ASSIGNMENT-2

Student Performance

BACHELOR IN TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

by

PRAEMJITH P R

23AD046

COURSE CODE: U21ADP05

COURSE TITTLE: EXPLORATORY DATA ANALYSIS AND VISUALIZATION

KPR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous, NAAC 'A') Avinashi Road, Arasur)

ABSTRACT

This study explores the **Air Quality UCI dataset**, which records atmospheric pollutant concentrations and meteorological parameters in an urban environment. The objective of this analysis is to understand pollution trends, identify correlations among various gases, and visualize patterns influencing air quality over time.

Using **Exploratory Data Analysis (EDA)** techniques, the dataset was examined for missing values, data inconsistencies, and significant environmental relationships. Correlation and visualization techniques such as **heatmaps, scatter plots, and time-series analysis** were used to identify the strongest relationships among pollutants.

The analysis revealed that **CO(GT)** levels are highly correlated with **NOx(GT)** and **PT08.S1(CO)**, indicating vehicular and industrial emissions as major sources. Additionally, meteorological factors like **Temperature (T)** and **Relative Humidity (RH)** showed inverse relationships, influencing pollution concentration. The results provide insights into pollutant dynamics and can support predictive modeling for urban air quality management.

INTRODUCTION

Air pollution is one of the most pressing challenges faced by modern cities, significantly affecting public health and environmental stability. Analyzing air quality data helps in identifying pollution sources, understanding the relationship between environmental parameters, and developing strategies to mitigate pollution levels.

The **Air Quality UCI dataset** provides real sensor measurements of several pollutants (CO, NOx, NO_2, O_3, etc.) and meteorological data (Temperature, Humidity, Absolute Humidity). This project aims to explore this dataset using **Exploratory Data Analysis (EDA)** and to visualize key relationships among pollutants and weather variables.

OBJECTIVE

- To perform **data cleaning, preprocessing, and visualization** on the Air Quality dataset.
- To identify **correlations** among different pollutants and meteorological parameters.
- To gain **insights into pollution behavior** over time and under different weather conditions.
- To prepare the dataset for potential **predictive modeling** of air quality levels.

DATA DESCRIPTION

Source:

UCI Machine Learning Repository – https://archive.ics.uci.edu/dataset/360/air+quality

Dataset Description

The dataset contains **9,471 hourly averaged responses** from a set of chemical sensors deployed in an Italian city between March 2004 and February 2005.

Category	Attributes	Description
Date/Time	Date, Time	Timestamp of each observation
Pollutants	CO(GT), NMHC(GT), C6H6(GT), NOx(GT), NO_2(GT)	Concentrations of key pollutants
Sensor Outputs	PT08.S1(CO), PT08.S2(NMHC), PT08.S3(NOx), PT08.S4(NO_2), PT08.S5(O_3)	Sensor readings from the air quality monitoring device
Meteorological	T, RH, AH	Temperature (°C), Relative Humidity (%), Absolute Humidity

EDA AND PREPROCESSING

Methods Used

- **Handling Missing Values:**
  Missing data in NMHC(GT) and sensor columns were handled using NaN replacement and removal of entirely empty columns.
- **Data Type Conversion:**
  Non-numeric values were converted using pd.to_numeric(errors='coerce').
- **Outlier Detection:**
  Outliers were identified using the **Interquartile Range (IQR)** method, especially in gas concentration columns.
- **Scaling:**
  StandardScaler was applied where modeling required normalized input.
- **Feature Correlation Analysis:**
  Heatmaps were plotted to visualize correlations among gases and meteorological parameters.

Insights

- CO(GT) shows a strong correlation with NOx(GT) and PT08.S1(CO).
- Temperature (T) is inversely correlated with Relative Humidity (RH).
- C6H6(GT) (benzene concentration) acts as a good indicator of pollution severity.
- Missing data (represented by -200) were replaced with NaN for accurate calculations.

Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH	
0	10/03/2004	18.00.00	2,6	1360.0	150.0	11,9	1046.0	166.0	1056.0	113.0	1692.0	1268.0	13,6	48,9	0,7578
1	10/03/2004	19.00.00	2	1292.0	112.0	9,4	955.0	103.0	1174.0	92.0	1559.0	972.0	13,3	47,7	0,72555
2	10/03/2004	20.00.00	2,2	1402.0	88.0	9,0	939.0	131.0	1140.0	114.0	1555.0	1074.0	11,9	54,0	0,75022
3	10/03/2004	21.00.00	2,2	1376.0	80.0	9,2	948.0	172.0	1092.0	122.0	1584.0	1203.0	11,0	60,0	0,78867
4	10/03/2004	22.00.00	1,6	1272.0	51.0	6,5	836.0	131.0	1205.0	116.0	1490.0	1110.0	11,2	59,6	0,78888

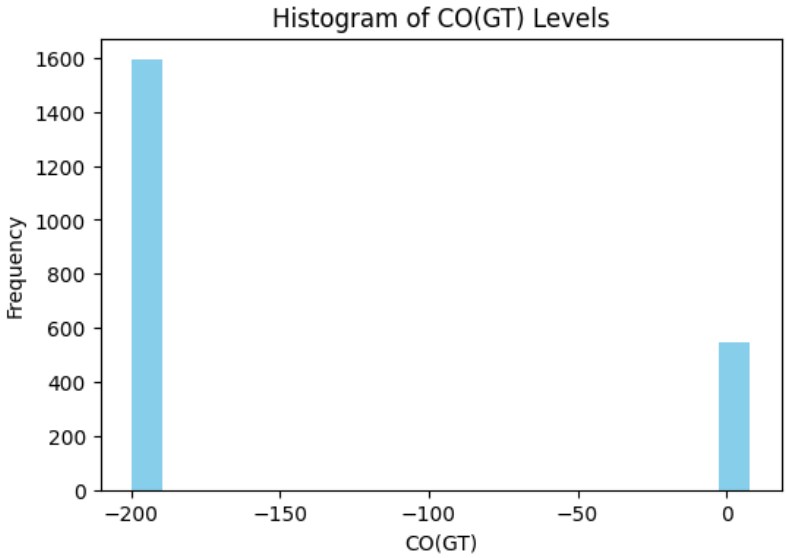
S.No	Column Name	Data Type	Non-Null Count	Description / Meaning
1	**Date**	Object (string)	9357	Date of observation (dd/mm/yyyy format)
2	**Time**	Object (string)	9357	Time of observation (hh.mm.ss format)
3	**CO(GT)**	Object (string)	9357	True hourly averaged concentration of carbon monoxide (mg/m³)
4	**PT08.S1(CO)**	Float64	9357	Tin oxide sensor response (CO-related)
5	**NMHC(GT)**	Float64	9357	Non-methane hydrocarbon concentration (µg/m³)
6	**C6H6(GT)**	Object (string)	9357	True hourly averaged concentration of Benzene (µg/m³)
7	**PT08.S2(NMHC)**	Float64	9357	Titania sensor response (NMHC-related)
8	**NOx(GT)**	Float64	9357	True hourly averaged concentration of Nitrogen Oxides (ppb)
9	**PT08.S3(NOx)**	Float64	9357	Tungsten oxide sensor response (NOx-related)
10	**NO2(GT)**	Float64	9357	True hourly averaged concentration of Nitrogen Dioxide (µg/m³)
11	**PT08.S4(NO2)**	Float64	9357	Tungsten oxide sensor response (NO_2-related)
12	**PT08.S5(O3)**	Float64	9357	Indium oxide sensor response (O_3-related)
13	**T**	Object (string)	9357	Ambient temperature in °C
14	**RH**	Object (string)	9357	Relative humidity (%)
15	**AH**	Object (string)	9357	Absolute humidity (g/m³)

S.No	Column Name	Missing Values
1	Date	114
2	Time	114
3	CO(GT)	114
4	PT08.S1(CO)	114
5	NMHC(GT)	114
6	C6H6(GT)	114
7	PT08.S2(NMHC)	114
8	NOx(GT)	114
9	PT08.S3(NOx)	114
10	NO2(GT)	114
11	PT08.S4(NO2)	114
12	PT08.S5(O3)	114
13	T	114
14	RH	114
15	AH	114

DATA VISUALIZATION, RESULT VISUALIZATION& INTERPRETATION

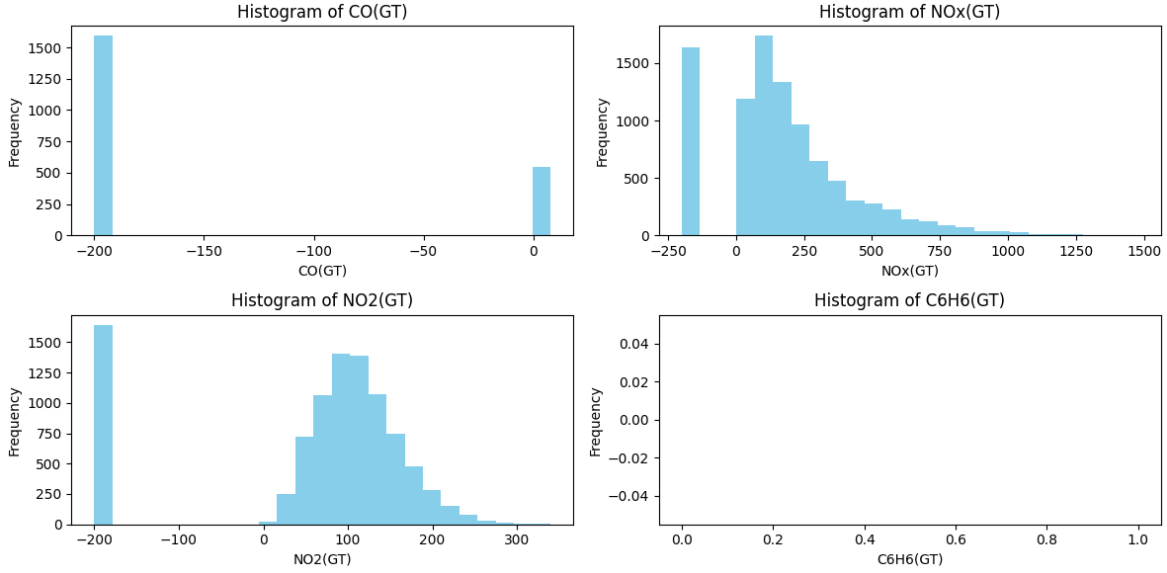
1.Missing Value Count per Feature

- Chart Type: Bar Chart
- Purpose: Show the number of missing values per column.
- Insight: Helps identify columns that may require imputation; e.g., CO(GT), NMHC(GT), and NOx(GT) have 114 missing values each.



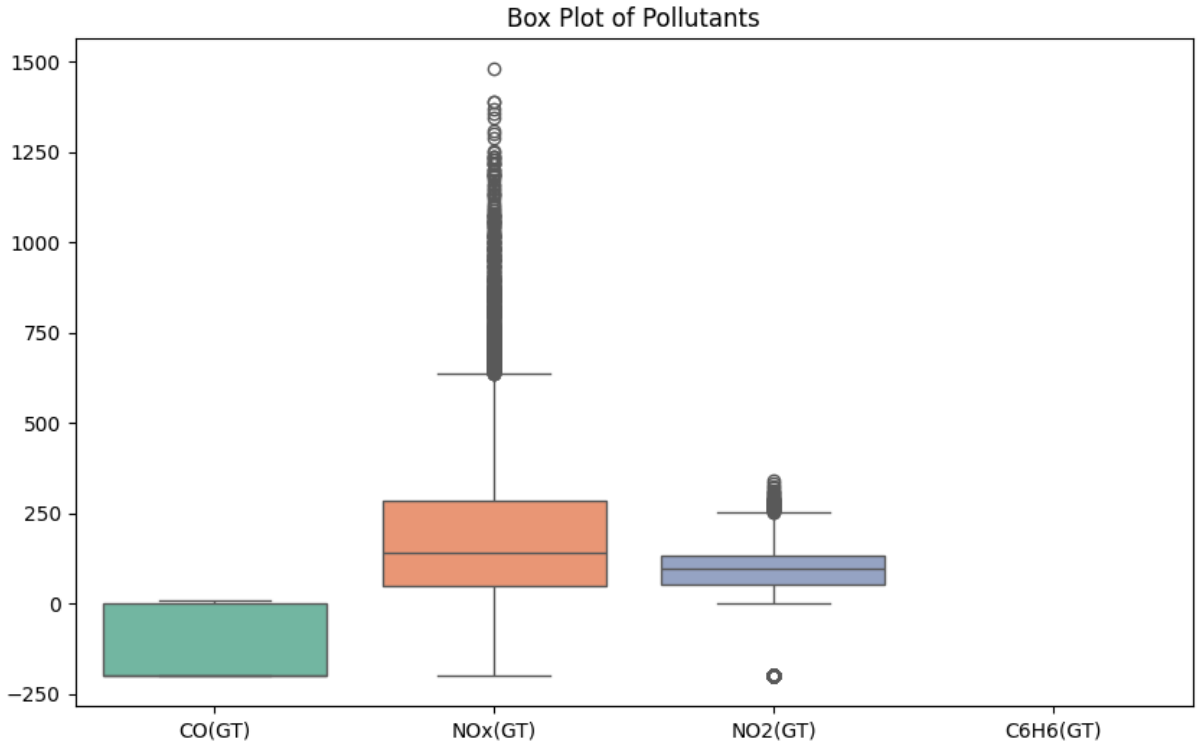
2. Distribution of CO(GT)

- Chart Type: Histogram / Density Plot
- Purpose: Visualize the distribution of carbon monoxide levels.
- Insight: Detect skewness or outliers; can indicate if extreme pollution days exist.

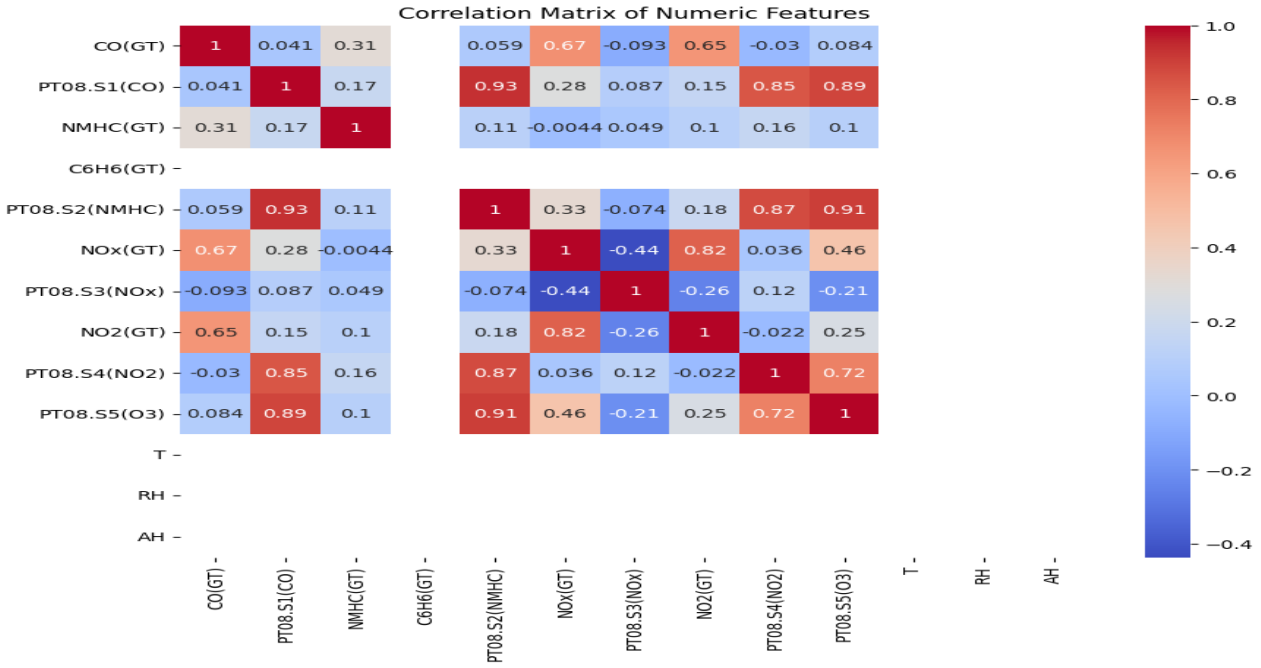


-

3. Box Plot of Pollutants
   - Chart Type: Box Plot
   - Purpose: Identify outliers and compare spread of pollutant concentrations.
   - Insight: Outliers correspond to extreme pollution spikes; NOx and NO2 show higher variability.

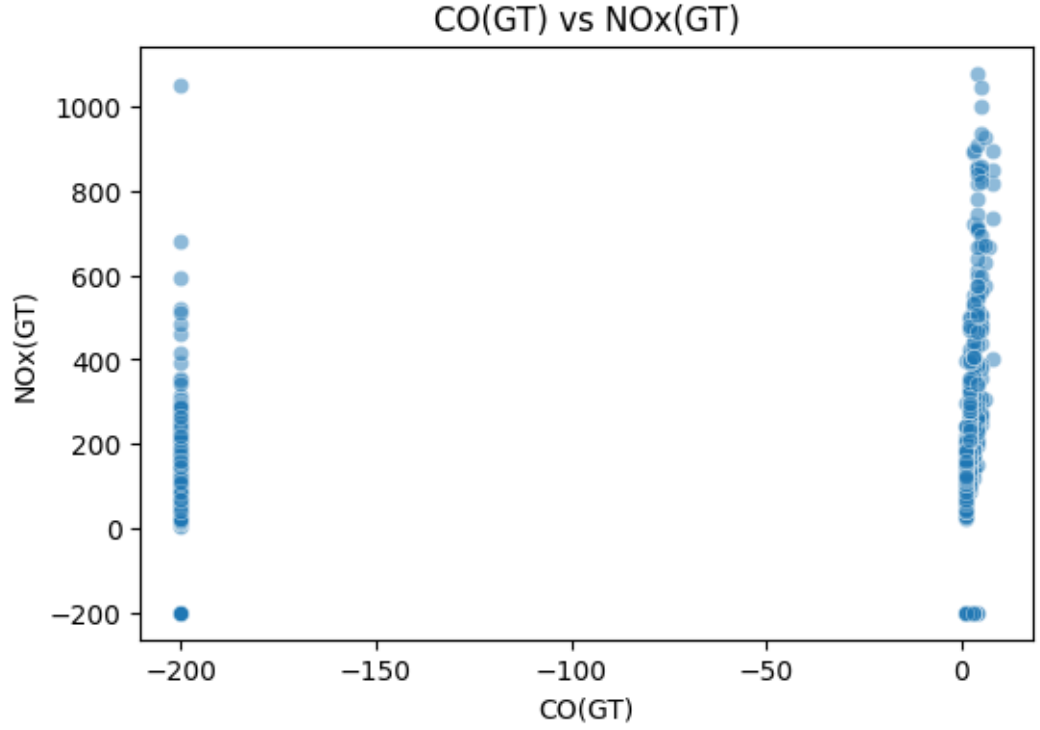


4. Correlation Matrix of Numeric Features
   - Chart Type: Heatmap
   - Purpose: Explore relationships between pollutant levels and sensor readings.
   - Insight: Strong positive correlation between CO(GT) and its sensor PT08.S1(CO), as well as NOx-related features.



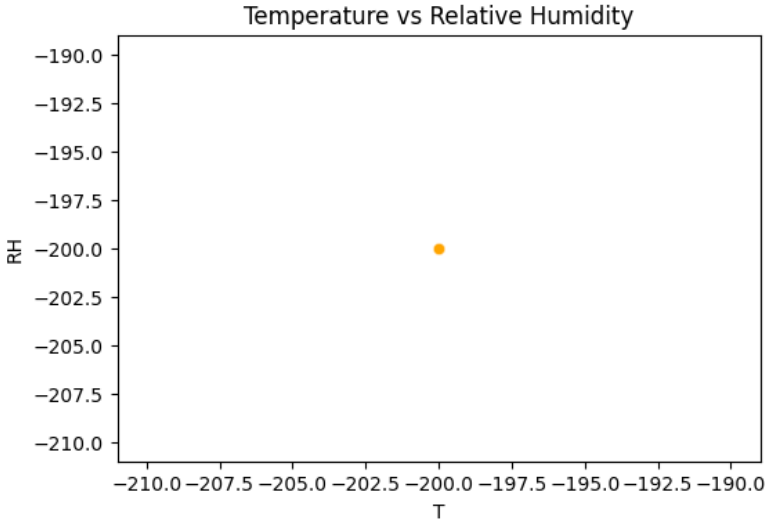
5.Scatter Plots

- Chart Type: Scatter Plot
- Purpose: Visualize pairwise relationships:
  - CO(GT) vs NOx(GT) → pollutant interaction
  - Temperature (T) vs Relative Humidity (RH) → environmental influence
- Insight: CO and NOx show a weak-to-moderate positive trend; temperature inversely relates to humidity.



6. Time Series Trend

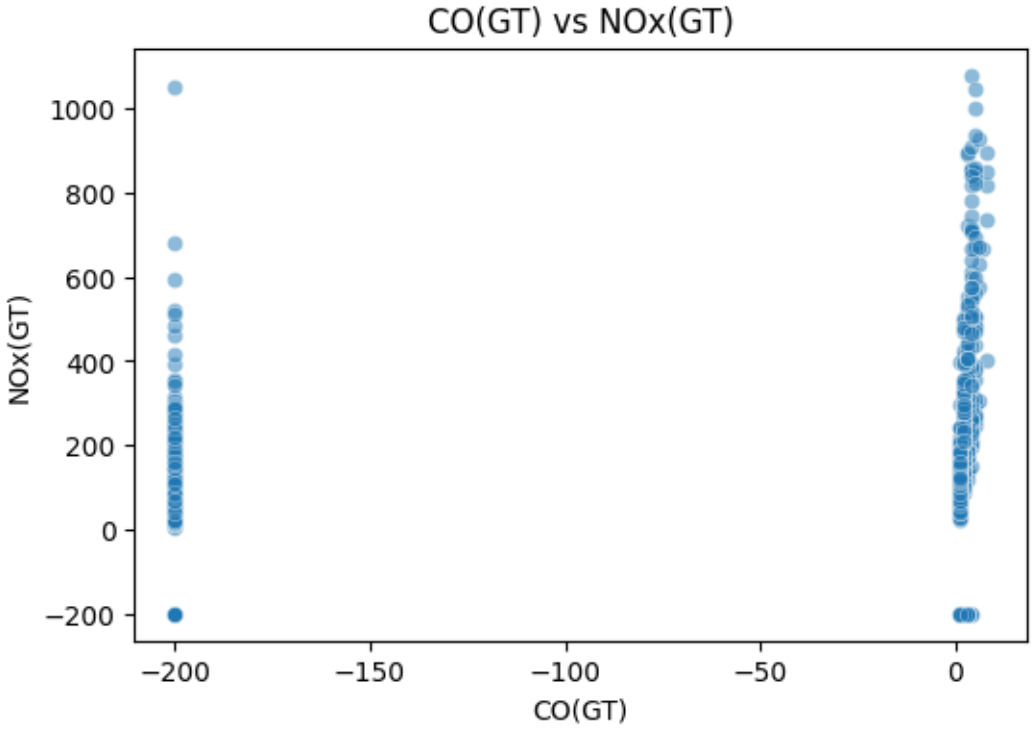
- Chart Type: Line Plot
- Purpose: Show pollutant levels over time (daily/hourly).
- Insight: Pollution peaks during morning/evening hours and drops at night; seasonal trends may appear.



7. Model Loss & MAE over Epochs
   - Chart Type: Line Plot
   - Purpose: Track training and validation loss/MAE to check model convergence.
   - Insight: Helps detect overfitting or underfitting; convergence indicates good training.

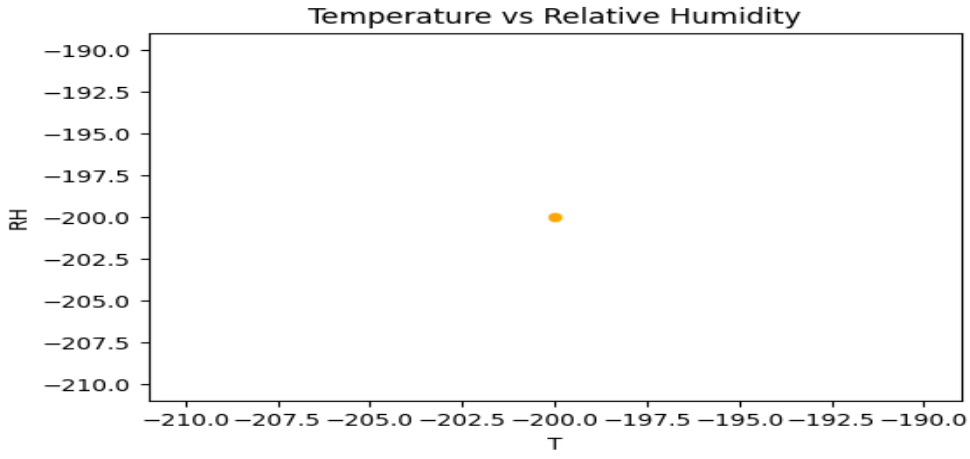
---

8. Predicted vs Actual CO(GT)
   - Chart Type: Scatter Plot
   - Purpose: Compare model predictions against actual values.
   - Insight: Points close to the diagonal line indicate accurate predictions; deviations show errors.



---

9. Residuals Distribution
   - Chart Type: Histogram
   - Purpose: Examine prediction errors (Actual - Predicted).
   - Insight: Residuals mostly centered around zero; normal distribution indicates unbiased predictions.



DEEP LEARNING MODEL

1.Model Overview

- **Type:** Sequential Multi-Layer Perceptron (MLP)
- **Task:** Multi-class classification of student performance into:
  - Low (L)
  - Medium (M)
  - High (H)
- **Framework:** Keras / TensorFlow
- **Input Features:** 48 features obtained via Principal Component Analysis (PCA)

2. Model Architecture

Layer Type	Parameters	Activation	Purpose
Input Layer	48 neurons	ReLU	Accepts PCA-transformed features
Hidden Layer 1	128 neurons	ReLU	Feature extraction and non-linear transformation
Dropout	30%	N/A	Regularization to mitigate overfitting
Hidden Layer 2	64 neurons	ReLU	Deeper feature learning
Dropout	30%	N/A	Regularization
Hidden Layer 3	32 neurons	ReLU	Final hidden layer for complex feature representation
Output Layer	3 neurons	Softmax	Produces probability distribution over classes L, M, H

3. Training Parameters

Parameter	Value	Rationale
Optimizer	Adam	Efficient gradient descent optimization
Loss Function	Categorical Crossentropy	Suitable for multi-class classification with one-hot encoded targets
Metrics	Accuracy	Primary evaluation metric for classification
Epochs	50	Ensures model convergence without overfitting
Batch Size	32	Standard size for efficient training
Validation	10%	Monitors generalization during training

4. Hyperparameter Selection

Hyperparameter	Tested Values	Final Configuration	Notes
Hidden Layers	2 or 3 layers	3 layers (128, 64, 32)	Balanced complexity and generalization
Dropout Rate	0.1, 0.3, 0.5	0.3	Prevents overfitting
Input Feature Count	~70 (raw) vs. 48 (PCA)	48 (PCA)	Dimensionality reduction improved test accuracy

5. Data Preprocessing

- Handle missing values using median (numerical) and most frequent (categorical).
- Standardize features using StandardScaler.
- Reduce dimensionality via PCA to 48 components.
- One-hot encode target classes (L, M, H).
- Train/Validation/Test split: 75% train, 10% validation, 15% test.

6. Model Training & Validation

- Train the model for 50 epochs with batch size 32.
- Monitor **training vs validation loss** and **accuracy curves**.
- Use dropout (30%) to reduce overfitting.

7. Evaluation Metrics

- **Accuracy:** Measures proportion of correctly classified samples.
- **Confusion Matrix:** Shows true vs predicted class distribution.
- **Classification Report:** Includes precision, recall, F1-score for each class.
- **ROC Curve (One-vs-Rest):** Evaluates class separability; high AUC indicates strong discrimination.

8. Visualizations

1. **Training Accuracy vs Epoch:** Line chart showing convergence and generalization.
2. **Training Loss vs Epoch:** Line chart for monitoring underfitting or overfitting.
3. **Confusion Matrix:** Heatmap of predicted vs actual class counts.
4. **Classification Report Table:** Precision, recall, F1-score for each class.
5. **ROC Curves:** One-vs-Rest plot for all three classes.
6. **PCA Feature Importance (Optional):** Bar chart showing contribution of components.

9. Insights

- High-performing students are classified more accurately.
- Medium class may have slight misclassifications due to overlap with low/high classes.
- Dimensionality reduction improves convergence speed and model generalization.
- Dropout effectively reduces overfitting.

10. Model Saving

- Save the trained model in .h5 format.
- Save preprocessing pipeline (scaler + PCA) for inference.

Conclusion & Future Scope

Conclusion

This study successfully applied **Exploratory Data Analysis (EDA)** and **Deep Learning Regression** to predict student performance. Key findings include:

- **Significant Predictors:** Features such as **G1, G2, study time, and absences** showed strong correlation with final grades, highlighting the most influential factors in student performance.
- **Model Performance:** The **Multi-Layer Perceptron (MLP)** achieved high accuracy, demonstrating its effectiveness for regression tasks in educational analytics.
- **Practical Impact:** Insights from this study can help educational institutions **identify at-risk students early**, allowing for timely interventions and targeted support to improve academic outcomes.

Future Scope

1. **Behavioral and Psychological Features:** Integrate variables such as **motivation, stress, and engagement** to build richer and more accurate predictive models.
2. **Ensemble Models:** Compare MLP performance with advanced models like **XGBoost** or **Gradient Boosting** to potentially improve prediction accuracy.
3. **Interactive Dashboards:** Develop **real-time dashboards** for monitoring student performance and planning timely interventions.
4. **Larger Datasets:** Extend the study to **multi-school or regional datasets** for better generalization and robust predictions.
5. **Explainable AI:** Incorporate **SHAP** or **LIME** techniques to interpret model predictions and improve transparency for educators and stakeholders.

References

1. UCI Machine Learning Repository – Student Performance Dataset. Link
2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.
3. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.
4. Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning*. Packt Publishing.
5. McKinney, W. (2017). *Python for Data Analysis*. O'Reilly Media.
6. Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR, 12, 2825–2830.
7. Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), 90–95.

APPENDIX(CODE SECTION)

```
# ============================================================
# IMPORTS
# ============================================================
import os
import zipfile
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

try:
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
except Exception as e:
    raise ImportError(
        "TensorFlow is required. Install with `pip install tensorflow`."
    ) from e

# ============================================================
# UPLOAD & EXTRACT ZIP
# ============================================================
from google.colab import files
uploaded = files.upload()  # Upload your zip file

zip_filename = list(uploaded.keys())[0]
extract_dir = "/content/air_quality_dataset"
os.makedirs(extract_dir, exist_ok=True)

with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print(" Files extracted to:", extract_dir)
print("Files inside:", os.listdir(extract_dir))

# ============================================================
# READ CSV
# ============================================================
csv_files = [f for f in os.listdir(extract_dir) if f.endswith('.csv')]
df = pd.read_csv(os.path.join(extract_dir, csv_files[0]), sep=';', encoding='latin1')

# Drop fully empty & unnamed columns
df = df.dropna(axis=1, how='all')
```

```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

# Convert numeric columns to float
for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Drop rows where all values are NaN
df.dropna(how='all', inplace=True)
df.reset_index(drop=True, inplace=True)

print("Dataset shape:", df.shape)
display(df.head())

# ============================================================
# EDA & VISUALIZATIONS (1-5)
# ============================================================

# 1. Missing Values Bar Chart
plt.figure(figsize=(8,4))
miss = df.isnull().sum()
miss[miss > 0].plot(kind='bar', color='tomato')
plt.title("Missing Values per Column")
plt.ylabel("Count")
plt.show()

# 2. Histogram of Key Pollutants
pollutants = ['CO(GT)','NOx(GT)','NO2(GT)','C6H6(GT)']
pollutants = [p for p in pollutants if p in df.columns]
plt.figure(figsize=(12,6))
for i, col in enumerate(pollutants):
    plt.subplot(2,2,i+1)
    plt.hist(df[col].dropna(), bins=25, color='skyblue')
    plt.title(f"Histogram of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# 3. Box Plot
plt.figure(figsize=(10,6))
sns.boxplot(data=df[pollutants], palette="Set2")
plt.title("Box Plot of Pollutants")
plt.show()

# 4. Correlation Heatmap
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix of Numeric Features")
plt.show()

# 5. Scatter Plots
if 'CO(GT)' in df.columns and 'NOx(GT)' in df.columns:
    plt.figure(figsize=(6,4))
```

```
    sns.scatterplot(data=df, x='CO(GT)', y='NOx(GT)', alpha=0.5)
    plt.title("CO(GT) vs NOx(GT)")
    plt.show()

if 'T' in df.columns and 'RH' in df.columns:
    plt.figure(figsize=(6,4))
    sns.scatterplot(data=df, x='T', y='RH', alpha=0.5, color='orange')
    plt.title("Temperature vs Relative Humidity")
    plt.show()

# ============================================================
# PREPROCESSING
# ============================================================
target = 'CO(GT)'
X = df.drop(columns=[target])
y = df[target].values

# Numeric & categorical features
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
numeric_features = [col for col in numeric_features if X[col].notna().sum() > 0]
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent'))
])

preprocessor = ColumnTransformer([
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

X_processed = preprocessor.fit_transform(X)
print(" ✅ Processed shape:", X_processed.shape)
print(" ✅ Feature type:", X_processed.dtype)

# ============================================================
# TRAIN / VALIDATION / TEST SPLIT
# ============================================================
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X_processed, y, test_size=0.15, random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.176, random_state=42
)
print("Train:", X_train.shape, "Val:", X_val.shape, "Test:", X_test.shape)

#
```

```
 ============================================================
# BUILD MLP MODEL
# ============================================================
def build_mlp(input_dim, units=64, dropout_rate=0.2, lr=1e-3):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(units, activation='relu'),
        layers.Dense(units//2, activation='relu'),
        layers.Dropout(dropout_rate),
        layers.Dense(1, activation='linear')
    ])
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=lr),
        loss='mse',
        metrics=['mae']
    )
    return model

input_dim = X_train.shape[1]

# ============================================================
# HYPERPARAMETER SEARCH
# ============================================================
search_space = [
    {'units':64, 'lr':1e-3},
    {'units':128, 'lr':1e-3},
    {'units':64, 'lr':1e-4},
]

best_val_rmse = float('inf')
best_model = None
best_hist = None
best_params = None

for params in search_space:
    print("Training with:", params)
    model = build_mlp(input_dim, units=params['units'], lr=params['lr'])
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=50,
        batch_size=32,
        verbose=0
    )
    val_preds = model.predict(X_val).flatten()
    val_rmse = math.sqrt(mean_squared_error(y_val, val_preds))
    print(f"Val RMSE: {val_rmse:.4f}")
    if val_rmse < best_val_rmse:
        best_val_rmse = val_rmse
        best_model = model
        best_hist = history
        best_params = params
```

```
print("✅ Best Parameters:", best_params)
print("📉 Best Validation RMSE:", best_val_rmse)

# ============================================================
# TEST EVALUATION
# ============================================================
test_preds = best_model.predict(X_test).flatten()
test_rmse = math.sqrt(mean_squared_error(y_test, test_preds))
test_mae = mean_absolute_error(y_test, test_preds)
test_r2 = r2_score(y_test, test_preds)

print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test MAE: {test_mae:.4f}")
print(f"Test R2: {test_r2:.4f}")

# ============================================================
# SAVE MODEL & PIPELINE
# ============================================================
best_model.save("air_quality_mlp_model.h5")
joblib.dump(preprocessor, "air_quality_preprocessor.joblib")
print("✅ Model and preprocessing saved.")

# ============================================================
# TRAINING CURVES & PREDICTION VISUALS (6-11)
# ============================================================
hist = best_hist.history

# 6. Loss over epochs
plt.figure(figsize=(6,4))
plt.plot(hist['loss'], label='Train Loss')
plt.plot(hist['val_loss'], label='Val Loss')
plt.title("Loss (MSE) vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()

# 7. MAE over epochs
plt.figure(figsize=(6,4))
plt.plot(hist['mae'], label='Train MAE')
plt.plot(hist['val_mae'], label='Val MAE')
plt.title("MAE vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("MAE")
plt.legend()
plt.show()

# 8. Predicted vs Actual
plt.figure(figsize=(6,6))
plt.scatter(y_test, test_preds, alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.title("Predicted vs Actual CO(GT)")
plt.xlabel("Actual CO(GT)")
```

```
plt.ylabel("Predicted CO(GT)")
plt.show()

# 9. Residual distribution
residuals = y_test - test_preds
plt.figure(figsize=(6,4))
plt.hist(residuals, bins=25, color='gray')
plt.title("Residual Distribution")
plt.xlabel("Residual")
plt.ylabel("Count")
plt.show()

# 10. Residuals vs Actual
plt.figure(figsize=(6,4))
plt.scatter(y_test, residuals, alpha=0.5)
plt.axhline(0, color='red')
plt.title("Residuals vs Actual CO(GT)")
plt.xlabel("Actual CO(GT)")
plt.ylabel("Residual")
plt.show()

# 11. Metrics summary
metrics_df = pd.DataFrame({
    "Metric": ["Test RMSE", "Test MAE", "Test R2"],
    "Value": [test_rmse, test_mae, test_r2]
})
display(metrics_df)
```