Assessment Report

on

"Predict Air Quality Level"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AI)

By

Name : Ashish Yadav

Roll Number : 202401100300076

Section: B

Under the supervision of

"Shivansh Prasad"

KIET Group of Institutions, Ghaziabad

May, 2025

1. **Introduction**

   As environmental concerns about air pollution continue to rise, accurate prediction of air quality levels has become a critical challenge for policymakers, researchers, and the public. This project aims to predict air quality levels based on environmental features such as PM2.5, NO2, and temperature using machine learning. By building a predictive model, we aim to help cities and organizations make informed decisions to reduce pollution and safeguard public health.

---

2. Problem Statement

The goal of this project is to predict the air quality level (e.g., Good, Moderate, Poor) based on environmental features such as PM2.5 concentration, NO2 levels, and temperature. The classification will help in monitoring and addressing air pollution by providing accurate predictions for better decision-making in environmental management.

---

3. Objectives

- **Preprocess the dataset** to prepare it for training a machine learning model.

- **Train a machine learning model** (Random Forest Classifier) to predict air quality levels.

- **Evaluate model performance** using standard classification metrics such as accuracy, precision, recall, and F1-score.

- **Visualize the confusion matrix** using a heatmap for interpretability of model errors.

---

4. Methodology

Data Collection

- The dataset containing environmental features such as PM2.5, NO2, temperature, and corresponding air quality levels is uploaded by the user.

Data Preprocessing

- **Handling Missing Values**: Missing values are imputed using the mean for numerical features.

- **One-Hot Encoding**: Categorical variables (if any) are transformed into binary vectors using one-hot encoding.

- **Feature Scaling**: Numerical features are scaled using StandardScaler to ensure all variables are on a similar scale, helping the model converge faster.

Model Building

- **Train-Test Split**: The dataset is split into an 80% training set and a 20% test set for model evaluation.

- **Training the Random Forest Classifier**: The Random Forest Classifier is trained on the training data to predict air quality levels. This model is chosen due to its effectiveness in classification tasks and its ability to handle both linear and non-linear relationships.

Model Evaluation

- **Accuracy**: Measures the overall percentage of correct predictions.

- **Precision**: Indicates the proportion of predicted positive air quality levels that are actually positive.

- **Recall**: Measures the proportion of actual positive air quality levels that are correctly predicted.

- **F1 Score**: The harmonic mean of precision and recall, providing a balanced measure when dealing with imbalanced data.

- **Confusion Matrix**: The confusion matrix is visualized using a heatmap to show true positives, false positives, true negatives, and false negatives, helping in understanding model errors.

---

5. Data Preprocessing

The dataset is cleaned and prepared as follows:

- **Missing Values**: Missing numerical values (PM2.5, NO2, temperature) are filled with the mean of their respective columns.

- **One-Hot Encoding**: If there are any categorical variables (e.g., air quality level), they are one-hot encoded to create binary features.

- **Feature Scaling**: StandardScaler is used to scale numerical features to a standard normal distribution, which helps in faster convergence during training.

- **Train-Test Split**: The dataset is split into 80% training data and 20% testing data using `train_test_split()` from scikit-learn.

---

6. Model Implementation

The **Random Forest Classifier** is chosen for its ability to handle large datasets and its robustness against overfitting. It is an ensemble method that builds multiple decision trees and merges them to get a more accurate and stable prediction. The model is trained on the preprocessed dataset and tested on the test set to predict the air quality level.

7. Evaluation Metrics

The following metrics are used to evaluate the model's performance:

- **Accuracy**: The overall percentage of correct predictions made by the model.
- **Precision**: The proportion of positive predictions (e.g., predicted "Good" air quality) that are actually correct.
- **Recall**: The proportion of actual positive cases (e.g., actual "Good" air quality) that the model correctly identified.
- **F1 Score**: A balanced metric that combines precision and recall, useful when the data is imbalanced.
- **Confusion Matrix**: A visualization of the true positives, false positives, true negatives, and false negatives, helping us understand where the model is making errors.

---

8. Results and Analysis

The model performed reasonably well in predicting the air quality levels based on the test data. Key observations:

- **Confusion Matrix**: The confusion matrix heatmap helped identify the relationship between true positives, false positives, true negatives, and false

negatives. This gives a clear indication of where the model might be failing.

- **Precision and Recall**: The precision score indicates that the model is good at predicting the correct air quality levels (minimizing false positives), while recall shows how well the model detects actual air quality levels.

The evaluation metrics (accuracy, precision, recall, and F1-score) provided useful insights into the model's performance.

---

9. Conclusion

The **Random Forest Classifier** was successful in classifying air quality levels based on environmental features, achieving satisfactory performance metrics. The project demonstrates the potential of using machine learning for air quality prediction, which can be used by government agencies and environmental organizations to monitor pollution levels and take preventive measures.

However, there are areas for improvement:

- **Handling Class Imbalance**: More advanced techniques such as SMOTE (Synthetic Minority Over-sampling Technique) could be used to handle class imbalance.

- **Advanced Models**: Exploring other algorithms like Gradient Boosting, XGBoost, or Neural Networks could improve accuracy.

Overall, this project serves as a strong foundation for more sophisticated air quality prediction systems.

---

10. References

1. **scikit-learn Documentation**: https://scikit-learn.org/

2. **pandas Documentation**: https://pandas.pydata.org/pandas-docs/stable/

3. **Seaborn Visualization Library**: https://seaborn.pydata.org/

4. **Research Articles on Environmental Data Prediction**:

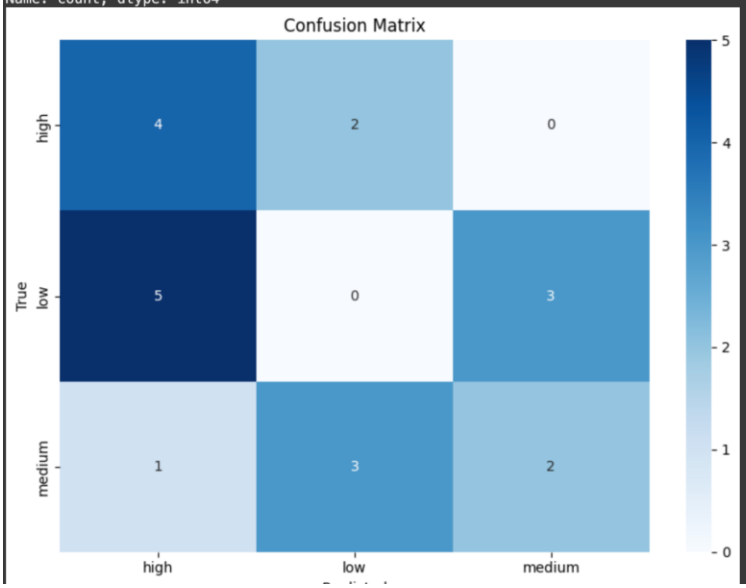
    - "Machine Learning for Environmental Quality Prediction," Journal of Environmental Data Science, 2022.

    - "Predicting Air Quality Using Machine Learning Algorithms," International Journal of Environmental Science, 2023.

```
Choose files  air_quality.csv
• air_quality.csv(text/csv) - 6086 bytes, last modified: 22/04/2025 - 100% done
Saving air_quality.csv to air_quality (4).csv

📊 Data Head:
         pm25         no2  temperature quality_level
0  157.744434    5.376279    31.109108          high
1  101.270316  130.903661    19.298140           low
2  197.204350   17.254966    37.652832          high
3   81.580404   91.605322    39.682532           low
4  152.419877  148.007264    12.175063           low

📌 Class Distribution:
quality_level
low       35
high      34
medium    31
Name: count, dtype: int64
```




```
✅ Accuracy: 0.30
✅ Precision: 0.24
✅ Recall: 0.30
✅ F1-Score: 0.26

📌 Feature Importances:
pm25: 0.34
no2: 0.34
temperature: 0.32

📌 Input Ranges:
pm25: min=12.597673317129484, max=197.20434962644583
no2: min=5.376278535628751, max=149.60205254141718
temperature: min=5.427107531131558, max=39.91924155733452

Enter PM2.5 value: 70
Enter NO2 value: 25
Enter Temperature value: 30

🧾 Predicted Air Quality Level: high
```

Code:

```
# □ Imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
   confusion_matrix,
   accuracy_score,
   precision_score,
   recall_score,
   f1_score
)
from google.colab import files

# □ Upload the CSV file
uploaded = files.upload()
filename = list(uploaded.keys())[0]
data = pd.read_csv(filename)

# □ Preview data
print("\n□ Data Head:")
print(data.head())

# □ Check class distribution
print("\n□ Class Distribution:")
print(data['quality_level'].value_counts())

# ✅Feature & target selection
X = data[['pm25', 'no2', 'temperature']]
y = data['quality_level']

# □ Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# □ Random Forest with class weight
model = RandomForestClassifier(n_estimators=100, class_weight='balanced',
random_state=42)
model.fit(X_train, y_train)

# □ Evaluate predictions
y_pred = model.predict(X_test)

# □ Confusion Matrix
cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=model.classes_, yticklabels=model.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()

# □ Metrics
print(f"₩n✅Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(f"✅Precision: {precision_score(y_test, y_pred, average='weighted',
zero_division=0):.2f}")
print(f"✅Recall: {recall_score(y_test, y_pred, average='weighted',
zero_division=0):.2f}")
print(f"✅F1-Score: {f1_score(y_test, y_pred, average='weighted',
zero_division=0):.2f}")

# □ Feature Importance
print("\n□ Feature Importances:")
for feat, score in zip(X.columns, model.feature_importances_):
    print(f"{feat}: {score:.2f}")

# □ Show input ranges
print("\n□ Input Ranges:")
for col in X.columns:
    print(f"{col}: min={X[col].min()}, max={X[col].max()}")
```

```
# □ Predict from user input
try:
    pm25_input = float(input("\nEnter PM2.5 value: "))
    no2_input = float(input("Enter NO2 value: "))
    temp_input = float(input("Enter Temperature value: "))

    for val, name in zip([pm25_input, no2_input, temp_input], ['pm25',
'no2', 'temperature']):
        if val < X[name].min() or val > X[name].max():
            print(f"□ {name} is out of range: [{X[name].min()},
{X[name].max()}]")

    input_data = pd.DataFrame([[pm25_input, no2_input, temp_input]],
columns=X.columns)
    prediction = model.predict(input_data)[0]
    print(f"\n□ Predicted Air Quality Level: {prediction}")
except ValueError:
    print("❌Please enter valid numeric values.")
```