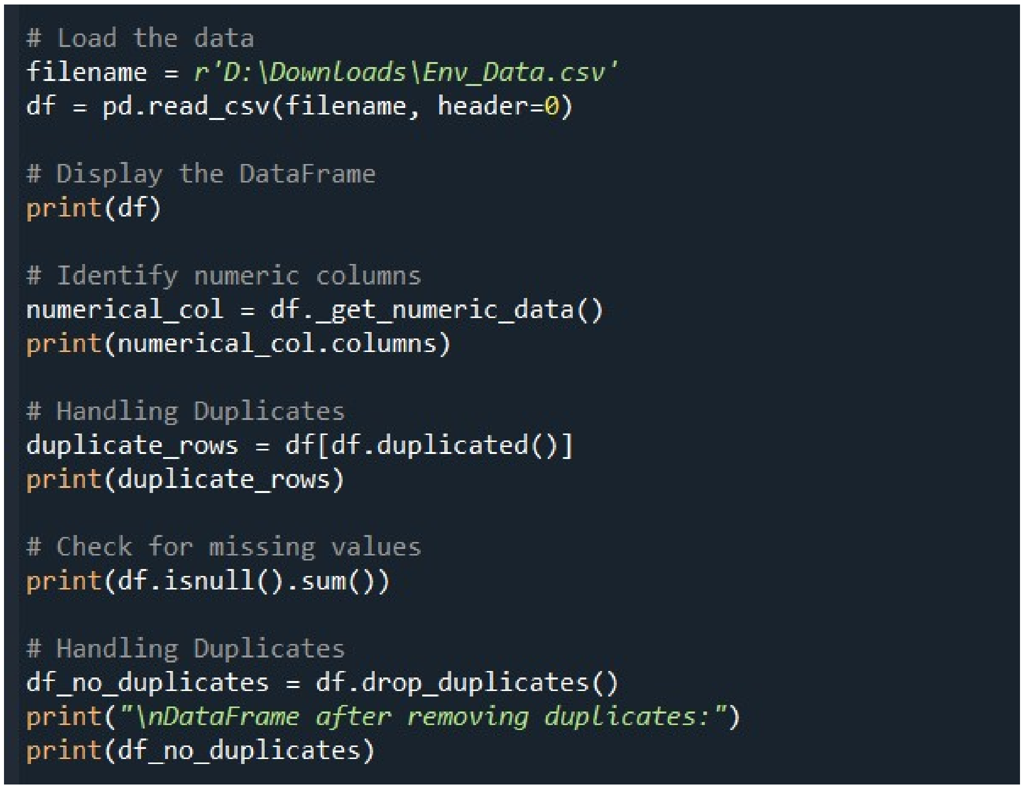
1.0 Introduction

This report will be discussing on developing a predictive system through programming for an Air Quality Index (AQI) based on the dataset of “Env_Data” provided. The developed system is designed to predict the AQI based on the dataset.

2.0 This section will talk about cleaning the dataset.

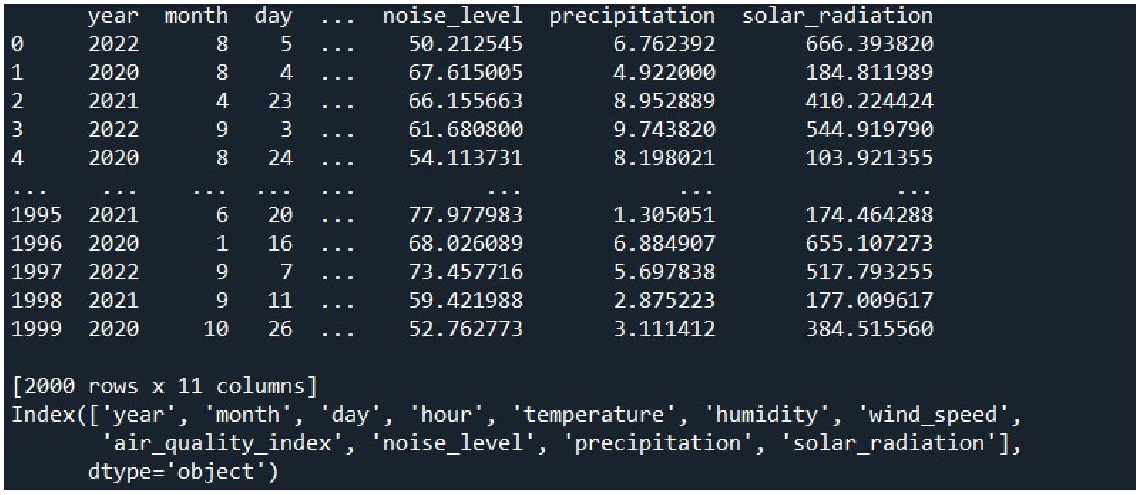
2.1 Identifying numeric columns, handling duplicates, and checking for missing values will be conducted at this part of the report.



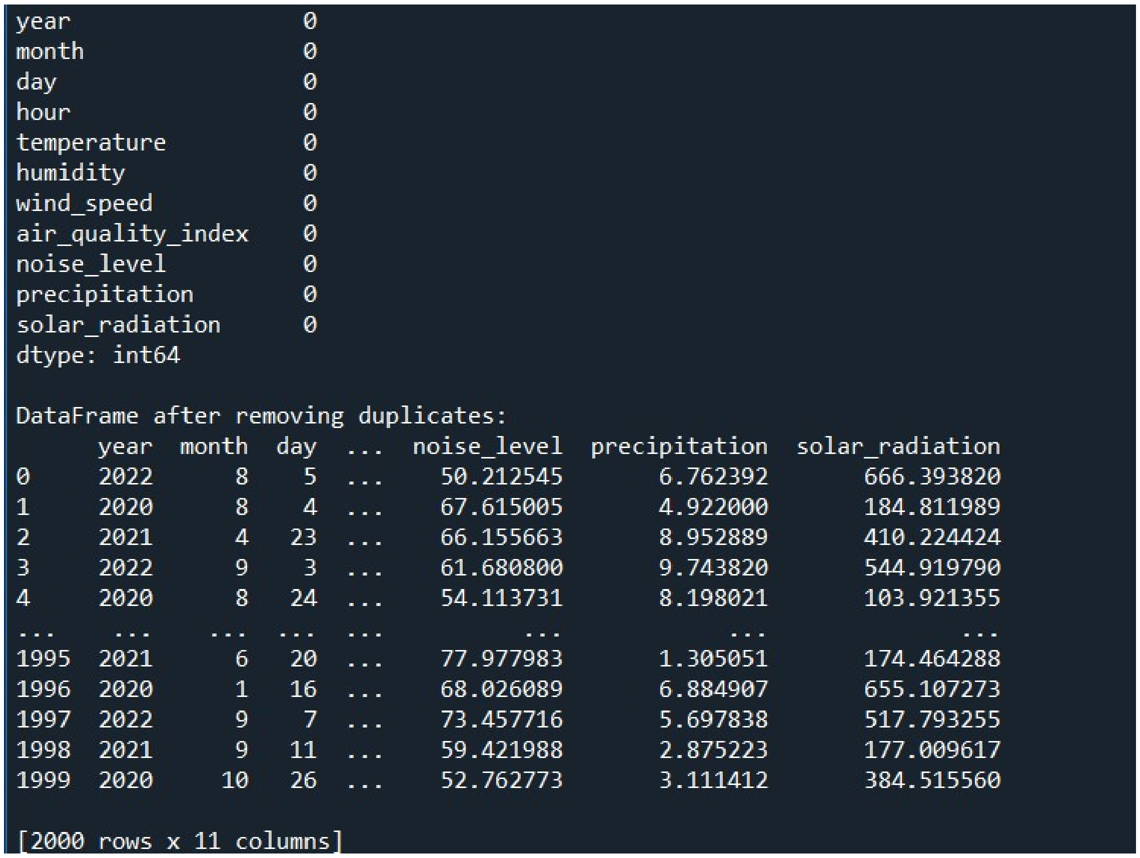
```
# Load the data
filename = r'D:\Downloads\Env_Data.csv'
df = pd.read_csv(filename, header=0)

# Display the DataFrame
print(df)

# Identify numeric columns
numerical_col = df._get_numeric_data()
print(numerical_col.columns)

# Handling Duplicates
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)

# Check for missing values
print(df.isnull().sum())

# Handling Duplicates
df_no_duplicates = df.drop_duplicates()
print("\nDataFrame after removing duplicates:")
print(df_no_duplicates)
```

The program begins with loading the dataset ‘Env_Data.csv’ into pandas dataframe, the structure is displayed using print(df) to show the dataset. ‘_get_numeric_data()’ method is used to identify the numeric columns and display. Duplicates are checked and removed in the dataset and missing values are identified before printing.

The image here displays the output of the codes.



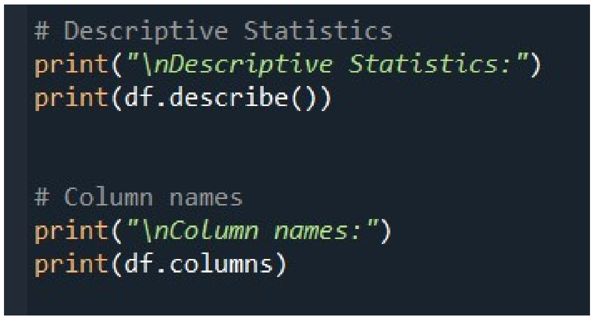
```
      year  month  day  ...  noise_level  precipitation  solar_radiation
0     2022      8    5  ...    50.212545       6.762392       666.393820
1     2020      8    4  ...    67.615005       4.922000       184.811989
2     2021      4   23  ...    66.155663       8.952889       410.224424
3     2022      9    3  ...    61.680800       9.743820       544.919790
4     2020      8   24  ...    54.113731       8.198021       103.921355
...    ...    ...  ...  ...          ...            ...              ...
1995  2021      6   20  ...    77.977983       1.305051       174.464288
1996  2020      1   16  ...    68.026089       6.884907       655.107273
1997  2022      9    7  ...    73.457716       5.697838       517.793255
1998  2021      9   11  ...    59.421988       2.875223       177.009617
1999  2020     10   26  ...    52.762773       3.111412       384.515560

[2000 rows x 11 columns]
Index(['year', 'month', 'day', 'hour', 'temperature', 'humidity', 'wind_speed',
       'air_quality_index', 'noise_level', 'precipitation', 'solar_radiation'],
      dtype='object')
```



```
year                 0
month                0
day                  0
hour                 0
temperature          0
humidity             0
wind_speed           0
air_quality_index    0
noise_level          0
precipitation        0
solar_radiation      0
dtype: int64

DataFrame after removing duplicates:
      year  month  day  ...  noise_level  precipitation  solar_radiation
0     2022      8    5  ...    50.212545       6.762392       666.393820
1     2020      8    4  ...    67.615005       4.922000       184.811989
2     2021      4   23  ...    66.155663       8.952889       410.224424
3     2022      9    3  ...    61.680800       9.743820       544.919790
4     2020      8   24  ...    54.113731       8.198021       103.921355
...    ...    ...  ...  ...          ...            ...              ...
1995  2021      6   20  ...    77.977983       1.305051       174.464288
1996  2020      1   16  ...    68.026089       6.884907       655.107273
1997  2022      9    7  ...    73.457716       5.697838       517.793255
1998  2021      9   11  ...    59.421988       2.875223       177.009617
1999  2020     10   26  ...    52.762773       3.111412       384.515560

[2000 rows x 11 columns]
```

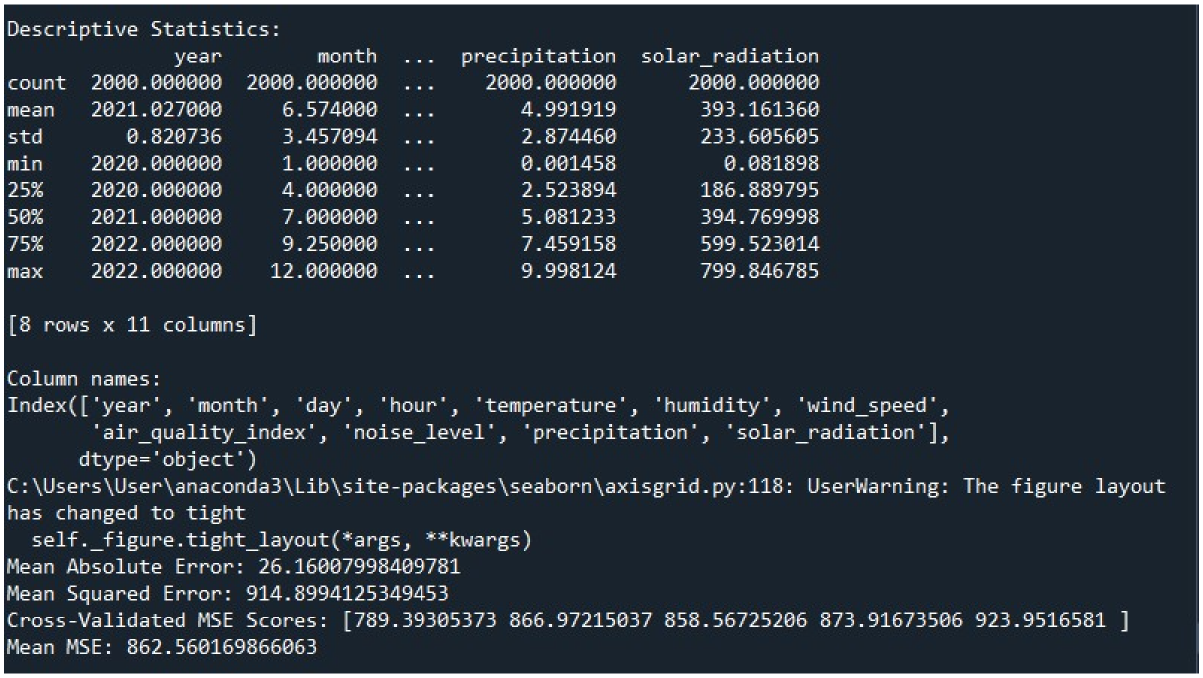
2.2 This part of the report will be talking about descriptive statistics and understanding of column names.

This part of the code talks about descriptive statistics based on dataframe df. It measures the mean, standard deviation, minimum, 25%, 50% and 75% of each numeric column from the dataframe. Whereas for column names, it is implemented to inspect and display the nanames of the columns which are in the dataset.



```
# Descriptive Statistics
print("\nDescriptive Statistics:")
print(df.describe())


# Column names
print("\nColumn names:")
print(df.columns)
```

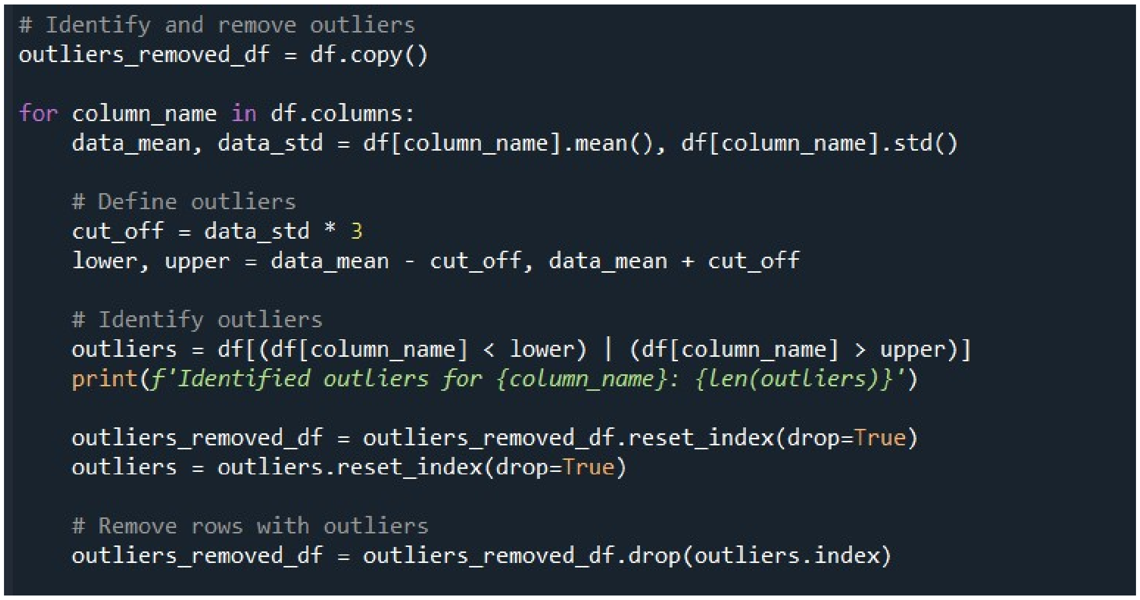
This image shows the results of the descriptive statistics and column names.



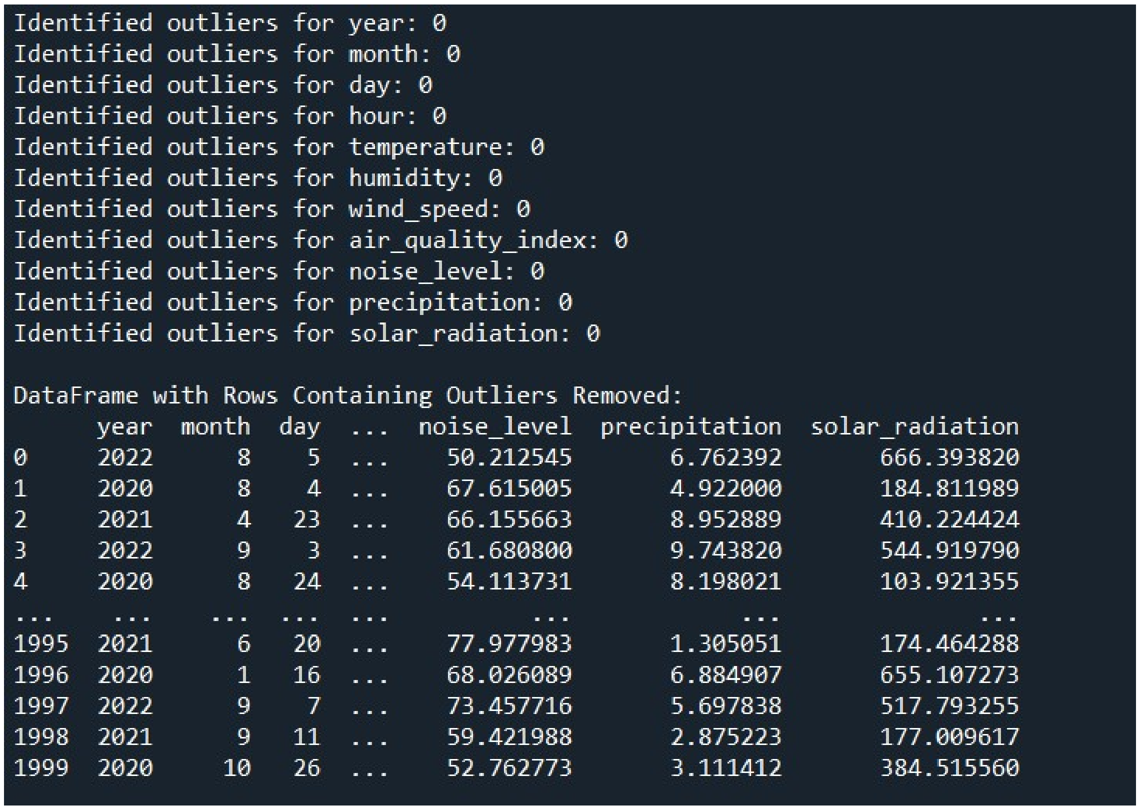
```
Descriptive Statistics:
              year        month  ...  precipitation  solar_radiation
count  2000.000000  2000.000000  ...    2000.000000      2000.000000
mean   2021.027000     6.574000  ...       4.991919       393.161360
std       0.820736     3.457094  ...       2.874460       233.605605
min    2020.000000     1.000000  ...       0.001458         0.081898
25%    2020.000000     4.000000  ...       2.523894       186.889795
50%    2021.000000     7.000000  ...       5.081233       394.769998
75%    2022.000000     9.250000  ...       7.459158       599.523014
max    2022.000000    12.000000  ...       9.998124       799.846785

[8 rows x 11 columns]

Column names:
Index(['year', 'month', 'day', 'hour', 'temperature', 'humidity', 'wind_speed',
       'air_quality_index', 'noise_level', 'precipitation', 'solar_radiation'],
      dtype='object')
C:\Users\User\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout
has changed to tight
  self._figure.tight_layout(*args, **kwargs)
Mean Absolute Error: 26.16007998409781
Mean Squared Error: 914.8994125349453
Cross-Validated MSE Scores: [789.39305373 866.97215037 858.56725206 873.91673506 923.9516581 ]
Mean MSE: 862.560169866063
```

2.3 Handling outliers will be conducted in this section.

The next step in cleaning the dataset will be to handle outliers, the outliers are removed based on 3-standard deviation thresholds and displayed.



```python
# Identify and remove outliers
outliers_removed_df = df.copy()

for column_name in df.columns:
    data_mean, data_std = df[column_name].mean(), df[column_name].std()

    # Define outliers
    cut_off = data_std * 3
    lower, upper = data_mean - cut_off, data_mean + cut_off

    # Identify outliers
    outliers = df[(df[column_name] < lower) | (df[column_name] > upper)]
    print(f'Identified outliers for {column_name}: {len(outliers)}')

    outliers_removed_df = outliers_removed_df.reset_index(drop=True)
    outliers = outliers.reset_index(drop=True)

    # Remove rows with outliers
    outliers_removed_df = outliers_removed_df.drop(outliers.index)
```

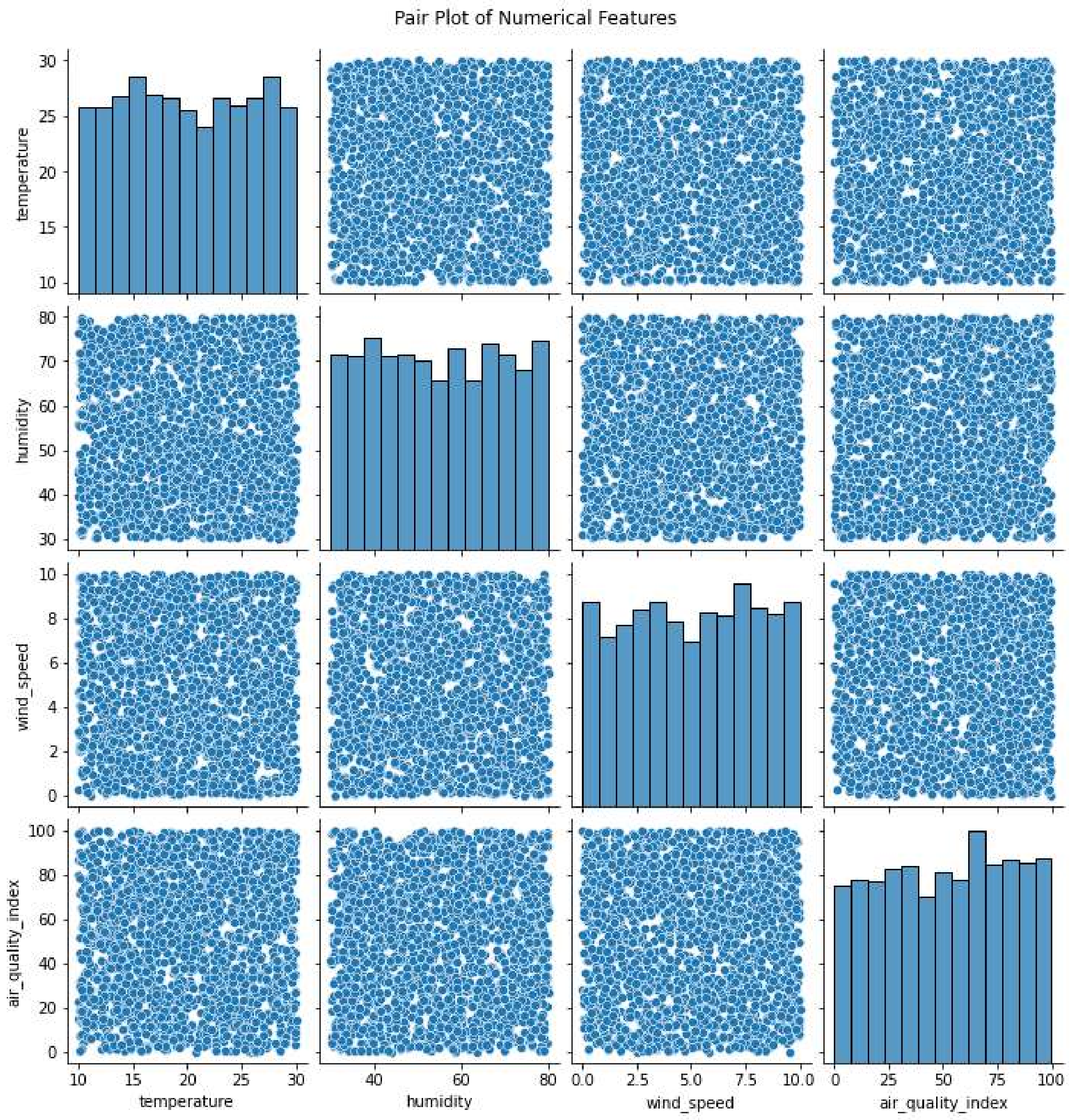


```
Identified outliers for year: 0
Identified outliers for month: 0
Identified outliers for day: 0
Identified outliers for hour: 0
Identified outliers for temperature: 0
Identified outliers for humidity: 0
Identified outliers for wind_speed: 0
Identified outliers for air_quality_index: 0
Identified outliers for noise_level: 0
Identified outliers for precipitation: 0
Identified outliers for solar_radiation: 0

DataFrame with Rows Containing Outliers Removed:
      year  month  day  ...  noise_level  precipitation  solar_radiation
0     2022      8    5  ...    50.212545       6.762392       666.393820
1     2020      8    4  ...    67.615005       4.922000       184.811989
2     2021      4   23  ...    66.155663       8.952889       410.224424
3     2022      9    3  ...    61.680800       9.743820       544.919790
4     2020      8   24  ...    54.113731       8.198021       103.921355
...    ...    ...  ...  ...          ...            ...              ...
1995  2021      6   20  ...    77.977983       1.305051       174.464288
1996  2020      1   16  ...    68.026089       6.884907       655.107273
1997  2022      9    7  ...    73.457716       5.697838       517.793255
1998  2021      9   11  ...    59.421988       2.875223       177.009617
1999  2020     10   26  ...    52.762773       3.111412       384.515560
```

3.0 This section will be discussing on data distribution through visualisation.

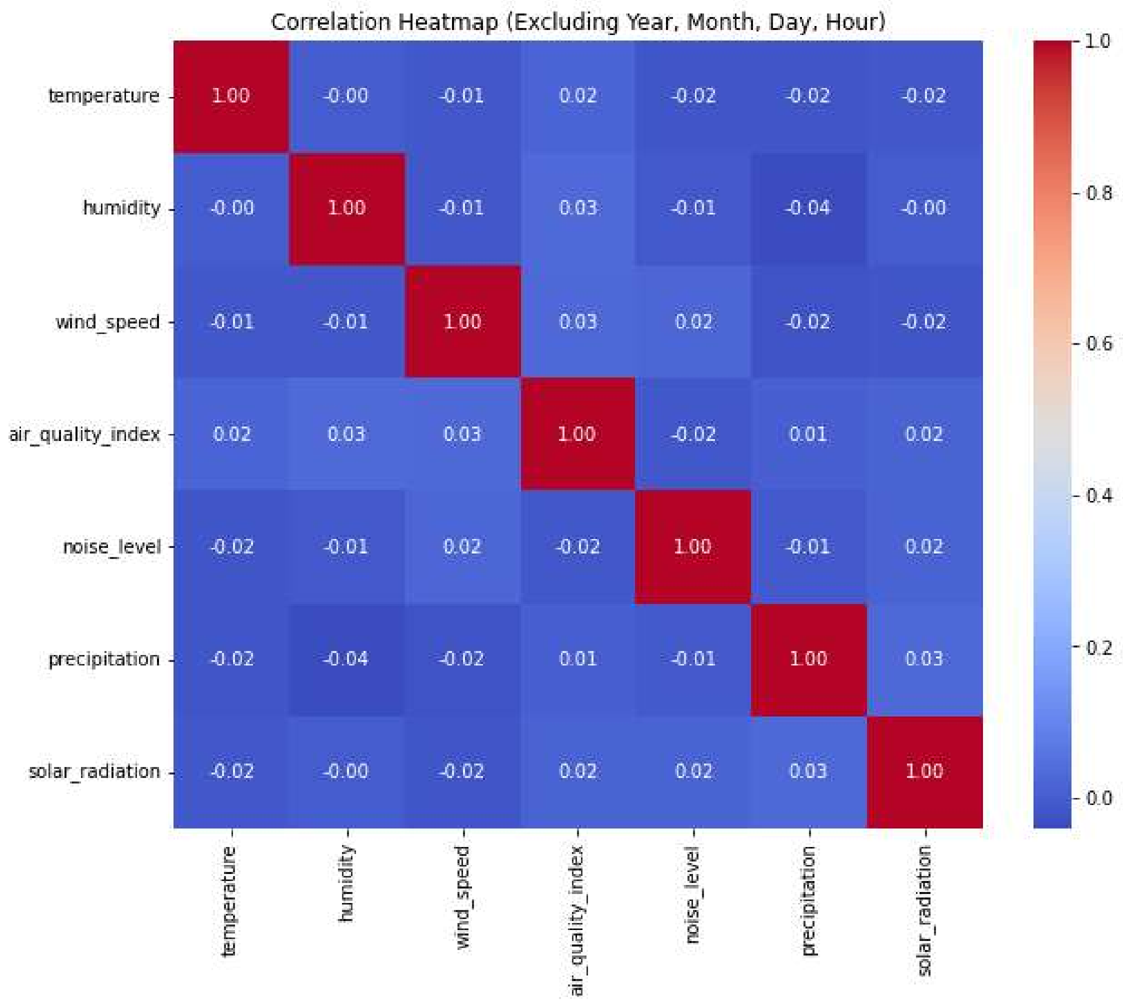
3.1 Visualise Pairplot

This method uses Seaborn to create ‘pairplot’ function to display the scatterplot of numerical features of 'temperature', 'humidity', 'wind_speed', 'air_quality_index' against each other in the features. However, the results do not show much relationship between each feature.



3.2 Correlation Heatmap

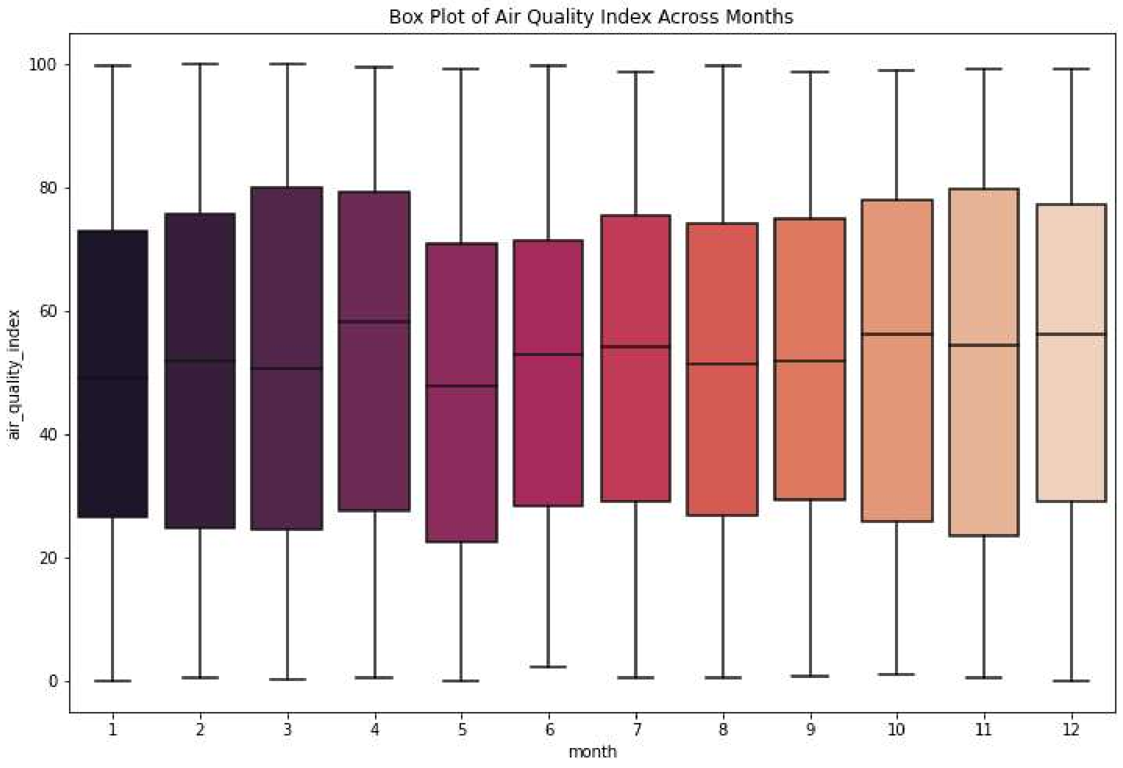
This method is executed through Seaborn's heatmap function to perform heatmap visualisation towards numerical values to understand the relationship with correlation matrix. The stronger the colour intensity, the higher the relationship of the values. However, there is no relationship between each other as all the colours are blue, in this case means that the intensity is low.



3.3 Box Plot

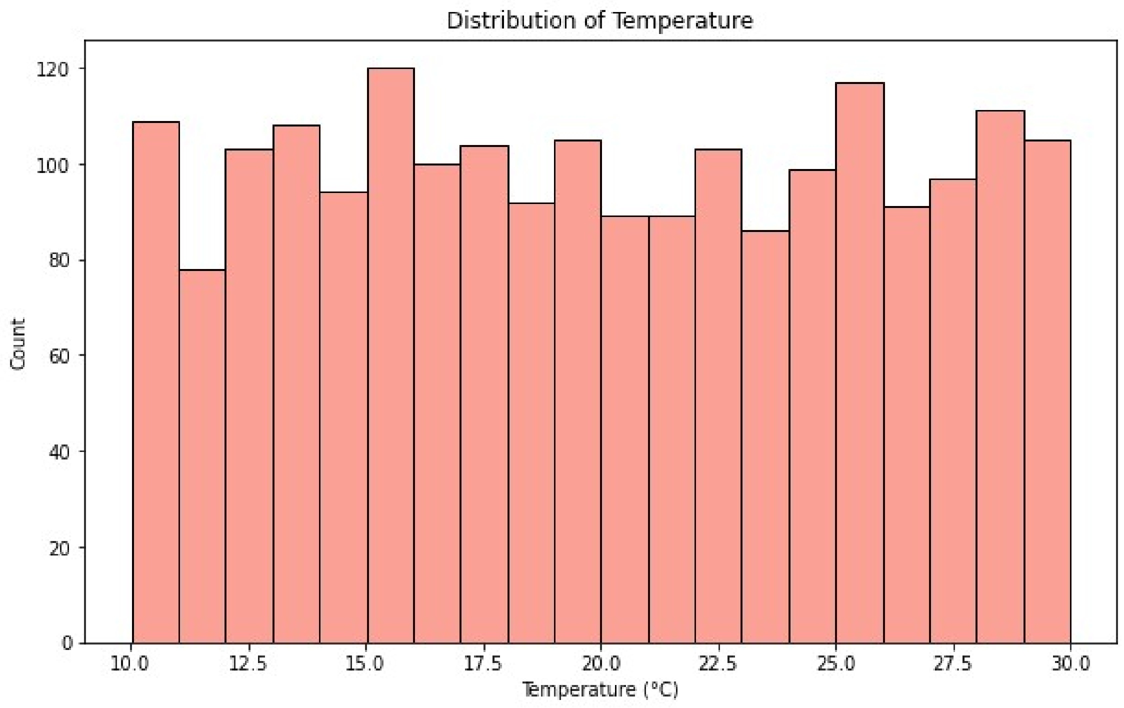
Box Plot is used here to display the distribution of air quality index through different months. It can be observed that the central line which represents the median of each box is around 50, this indicates a balanced distribution for the air quality index values.

.

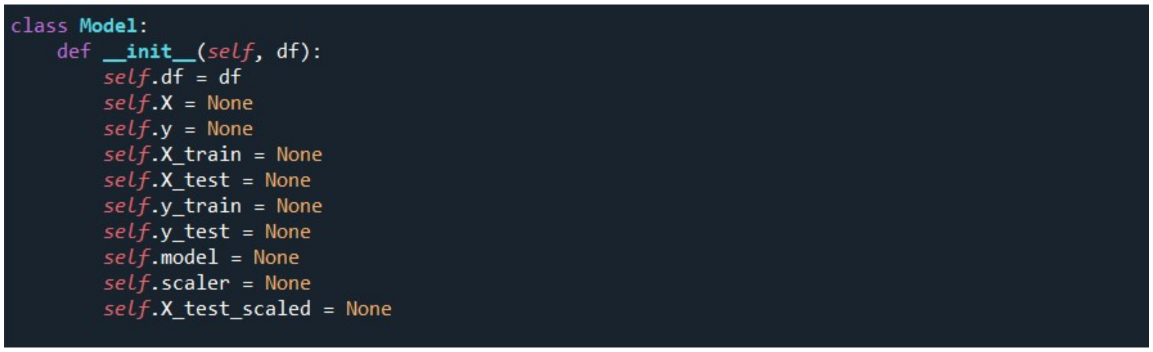


3.4 Histogram

Histogram method is used to visualise the distribution of the temperature variable. From the graph, the frequency of each temperature variable is quite consistent and the mean for the distribution is between 80-90. The spread is concentrated at about 80 to 100. It can be said that the count of the temperature in this histogram is relatively consistent.

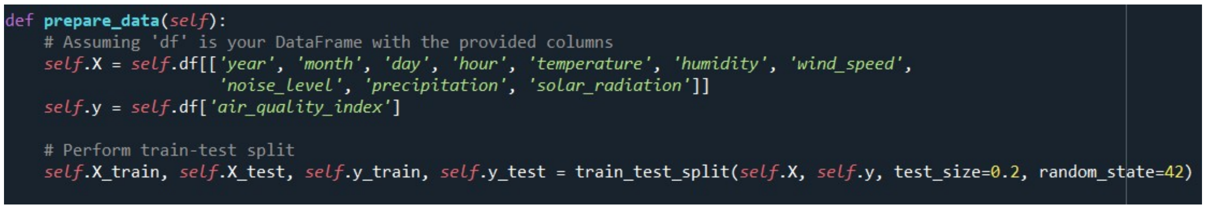


4.0 This section will be discussing about modelling.



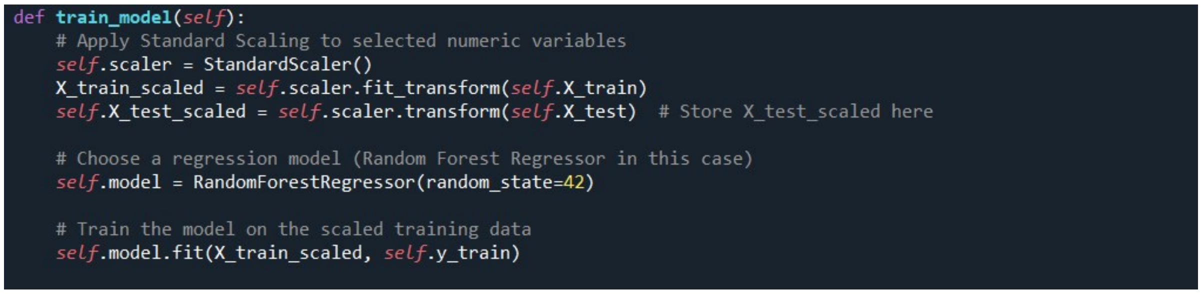
```python
class Model:
    def __init__(self, df):
        self.df = df
        self.X = None
        self.y = None
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None
        self.model = None
        self.scaler = None
        self.X_test_scaled = None
```

This part will be talking about initialisation. The method applied ‘__init__’ for the model is used to initialise the attributes that will be used at the following steps. ‘df’ is the dataframe for the dataset.



```python
def prepare_data(self):
    # Assuming 'df' is your DataFrame with the provided columns
    self.X = self.df[['year', 'month', 'day', 'hour', 'temperature', 'humidity', 'wind_speed',
                      'noise_level', 'precipitation', 'solar_radiation']]
    self.y = self.df['air_quality_index']

    # Perform train-test split
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=0.2, random_state=42)
```

In ‘prepare_data’, this method will separate the dataset into X and Y. The target variable will be Y and the features will be X based on the columns on the dataframe. ‘train_test_split’ from scikit-learn is applied to perform train-test split, this allows the creation of training and testing for both the feature and target variables.



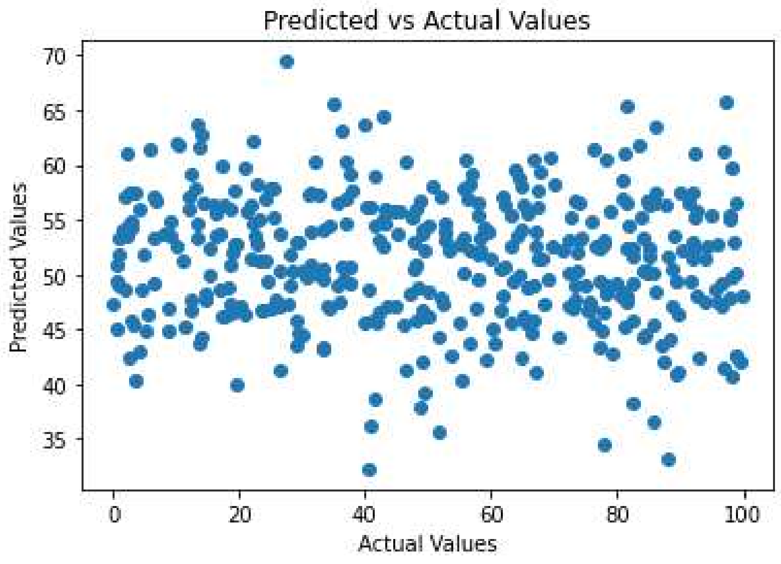
```python
def train_model(self):
    # Apply Standard Scaling to selected numeric variables
    self.scaler = StandardScaler()
    X_train_scaled = self.scaler.fit_transform(self.X_train)
    self.X_test_scaled = self.scaler.transform(self.X_test)  # Store X_test_scaled here

    # Choose a regression model (Random Forest Regressor in this case)
    self.model = RandomForestRegressor(random_state=42)

    # Train the model on the scaled training data
    self.model.fit(X_train_scaled, self.y_train)
```

‘Train_model’ method here uses standard scaling to the numeric features using ‘StandardScaler’. Then, will continue to Random Forest Regressor model and trains it on the scaled training data.

```python
def evaluate_model(self):
    # Make predictions on the scaled test set
    y_pred = self.model.predict(self.X_test_scaled)  # Use self.X_test_scaled

    # Evaluate the model on the test set
    mae = mean_absolute_error(self.y_test, y_pred)
    mse = mean_squared_error(self.y_test, y_pred)

    print(f'Mean Absolute Error: {mae}')
    print(f'Mean Squared Error: {mse}')

def save_model(self):
    # Apply Standard Scaling to the entire feature matrix
    X_scaled = self.scaler.fit_transform(self.X)

    # Perform cross-validation
    cv_scores = cross_val_score(self.model, X_scaled, self.y, cv=5, scoring='neg_mean_squared_error')

    # Convert the scores back to positive
    cv_scores_positive = -cv_scores

    # Print the cross-validated MSE scores
    print("Cross-Validated MSE Scores:", cv_scores_positive)
    print("Mean MSE:", cv_scores_positive.mean())

    # Save the trained model and scaler
    joblib.dump(self.model, 'air_quality_model.pkl')
    joblib.dump(self.scaler, 'scaler.pkl')
```

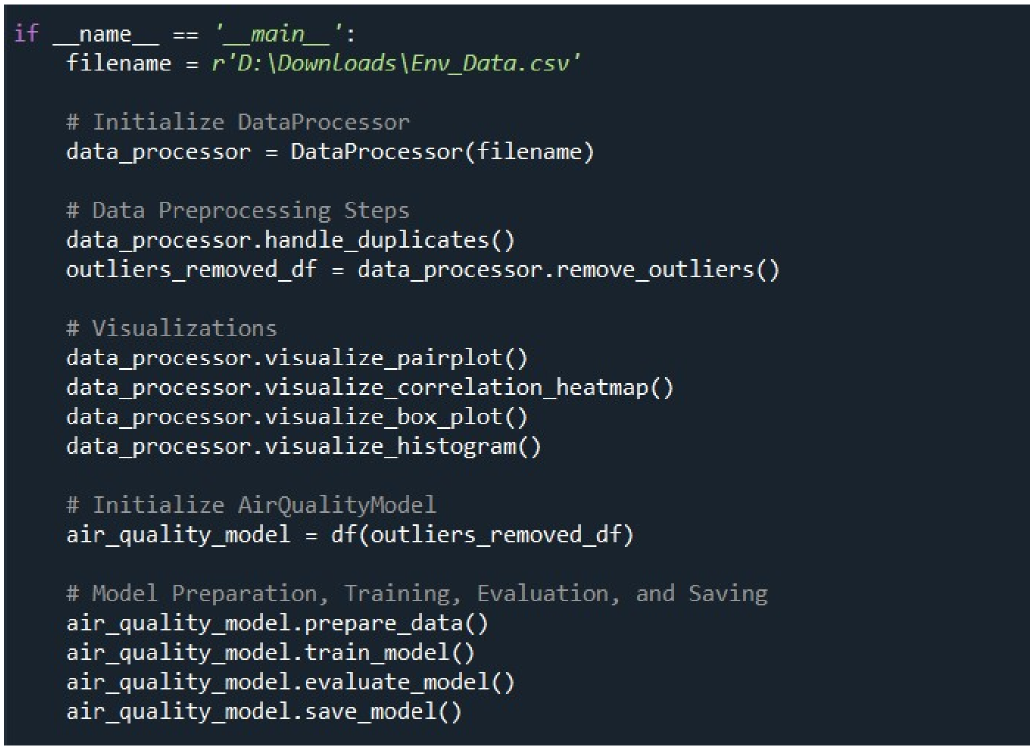
The trained model is used in the ‘evaluate_model’ to make predictions on the test set that is scaled. After that, calculation has been done for Mean Absolute Error (MAE) and Mean Squared Error (MSE) for metrics evaluation. From the results, MAE 26.16 is relatively close to the actual values, however MSE of 914.90 shows moderate squared error between predicted and actual values, as for cross-validation it is 862.56

```
Mean Absolute Error: 26.16007998409781
Mean Squared Error: 914.8994125349453
Cross-Validated MSE Scores: [789.39305373 866.97215037 858.56725206 873.91673506 923.9516581 ]
Mean MSE: 862.560169866063
```



Based on "Predicted vs Actual Values", the points are plotted on x-axis and the predicted value is at y-axis. The plot is used to test the accuracy of a predicted model. However, based on the observation, the range is about 40-60.

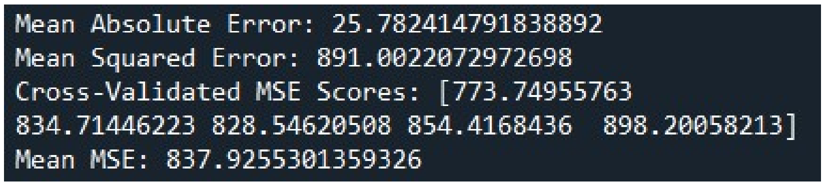
The dataset (D:\Downloads\Env_Data.csv) is analysed and processed to build the model for the air quality index. The process starts with ‘DataProcessor’ which manages the taks of data cleaning. Then visualisations are used to geenrate information from the dataset. Next, the ‘model’ is used to process the dataset and the model is processed step by step through data preperation, and traine through RandomForestRegressor and it is trained. The code here encapsulates a pipeline for data processing, exploration, machine learning model to create an air quality index predator.



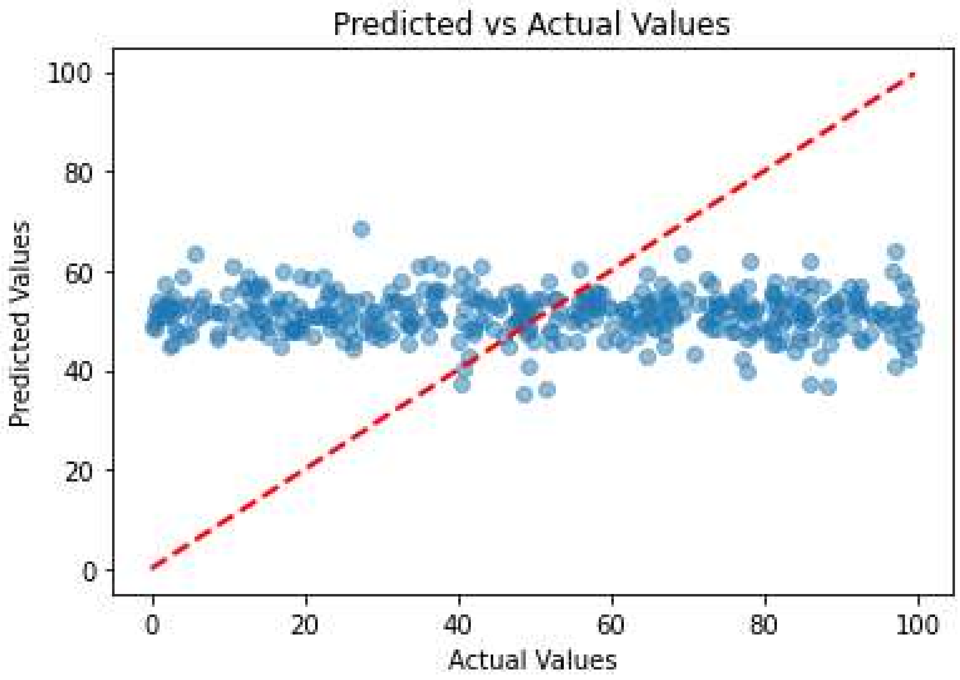
```
if __name__ == '__main__':
    filename = r'D:\Downloads\Env_Data.csv'

    # Initialize DataProcessor
    data_processor = DataProcessor(filename)

    # Data Preprocessing Steps
    data_processor.handle_duplicates()
    outliers_removed_df = data_processor.remove_outliers()

    # Visualizations
    data_processor.visualize_pairplot()
    data_processor.visualize_correlation_heatmap()
    data_processor.visualize_box_plot()
    data_processor.visualize_histogram()

    # Initialize AirQualityModel
    air_quality_model = df(outliers_removed_df)

    # Model Preparation, Training, Evaluation, and Saving
    air_quality_model.prepare_data()
    air_quality_model.train_model()
    air_quality_model.evaluate_model()
    air_quality_model.save_model()
```

5.0 This section will be discussing on improving the model.

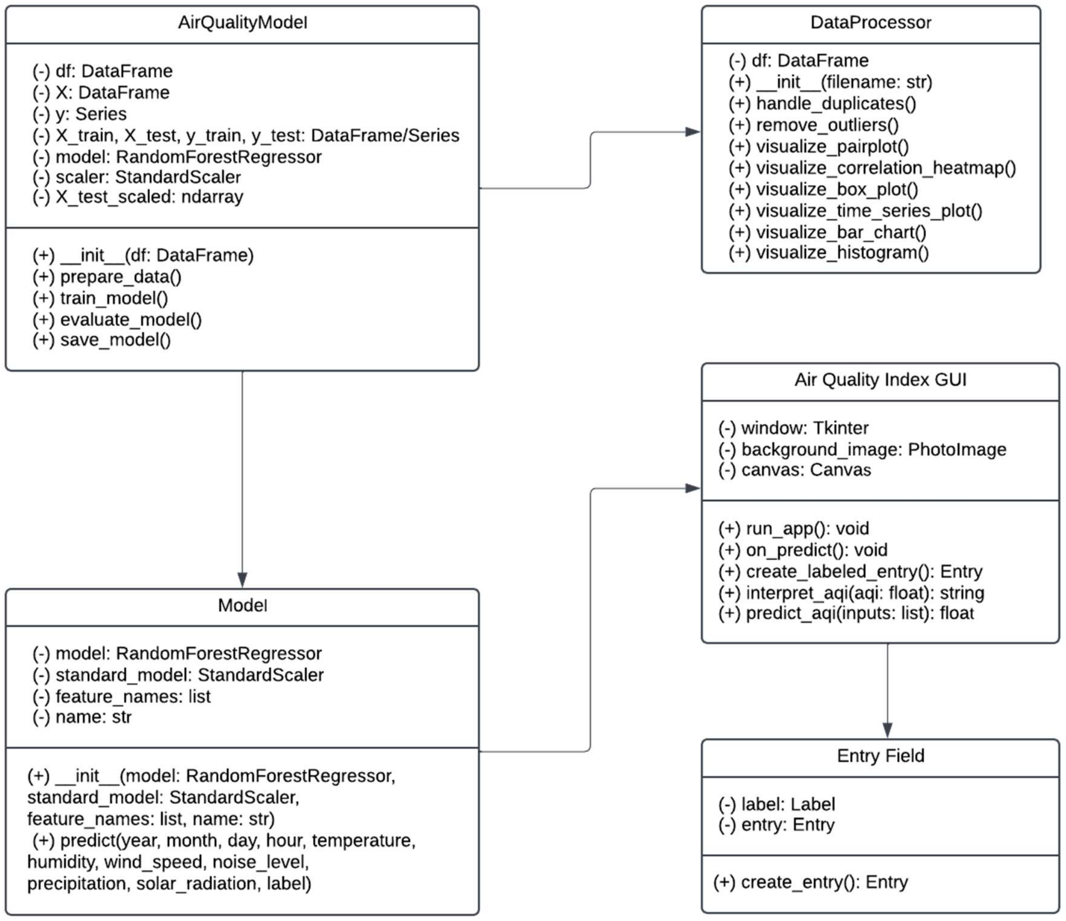
Adjustments have been made to improve the model's accuracy through the hyperparameters. However, the Mean Absolute Error, Mean Squared Error, Cross-Validated MSE Scores and Mean MSE have been reduced at the values, the results are now in the range of 40-60 and a consistent horizontal pattern instead of a visual of dots which follows the 45-degree line, this indicates the model might be potentially imbalanced.



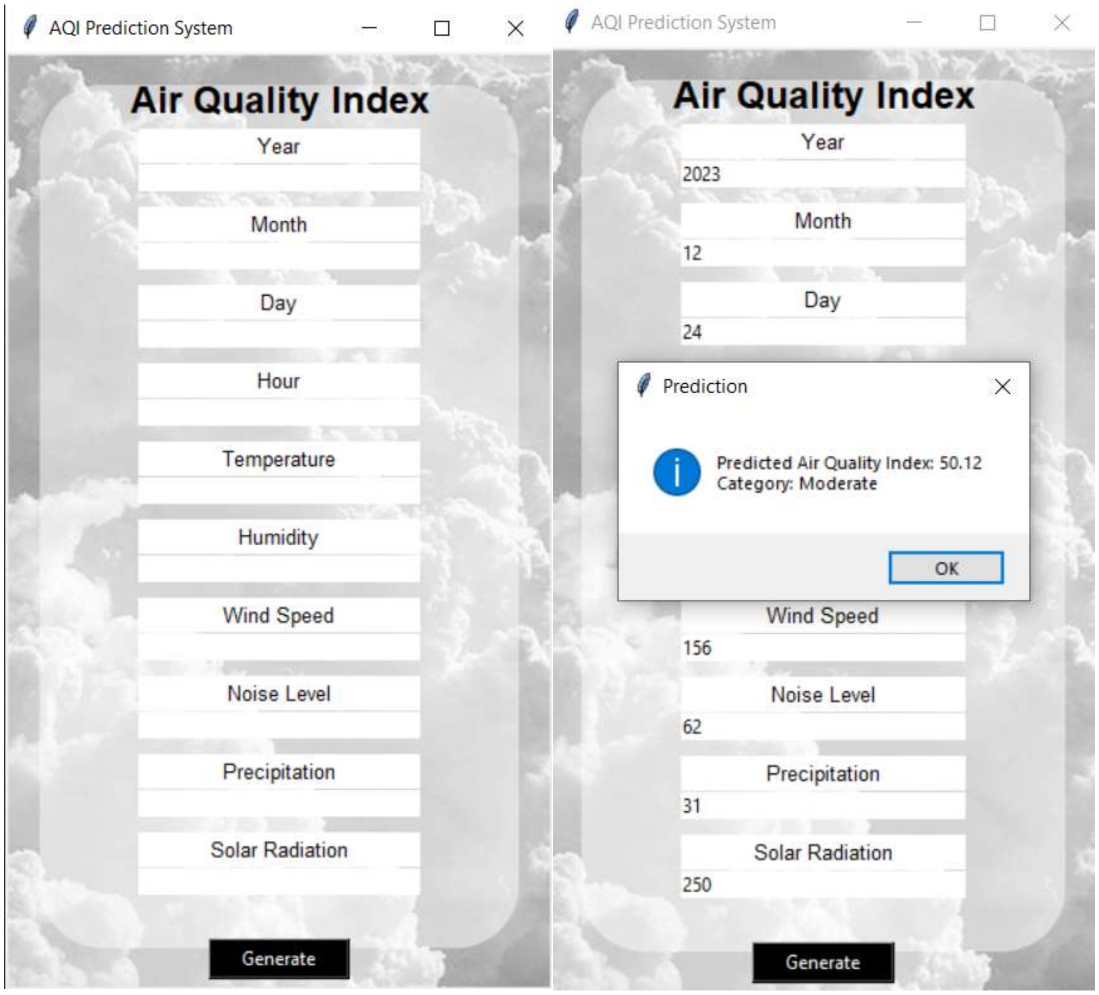
```
Mean Absolute Error: 25.782414791838892
Mean Squared Error: 891.0022072972698
Cross-Validated MSE Scores: [773.74955763
834.71446223 828.54620508 854.4168436  898.20058213]
Mean MSE: 837.9255301359326
```



6.0 This section will be showcasing the UML diagram based on the system.


AirQualityModel
(-) df: DataFrame
(-) X: DataFrame
(-) y: Series
(-) X_train, X_test, y_train, y_test: DataFrame/Series
(-) model: RandomForestRegressor
(-) scaler: StandardScaler
(-) X_test_scaled: ndarray
(+) __init__(df: DataFrame)
(+) prepare_data()
(+) train_model()
(+) evaluate_model()
(+) save_model()
DataProcessor
(-) df: DataFrame
(+) __init__(filename: str)
(+) handle_duplicates()
(+) remove_outliers()
(+) visualize_pairplot()
(+) visualize_correlation_heatmap()
(+) visualize_box_plot()
(+) visualize_time_series_plot()
(+) visualize_bar_chart()
(+) visualize_histogram()
Air Quality Index GUI
(-) window: Tkinter
(-) background_image: PhotoImage
(-) canvas: Canvas
(+) run_app(): void
(+) on_predict(): void
(+) create_labeled_entry(): Entry
(+) interpret_aqi(aqi: float): string
(+) predict_aqi(inputs: list): float
Model
(-) model: RandomForestRegressor
(-) standard_model: StandardScaler
(-) feature_names: list
(-) name: str
(+) __init__(model: RandomForestRegressor, standard_model: StandardScaler, feature_names: list, name: str)
(+) predict(year, month, day, hour, temperature, humidity, wind_speed, noise_level, precipitation, solar_radiation, label)
Entry Field
(-) label: Label
(-) entry: Entry
(+) create_entry(): Entry

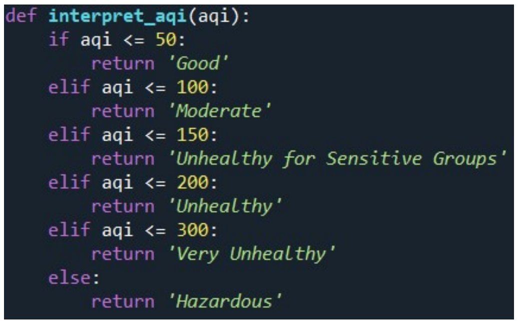
7.0 This section will be displaying the user-friendly interface developed for Air Quality Index.



Year: 2023
Month: 12
Day: 24
Wind Speed: 156
Noise Level: 62
Precipitation: 31
Solar Radiation: 250

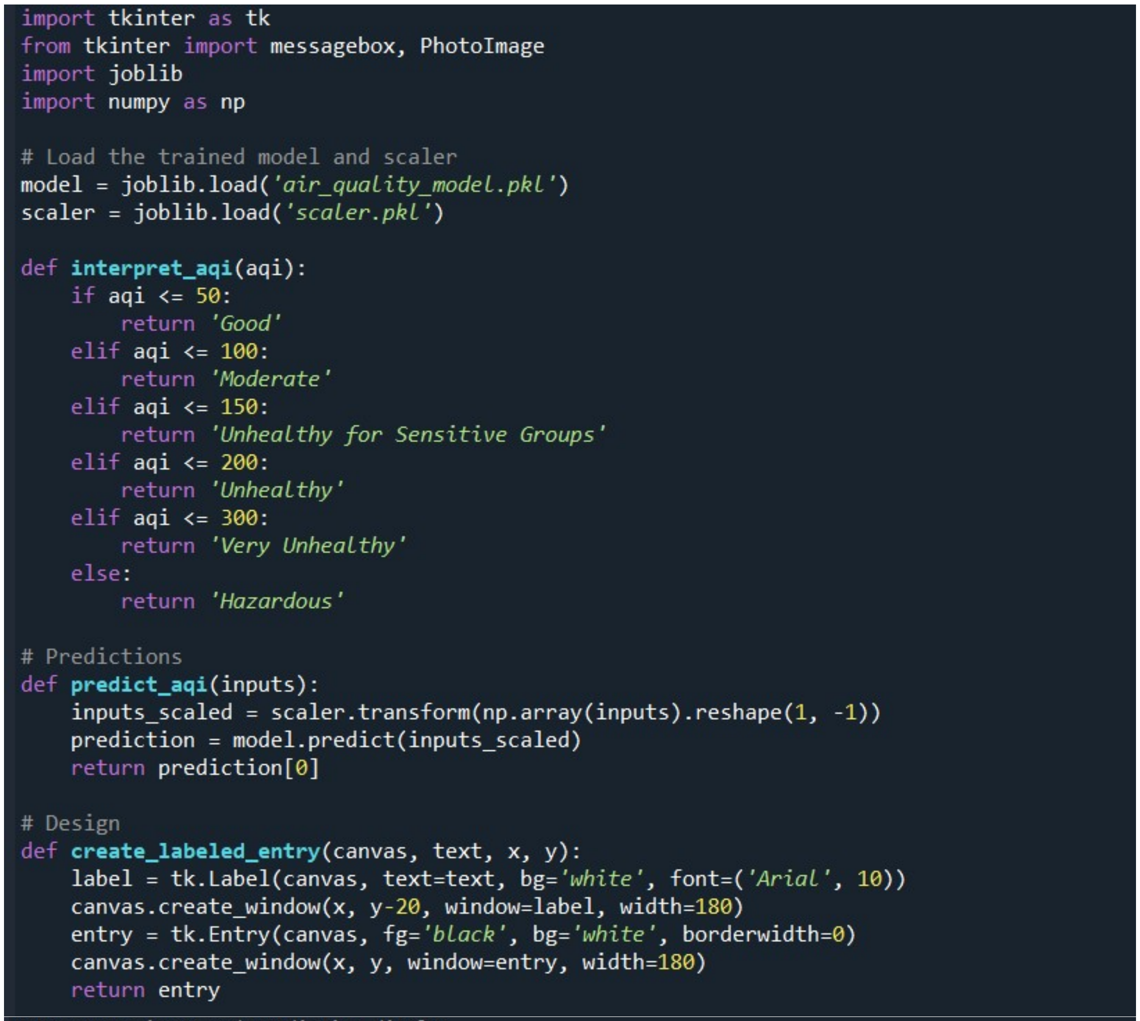
Predicted Air Quality Index: 50.12
Category: Moderate

The images above show the Air Quality Index Prediction system which is created through tkinter. The user is required to enter the value of the parameters and then generate it. The prediction will tell the value of the predicted air quality index and which category the air quality is. This allows users to understand the quality of the air.

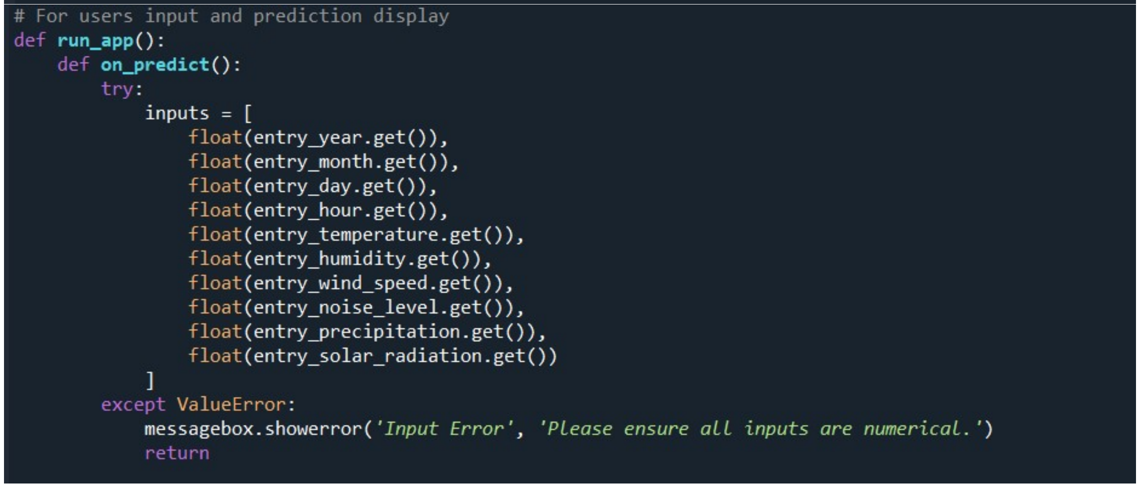


```python
def interpret_aqi(aqi):
    if aqi <= 50:
        return 'Good'
    elif aqi <= 100:
        return 'Moderate'
    elif aqi <= 150:
        return 'Unhealthy for Sensitive Groups'
    elif aqi <= 200:
        return 'Unhealthy'
    elif aqi <= 300:
        return 'Very Unhealthy'
    else:
        return 'Hazardous'
```

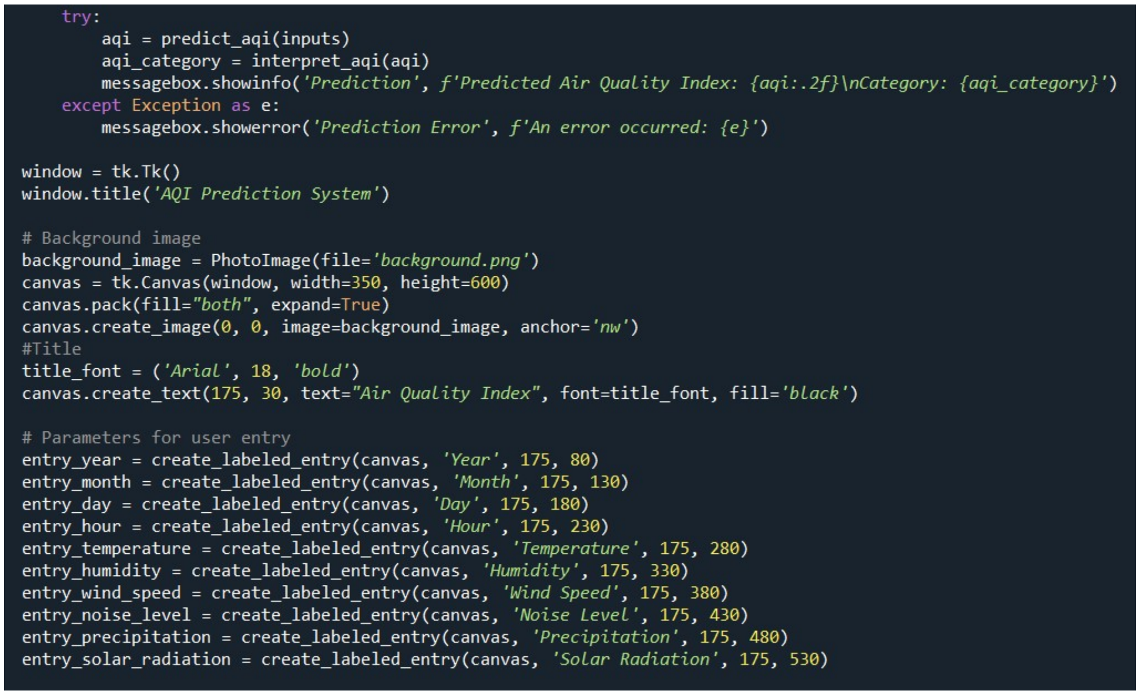
8.0 This section will be showing the code used to create the user interface.



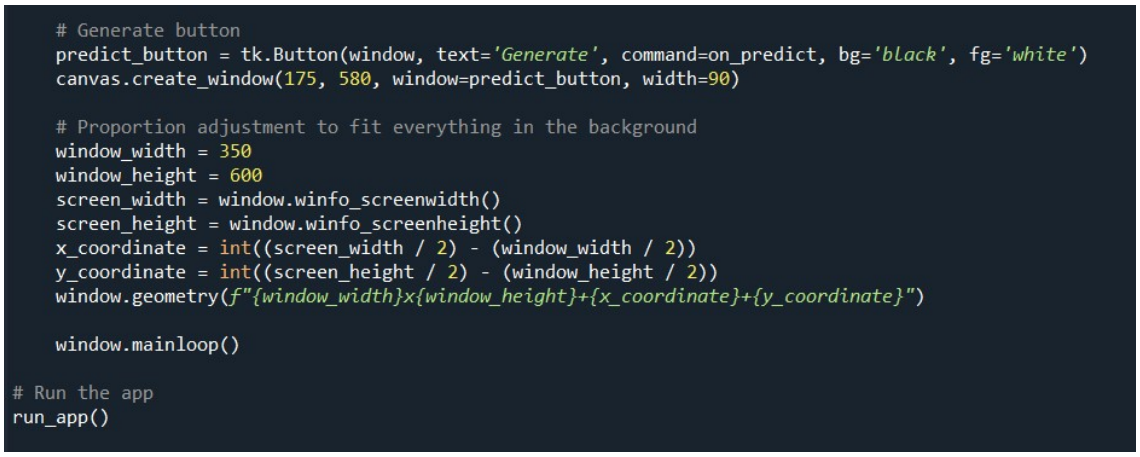
```python
import tkinter as tk
from tkinter import messagebox, PhotoImage
import joblib
import numpy as np

# Load the trained model and scaler
model = joblib.load('air_quality_model.pkl')
scaler = joblib.load('scaler.pkl')

def interpret_aqi(aqi):
    if aqi <= 50:
        return 'Good'
    elif aqi <= 100:
        return 'Moderate'
    elif aqi <= 150:
        return 'Unhealthy for Sensitive Groups'
    elif aqi <= 200:
        return 'Unhealthy'
    elif aqi <= 300:
        return 'Very Unhealthy'
    else:
        return 'Hazardous'

# Predictions
def predict_aqi(inputs):
    inputs_scaled = scaler.transform(np.array(inputs).reshape(1, -1))
    prediction = model.predict(inputs_scaled)
    return prediction[0]

# Design
def create_labeled_entry(canvas, text, x, y):
    label = tk.Label(canvas, text=text, bg='white', font=('Arial', 10))
    canvas.create_window(x, y-20, window=label, width=180)
    entry = tk.Entry(canvas, fg='black', bg='white', borderwidth=0)
    canvas.create_window(x, y, window=entry, width=180)
    return entry
```



```python
# For users input and prediction display
def run_app():
    def on_predict():
        try:
            inputs = [
                float(entry_year.get()),
                float(entry_month.get()),
                float(entry_day.get()),
                float(entry_hour.get()),
                float(entry_temperature.get()),
                float(entry_humidity.get()),
                float(entry_wind_speed.get()),
                float(entry_noise_level.get()),
                float(entry_precipitation.get()),
                float(entry_solar_radiation.get())
            ]
        except ValueError:
            messagebox.showerror('Input Error', 'Please ensure all inputs are numerical.')
            return
```



```python
    try:
        aqi = predict_aqi(inputs)
        aqi_category = interpret_aqi(aqi)
        messagebox.showinfo('Prediction', f'Predicted Air Quality Index: {aqi:.2f}\nCategory: {aqi_category}')
    except Exception as e:
        messagebox.showerror('Prediction Error', f'An error occurred: {e}')

window = tk.Tk()
window.title('AQI Prediction System')

# Background image
background_image = PhotoImage(file='background.png')
canvas = tk.Canvas(window, width=350, height=600)
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, image=background_image, anchor='nw')
#Title
title_font = ('Arial', 18, 'bold')
canvas.create_text(175, 30, text="Air Quality Index", font=title_font, fill='black')

# Parameters for user entry
entry_year = create_labeled_entry(canvas, 'Year', 175, 80)
entry_month = create_labeled_entry(canvas, 'Month', 175, 130)
entry_day = create_labeled_entry(canvas, 'Day', 175, 180)
entry_hour = create_labeled_entry(canvas, 'Hour', 175, 230)
entry_temperature = create_labeled_entry(canvas, 'Temperature', 175, 280)
entry_humidity = create_labeled_entry(canvas, 'Humidity', 175, 330)
entry_wind_speed = create_labeled_entry(canvas, 'Wind Speed', 175, 380)
entry_noise_level = create_labeled_entry(canvas, 'Noise Level', 175, 430)
entry_precipitation = create_labeled_entry(canvas, 'Precipitation', 175, 480)
entry_solar_radiation = create_labeled_entry(canvas, 'Solar Radiation', 175, 530)
```



```python
    # Generate button
    predict_button = tk.Button(window, text='Generate', command=on_predict, bg='black', fg='white')
    canvas.create_window(175, 580, window=predict_button, width=90)

    # Proportion adjustment to fit everything in the background
    window_width = 350
    window_height = 600
    screen_width = window.winfo_screenwidth()
    screen_height = window.winfo_screenheight()
    x_coordinate = int((screen_width / 2) - (window_width / 2))
    y_coordinate = int((screen_height / 2) - (window_height / 2))
    window.geometry(f"{window_width}x{window_height}+{x_coordinate}+{y_coordinate}")

    window.mainloop()

# Run the app
run_app()
```