.best\_score\_))

Tuned decision Tree Parameters: {'min\_samples\_leaf': 22, 'max\_features': 14, 'max\_depth': 9, 'criterion': 'entropy'}

Best score is 0.6594632958385769

from sklearn.metrics import accuracy\_score

y\_predtrain=rs\_dt.predict(X\_train)

y\_pred=rs\_dt.predict(X\_test)

print(confusion\_matrix(Y\_test2,y\_pred))

print("Accuracy Score for train {}".format(accuracy\_score(Y\_train2,y\_predtrain)))

print("Accuracy Score for test {}".format(accuracy\_score(Y\_test2,y\_pred)))

print("Classification report: {}".format(classification\_report(Y\_test2,y\_pred)))

[[3779 95 213 124 22]

[ 488 141 121 74 3]

[ 596 32 275 115 26]

[ 120 9 24 297 6]

[ 221 3 24 6 24]]

Accuracy Score for train 0.676941641070645

Accuracy Score for test 0.6604270254460368

Classification report: precision recall f1-score support

1 0.73 0.89 0.80 4233

2 0.50 0.17 0.25 827

3 0.42 0.26 0.32 1044

4 0.48 0.65 0.55 456

5 0.30 0.09 0.13 278

accuracy 0.66 6838

macro avg 0.49 0.41 0.41 6838

weighted avg 0.62 0.66 0.62 6838

**Random Forest**

RF=RandomForestClassifier().fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred3 = RF.predict(X\_train)

*#accuracy on train data*

print("model accuracy on train data is: ", accuracy\_score(Y\_train2, train\_pred3))

*#prediction on test data*

test\_pred3 = RF.predict(X\_test)

*#accuracy on test data*

print("model accuracy on test data is: ", accuracy\_score(Y\_test2, test\_pred3))

model accuracy on train data is: 0.9816805616498464

model accuracy on test data is: 0.6692015209125475

print(confusion\_matrix(Y\_test2, test\_pred3))

[[3671 190 241 63 68]

[ 397 223 142 38 27]

[ 418 103 356 93 74]

[ 90 21 73 254 18]

[ 117 19 60 10 72]]

print(classification\_report(Y\_test2, test\_pred3))

precision recall f1-score support

1 0.78 0.87 0.82 4233

2 0.40 0.27 0.32 827

3 0.41 0.34 0.37 1044

4 0.55 0.56 0.56 456

5 0.28 0.26 0.27 278

accuracy 0.67 6838

macro avg 0.48 0.46 0.47 6838

weighted avg 0.64 0.67 0.65 6838

RF.predict([[1.609438,3.218876,4.543295,5.043425,150.0]])

array([1], dtype=int64)

RF.predict([[1.945910,2.484907,3.496508,5.043425,0.0]])

array([1], dtype=int64)

sns.distplot(Y\_test2-test\_pred3)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

#TUNING FOR RANDOM FOREST

from sklearn.model\_selection import RandomizedSearchCV

*# Number of trees of random forest*

n\_estimators = [int(x) **for** x **in** np.linspace(start = 10, stop = 80, num = 10)]

*# Number of features to consider at every split*

max\_features = ['auto', 'sqrt']

*# Maximum number of levels in tree*

max\_depth = [2,4]

*# Minimum number of samples required to split a node*

min\_samples\_split = [2, 5]

*# Minimum number of samples required at each leaf node*

min\_samples\_leaf = [1, 2]

*# Method of selecting samples for training each tree*

bootstrap = [True, False]

*# Create the param grid*

param\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

print(param\_grid)

{'n\_estimators': [10, 17, 25, 33, 41, 48, 56, 64, 72, 80], 'max\_features': ['auto', 'sqrt'], 'max\_depth': [2, 4], 'min\_samples\_split': [2, 5], 'min\_samples\_leaf': [1, 2], 'bootstrap': [True, False]}

rf\_Model = RandomForestClassifier()

rf\_random= RandomizedSearchCV(estimator = rf\_Model, param\_distributions = param\_grid, cv = 10, verbose=2, n\_jobs =-1)

rf\_random.fit(X\_train,Y\_train2)

Fitting 10 folds for each of 10 candidates, totalling 100 fits

RandomizedSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

print('best parameters:',rf\_random.best\_params\_)

print('best score:',rf\_random.best\_score\_)

best parameters: {'n\_estimators': 56, 'min\_samples\_split': 5, 'min\_samples\_leaf': 2, 'max\_features': 'auto', 'max\_depth': 4, 'bootstrap': False}

best score: 0.6158768517243086

from sklearn.metrics import accuracy\_score

y\_predtrain=rf\_random.predict(X\_train)

y\_pred=rf\_random.predict(X\_test)

print(confusion\_matrix(Y\_test2,y\_pred))

print("Accuracy Score for train {}".format(accuracy\_score(Y\_train2,y\_predtrain)))

print("Accuracy Score for test {}".format(accuracy\_score(Y\_test2,y\_pred)))

print("Classification report: {}".format(classification\_report(Y\_test2,y\_pred)))

[[4233 0 0 0 0]

[ 827 0 0 0 0]

[1044 0 0 0 0]

[ 456 0 0 0 0]

[ 278 0 0 0 0]]

Accuracy Score for train 0.6158768465701331

Accuracy Score for test 0.6190406551623282

Classification report: precision recall f1-score support

1 0.62 1.00 0.76 4233

2 0.00 0.00 0.00 827

3 0.00 0.00 0.00 1044

4 0.00 0.00 0.00 456

5 0.00 0.00 0.00 278

accuracy 0.62 6838

macro avg 0.12 0.20 0.15 6838

weighted avg 0.38 0.62 0.47 6838

**K-Nearest Neighbours**

KNN = KNeighborsClassifier(n\_neighbors=1)

KNN.fit(X\_train2,Y\_train2)

*#predictION on train data*

train\_pred4 = KNN.predict(X\_train2)

*#accuracy on train data*

print("Model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred4))

*#prediction on test data*

test\_pred4 = KNN.predict(X\_test2)

*#accuracy on test data*

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred4))

Model accuracy on train is: 0.9795231826824631

Model accuracy on test is: 0.5845276396607195

KNN.predict([[1,1.2,3.12,0.0,76]])

*# Predictions on random values*

array([4], dtype=int64)

KNN.predict([[325.7,345,798.182,203,0.0]])

*# Predictions on random values*

array([1], dtype=int64)

print(confusion\_matrix(Y\_test2,test\_pred4))

[[3194 405 394 140 100]

[ 373 229 157 41 27]

[ 399 159 342 81 63]

[ 155 37 77 162 25]

[ 97 23 71 17 70]]

print(classification\_report(Y\_test2,test\_pred4))

precision recall f1-score support

1 0.76 0.75 0.76 4233

2 0.27 0.28 0.27 827

3 0.33 0.33 0.33 1044

4 0.37 0.36 0.36 456

5 0.25 0.25 0.25 278

accuracy 0.58 6838

macro avg 0.39 0.39 0.39 6838

weighted avg 0.59 0.58 0.59 6838

sns.distplot(Y\_test2-test\_pred4)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

#TUNING FOR KNN

*#selecting best K value*

err\_rate = []

**for** i **in** range(1,40):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(X\_train2,Y\_train2)

pred\_i = knn.predict(X\_test2)

err\_rate.append(np.mean(pred\_i != Y\_test2))

err\_rate

[0.4154723603392805,

0.38871014916642294,

0.3831529687042995,

0.3843229014331676,

0.38286048552208246,

0.3778882714243931,

0.3775957882421761,

0.37437847323778883,

0.3786194793799357,

0.37598713073998247,

0.3754021643755484,

0.3762796139221995,

0.3755484059666569,

0.37598713073998247,

0.3745247148288973,

0.3751096811933314,

0.37598713073998247,

0.3736472652822463,

0.3758408891488739,

0.37437847323778883,

0.3745247148288973,

0.3740859900555718,

0.3736472652822463,

0.37247733255337817,

0.37218484937116114,

0.3727698157355952,

0.37159988300672714,

0.3727698157355952,

0.37086867505118454,

0.3691137759578824,

0.37247733255337817,

0.36984498391342496,

0.36984498391342496,

0.3729160573267037,

0.37218484937116114,

0.37072243346007605,

0.37086867505118454,

0.37159988300672714,

0.3718923661889441]

plt.figure(figsize=(10,6))

plt.plot(range(1,40),err\_rate,color='blue', linestyle='dashed', marker='o',

markerfacecolor='black', markersize=10)

plt.title('error rate vs K Value')

plt.xlabel('K')

plt.ylabel('error rate')

Text(0, 0.5, 'error rate')

*#it is noticeable that after k=32 the error rate tends to rise so needs to be checked k=30*

*#creating model assigning n\_neighbors=30*

KNN = KNeighborsClassifier(n\_neighbors=30)

*#fitting data on model*

KNN.fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred4 = KNN.predict(X\_train)

*#accuracy on train data*

print("Model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred4))

*#prediction on test data*

test\_pred4 = KNN.predict(X\_test)

*#accuracy on test data*

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred4))

Model accuracy on train is: 0.6873994442006728

Model accuracy on test is: 0.6667154138637028

from sklearn.metrics import accuracy\_score

y\_predtrain=KNN.predict(X\_train)

y\_pred=KNN.predict(X\_test)

print(confusion\_matrix(Y\_test2,y\_pred))

print('\n\n')

print("Classification report: {}".format(classification\_report(Y\_test2,y\_pred)))

[[3769 81 197 123 63]

[ 521 155 103 39 9]

[ 555 54 303 74 58]

[ 138 8 32 260 18]

[ 149 2 54 1 72]]

Classification report: precision recall f1-score support

1 0.73 0.89 0.80 4233

2 0.52 0.19 0.28 827

3 0.44 0.29 0.35 1044

4 0.52 0.57 0.55 456

5 0.33 0.26 0.29 278

accuracy 0.67 6838

macro avg 0.51 0.44 0.45 6838

weighted avg 0.63 0.67 0.63 6838

sns.distplot(Y\_test2-y\_pred)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

**MULTINOMIAL NAIVE BAYES**

from sklearn.naive\_bayes import MultinomialNB

*#fit the model on train data*

MNB = MultinomialNB()

MNB.fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred5 = MNB.predict(X\_train)

*#accuracy on train data*

print("model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred5))

*#prediction on test data*

test\_pred5 = MNB.predict(X\_test)

*#accuracy on test data*

print("model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred5))

model accuracy on train is: 0.6158768465701331

model accuracy on test is: 0.6190406551623282

print(confusion\_matrix(Y\_test2,test\_pred5))

[[4233 0 0 0 0]

[ 827 0 0 0 0]

[1044 0 0 0 0]

[ 456 0 0 0 0]

[ 278 0 0 0 0]]

print(classification\_report(Y\_test2,test\_pred5))

precision recall f1-score support

1 0.62 1.00 0.76 4233

2 0.00 0.00 0.00 827

3 0.00 0.00 0.00 1044

4 0.00 0.00 0.00 456

5 0.00 0.00 0.00 278

accuracy 0.62 6838

macro avg 0.12 0.20 0.15 6838

weighted avg 0.38 0.62 0.47 6838

from sklearn.model\_selection import GridSearchCV

param={'alpha': [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]}

grid\_mnb=GridSearchCV(MNB,param,cv=10,error\_score='raise')

grid\_mnb.fit(X\_train,Y\_train2)

GridSearchCV

estimator: MultinomialNB

MultinomialNB

grid\_mnb.best\_params\_

{'alpha': 1e-05}

grid\_mnb.best\_score\_

0.6158768517243086

trprd\_mnb=grid\_mnb.predict(X\_train)

print('accuracy for train:',accuracy\_score(Y\_train2,trprd\_mnb))

teprd\_mnb=grid\_mnb.predict(X\_test)

print('accuracy for train:',accuracy\_score(Y\_test2,teprd\_mnb))

accuracy for train: 0.6158768465701331

accuracy for train: 0.6190406551623282

print(classification\_report(Y\_test2,teprd\_mnb))

precision recall f1-score support

1 0.62 1.00 0.76 4233

2 0.00 0.00 0.00 827

3 0.00 0.00 0.00 1044

4 0.00 0.00 0.00 456

5 0.00 0.00 0.00 278

accuracy 0.62 6838

macro avg 0.12 0.20 0.15 6838

weighted avg 0.38 0.62 0.47 6838

sns.distplot(Y\_test2-teprd\_mnb)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

**SUPPORT VECTOR MACHINE**

from sklearn.svm import SVC

svc = SVC()

svc.fit(X\_train, Y\_train2)

SVC

SVC()

*#prediction on train data*

train\_pred6 = svc.predict(X\_train)

*#accuracy on train data*

print("Model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred6))

Model accuracy on train is: 0.6348179025888547

*#predict on test data*

test\_pred6 = svc.predict(X\_test)

*#accuracy on test data*

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred6))

Model accuracy on test is: 0.6355659549575899

svc.predict([[1,1.2,3.12,0.0,48]])

*# Predictions on random values*

array([1], dtype=int64)

svc.predict([[1,4,3.12,0.0,48]])

*# Predictions on random values*

array([1], dtype=int64)

print(classification\_report(Y\_test2,test\_pred6))

precision recall f1-score support

1 0.64 0.98 0.77 4233

2 0.00 0.00 0.00 827

3 0.55 0.04 0.07 1044

4 0.53 0.39 0.45 456

5 0.33 0.00 0.01 278

accuracy 0.64 6838

macro avg 0.41 0.28 0.26 6838

weighted avg 0.53 0.64 0.52 6838

sns.distplot(Y\_test2-test\_pred6)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

#tuning

from sklearn.svm import SVC

sv=SVC()

params\_grid ={

"gamma": [0.1, 1.0, 10, 100],

"C": [0.1, 1.0, 10, 100],

'kernel': ['rbf', 'poly', 'sigmoid']}

*# This allows us to speed up the computation power by performing tasks in parallel*

rand\_svc = RandomizedSearchCV(sv, params\_grid, cv=3, verbose=2, n\_jobs=-1, n\_iter=5)

rand\_svc.fit(X\_train, Y\_train2)

Fitting 3 folds for each of 5 candidates, totalling 15 fits

RandomizedSearchCV

estimator: SVC

SVC

rand\_svc.best\_params\_

{'kernel': 'rbf', 'gamma': 0.01, 'C': 0.1}

rand\_svc.best\_score\_

0.6158768465701331

pred\_svctr=rand\_svc.predict(X\_train)

pred\_svcte=rand\_svc.predict(X\_test)

print('accuracy score for train svc :',accuracy\_score(Y\_train2,pred\_svctr))

print('accuracy score for test svc :',accuracy\_score(Y\_test2,pred\_svcte))

accuracy score for train svc : 0.6158768465701331

accuracy score for test svc : 0.6190406551623282

**GRADIENT BOOSTING CLASSIFIER**

from sklearn.ensemble import GradientBoostingClassifier

*#fit the model on train data*

GBC = GradientBoostingClassifier().fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred7 = GBC.predict(X\_train)

*#accuracy on train data*

print("Model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred7))

*#predict on test data*

test\_pred7 = GBC.predict(X\_test)

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred7))

Model accuracy on train is: 0.6907269270147726

Model accuracy on test is: 0.6823632641123135

print(confusion\_matrix(Y\_test2,test\_pred7))

[[3838 99 162 91 43]

[ 447 193 126 56 5]

[ 546 68 311 80 39]

[ 128 9 27 280 12]

[ 183 1 48 2 44]]

print(classification\_report(Y\_test2,test\_pred7))

precision recall f1-score support

1 0.75 0.91 0.82 4233

2 0.52 0.23 0.32 827

3 0.46 0.30 0.36 1044

4 0.55 0.61 0.58 456

5 0.31 0.16 0.21 278

accuracy 0.68 6838

macro avg 0.52 0.44 0.46 6838

weighted avg 0.64 0.68 0.65 6838

sns.distplot(Y\_test2-test\_pred7)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

lerrate\_gb = [0.05, 0.075, 0.1, 0.25,0.3, 0.5, 0.75,0.8,0.9,1]

**for** learning\_rate **in** lerrate\_gb:

gbc = GradientBoostingClassifier(n\_estimators=20, learning\_rate=learning\_rate, max\_features=2, max\_depth=2, random\_state=5)

gbc.fit(X\_train, Y\_train2)

print("learning rate is: ", learning\_rate)

print("Accuracy score for training: {0:.2f}".format(gbc.score(X\_train, Y\_train2)))

print("Accuracy score for test: {0:.2f}".format(gbc.score(X\_test, Y\_test2)))

learning rate is: 0.05

Accuracy score for training: 0.62

Accuracy score for test: 0.62

learning rate is: 0.075

Accuracy score for training: 0.62

Accuracy score for test: 0.62

learning rate is: 0.1

Accuracy score for training: 0.62

Accuracy score for test: 0.62

learning rate is: 0.25

Accuracy score for training: 0.64

Accuracy score for test: 0.64

learning rate is: 0.3

Accuracy score for training: 0.65

Accuracy score for test: 0.64

learning rate is: 0.5

Accuracy score for training: 0.66

Accuracy score for test: 0.66

learning rate is: 0.75

Accuracy score for training: 0.67

Accuracy score for test: 0.67

learning rate is: 0.8

Accuracy score for training: 0.67

Accuracy score for test: 0.67

learning rate is: 0.9

Accuracy score for training: 0.67

Accuracy score for test: 0.67

learning rate is: 1

Accuracy score for training: 0.67

Accuracy score for test: 0.67

from sklearn.ensemble import AdaBoostClassifier

*#fit the model on train data*

ABC = AdaBoostClassifier().fit(X\_train,Y\_train2)

*#prediction on train data*

train\_pred8 = ABC.predict(X\_train)

*#accuracy on train data*

print("Model accuracy on train is: ", accuracy\_score(Y\_train2, train\_pred8))

*#prediction on test data*

test\_pred8 = ABC.predict(X\_test)

*#accuracy on test data*

print("Model accuracy on test is: ", accuracy\_score(Y\_test2, test\_pred8))

Model accuracy on train is: 0.6339768904490274

Model accuracy on test is: 0.6371746124597836

print(confusion\_matrix(Y\_test2,test\_pred8))

[[3947 4 54 199 29]

[ 696 3 25 94 9]

[ 823 1 66 134 20]

[ 120 1 5 323 7]

[ 230 0 28 2 18]]

print(classification\_report(Y\_test2,test\_pred8))

precision recall f1-score support

1 0.68 0.93 0.79 4233

2 0.33 0.00 0.01 827

3 0.37 0.06 0.11 1044

4 0.43 0.71 0.53 456

5 0.22 0.06 0.10 278

accuracy 0.64 6838

macro avg 0.41 0.35 0.31 6838

weighted avg 0.55 0.64 0.54 6838

sns.distplot(Y\_test2-test\_pred8)

<AxesSubplot:xlabel='AQI\_RANKING', ylabel='Density'>

**Appendix E Roc\_Auc\_Score**

**ROC\_AUC\_Score**

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

#LOGISTIC REGRESSION

from sklearn.linear\_model import LogisticRegression

log\_classifier=LogisticRegression()

log\_classifier.fit(X\_train, Y\_train2)

ytrain\_pred\_lg = log\_classifier.predict\_proba(X\_train)

print('logistic regression train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred\_lg,multi\_class='ovr',average='weighted')))

ytest\_pred\_lg = log\_classifier.predict\_proba(X\_test)

print('logistic regression test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred\_lg,multi\_class='ovr',average='weighted')))

logistic regression train roc-auc: 0.6813156869999026

logistic regression test roc-auc: 0.6849230417812592

#RANDOM FOREST

*## Apply RandomForestClassifier*

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train, Y\_train2)

ytrain\_pred = rf\_model.predict\_proba(X\_train)

print('RF train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = rf\_model.predict\_proba(X\_test)

print('RF test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

RF train roc-auc: 0.9992887476874209

RF test roc-auc: 0.8537605271867521

#DECISION TREE CLASSIFIER

dt\_model = DecisionTreeClassifier()

dt\_model.fit(X\_train2, Y\_train2)

ytrain\_pred = dt\_model.predict\_proba(X\_train2)

print('DT train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = rf\_model.predict\_proba(X\_test2)

print('DT test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

DT train roc-auc: 0.9997167940998236

DT test roc-auc: 0.8555026882499169

#KNeighborsClassifier

knn\_model = KNeighborsClassifier()

knn\_model.fit(X\_train, Y\_train2)

ytrain\_pred = knn\_model.predict\_proba(X\_train)

print('KNN train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = knn\_model.predict\_proba(X\_test)

print('KNN test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

KNN train roc-auc: 0.9193773570677194

KNN test roc-auc: 0.7978294283229883

#GaussianNB

gnb\_model = MultinomialNB()

gnb\_model.fit(X\_train, Y\_train2)

ytrain\_pred = gnb\_model.predict\_proba(X\_train)

print('GNB train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = gnb\_model.predict\_proba(X\_test)

print('GNB test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

GNB train roc-auc: 0.6137631137419249

GNB test roc-auc: 0.6135973558015236

#GradientBoostingClassifier

gbc\_model = GradientBoostingClassifier()

gbc\_model.fit(X\_train, Y\_train2)

ytrain\_pred = gbc\_model.predict\_proba(X\_train)

print('GBC train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = gbc\_model.predict\_proba(X\_test)

print('GBC test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

GBC train roc-auc: 0.8750650038530623

GBC test roc-auc: 0.8675954868659398

#AdaBoostClassifier

abc\_model = AdaBoostClassifier()

abc\_model.fit(X\_train, Y\_train2)

ytrain\_pred = abc\_model.predict\_proba(X\_train)

print('ABC train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = abc\_model.predict\_proba(X\_test)

print('ABC test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

ABC train roc-auc: 0.6722979602380204

ABC test roc-auc: 0.6726712083885941

#Support Vector machine

svc\_model = SVC(probability=True)

svc\_model.fit(X\_train, Y\_train2)

ytrain\_pred = svc\_model.predict\_proba(X\_train)

print('SVC train roc-auc: {}'.format(roc\_auc\_score(Y\_train2, ytrain\_pred,multi\_class='ovr',average='weighted')))

ytest\_pred = svc\_model.predict\_proba(X\_test)

print('SVC test roc-auc: {}'.format(roc\_auc\_score(Y\_test2, ytest\_pred,multi\_class='ovr',average='weighted')))

SVC train roc-auc: 0.7925289294373973

SVC test roc-auc: 0.790247923594469