In [52]:
```
#step 1: Import necassary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
```

In [7]:
```
#step 2: Load the dataset
dataset = pd.read_csv(r"C:\Users\Aadesh\Downloads\Air_Quality.csv")
print(dataset.columns)
```

```
Index(['Unique ID', 'Indicator ID', 'Name', 'Measure', 'Measure Info',
       'Geo Type Name', 'Geo Join ID', 'Geo Place Name', 'Time Period',
       'Start_Date', 'Data Value', 'Message'],
      dtype='object')
```

In [12]:
```
dataset.columns = dataset.columns.str.strip()
dataset['Start_Date'] = pd.to_datetime(dataset['Start_Date'], errors='coerce
print(dataset.head())
```

```
   Unique ID  Indicator ID                                 Name  \
0     179772           640  Boiler Emissions- Total SO2 Emissions
1     179785           640  Boiler Emissions- Total SO2 Emissions
2     178540           365              Fine particles (PM 2.5)
3     178561           365              Fine particles (PM 2.5)
4     823217           365              Fine particles (PM 2.5)

          Measure Measure Info Geo Type Name  Geo Join ID  \
0  Number per km2       number         UHF42        409.0
1  Number per km2       number         UHF42        209.0
2            Mean       mcg/m3         UHF42        209.0
3            Mean       mcg/m3         UHF42        409.0
4            Mean       mcg/m3         UHF42        409.0

            Geo Place Name          Time Period Start_Date  Data Value  \
0         Southeast Queens                 2015 2015-01-01         0.3
1  Bensonhurst - Bay Ridge                 2015 2015-01-01         1.2
2  Bensonhurst - Bay Ridge  Annual Average 2012 2011-12-01         8.6
3         Southeast Queens  Annual Average 2012 2011-12-01         8.0
4         Southeast Queens          Summer 2022 2022-06-01         6.1

   Message
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
```

In [13]:
```
# Remove rows where 'Start_Date' or 'Data Value' is missing
dataset = dataset.dropna(subset=['Start_Date', 'Data Value'])
```

In [14]:
```
# Step 4: Filter data for a specific pollutant (e.g., PM2.5 or SO2 Emissions
# You can filter based on the 'Name' column. For example, let's work with "F
pm25_data = dataset[dataset['Name'] == 'Fine particles (PM 2.5)']
```

In [15]:
```
# Step 5: Group the data by year or other time periods and calculate the mea
# Extract year from the 'Start_Date' and create a new column 'Year'
pm25_data['Year'] = pm25_data['Start_Date'].dt.year
```

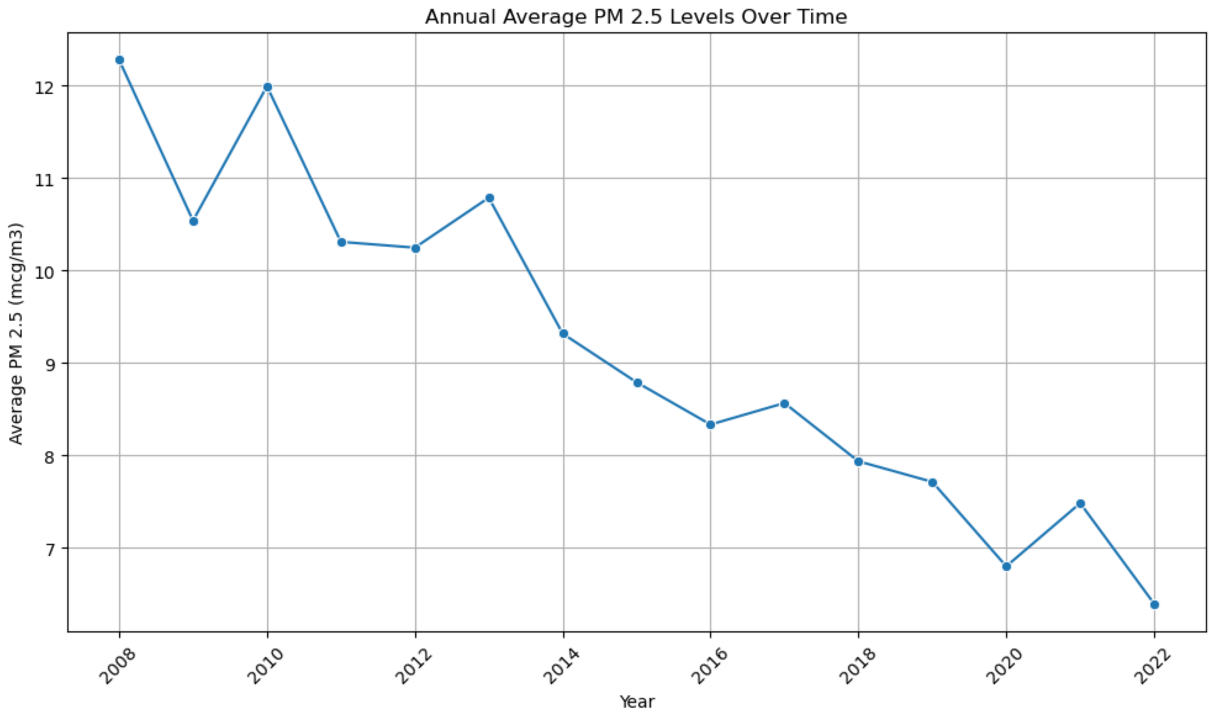
```
C:\Users\Aadesh\AppData\Local\Temp\ipykernel_19064\2754015938.py:3: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  pm25_data['Year'] = pm25_data['Start_Date'].dt.year
```

In [16]:
```
# Group by 'Year' and calculate the mean of 'Data Value'
pm25_trends = pm25_data.groupby('Year')['Data Value'].mean().reset_index()
```

In [19]:
```
# Step 6: Plot the trend over time
plt.figure(figsize=(10, 6))
sns.lineplot(data=pm25_trends, x='Year', y='Data Value', marker='o')
plt.title('Annual Average PM 2.5 Levels Over Time')
```

```
plt.xlabel('Year')
plt.ylabel('Average PM 2.5 (mcg/m3)')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```

```
C:\Users\Aadesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: Future
Warning: use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Aadesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: Future
Warning: use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [20]:
```
# You can filter based on the 'Name' column. For example, let's work with "E
Total_SO2_Emissions_data = dataset[dataset['Name'] == 'Boiler Emissions- Tot
```

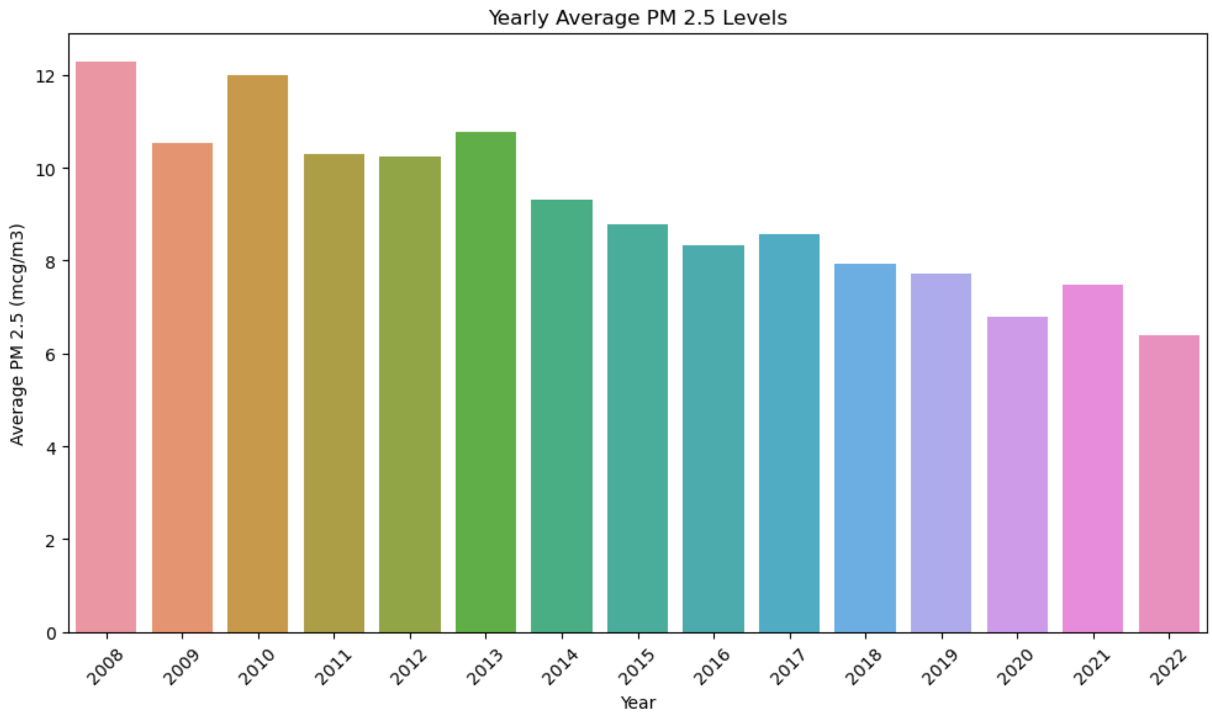
In [21]:
```
#Extract year from the 'Start_Date' and create a new column 'Year'
Total_SO2_Emissions_data['Year'] = Total_SO2_Emissions_data['Start_Date'].dt
```

```
C:\Users\Aadesh\AppData\Local\Temp\ipykernel_19064\979582381.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Total_SO2_Emissions_data['Year'] = Total_SO2_Emissions_data['Start_Date'].
dt.year
```

Loading [MathJax]/extensions/Safe.js

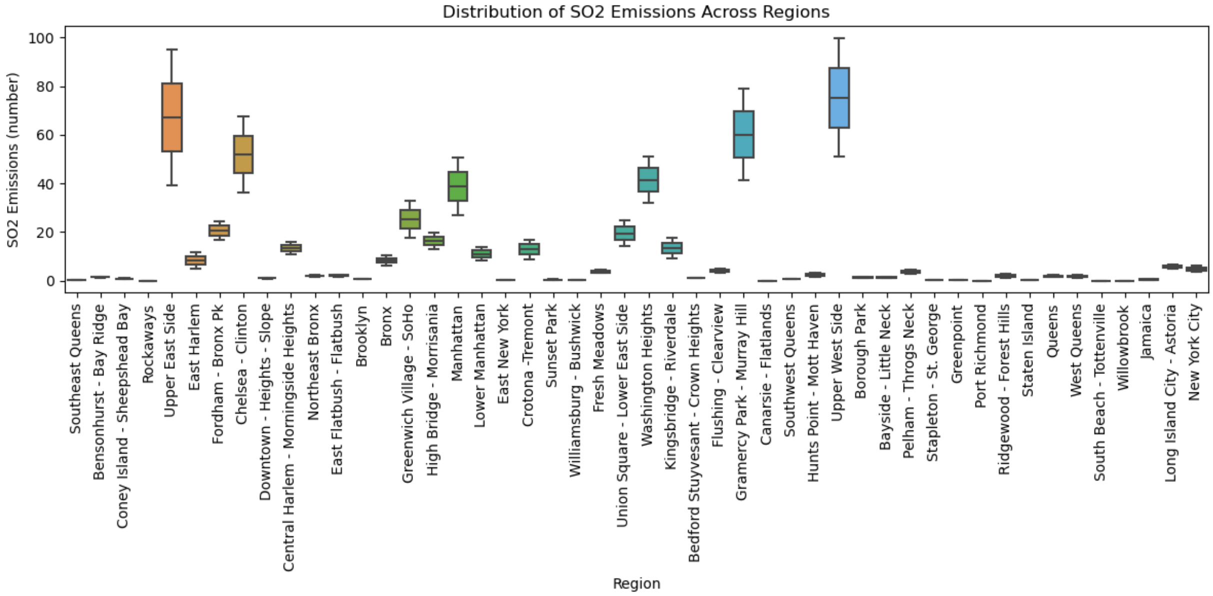
In [22]:
```
# Group by 'Year' and calculate the mean of 'Data Value'
Total_SO2_Emissions_trends = Total_SO2_Emissions_data.groupby('Year')['Data
```

In [33]:
```
# Pivot the dataset for correlation analysis
plt.figure(figsize=(10, 6))
sns.barplot(data=pm25_trends, x='Year', y='Data Value')
plt.title('Yearly Average PM 2.5 Levels')
plt.xlabel('Year')
plt.ylabel('Average PM 2.5 (mcg/m3)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [32]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=Total_SO2_Emissions_data, x='Geo Place Name', y='Data Value
plt.title('Distribution of SO2 Emissions Across Regions')
plt.xlabel('Region')
plt.ylabel('SO2 Emissions (number)')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [35]:
```
# Extract 'Year' and 'Month' from 'Start_Date'
dataset['Year'] = dataset['Start_Date'].dt.year
dataset['Month'] = dataset['Start_Date'].dt.month
```

In [36]:
```
# Step 4: Feature Engineering
# Encoding categorical columns
label_encoder = LabelEncoder()
dataset['Geo Place Name'] = label_encoder.fit_transform(dataset['Geo Place N
```

In [37]:
```
# Step 5: Select features and target variable
# Use 'Year', 'Month', and 'Geo Place Name' as features, and 'Data Value' as
features = dataset[['Year', 'Month', 'Geo Place Name']]
target = dataset['Data Value']
```

In [38]:
```
# Step 6: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_s
```

In [39]:
```
# Step 7: Model Building (Random Forest Regressor)
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[39]:
```
▼ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [40]:
```
# Step 8: Model Evaluation
# Predict on the test set
y_pred = model.predict(X_test)
```

In [41]:
```
# Calculate Mean Squared Error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [42]:
```
print(f'Mean Squared Error (MSE): {mse}')
```

Loading [MathJax]/extensions/Safe.js

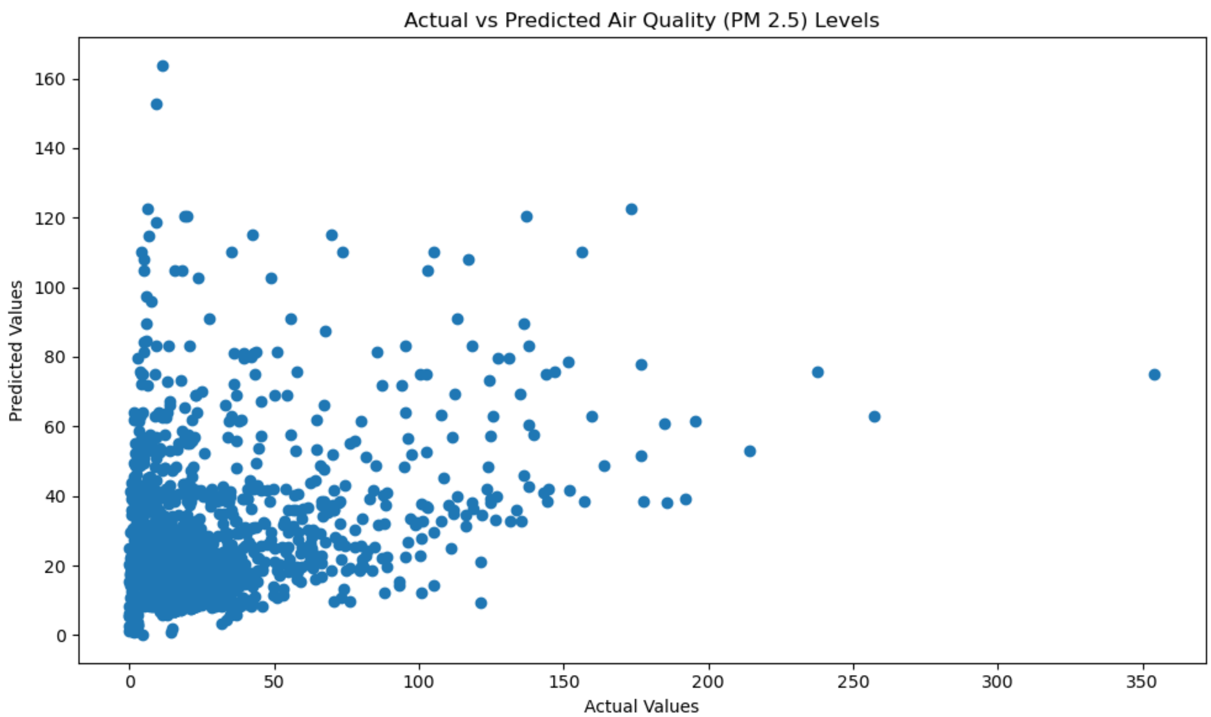
```
print(f'R-squared: {r2}')
```

```
Mean Squared Error (MSE): 525.6483090366401
R-squared: 0.032408331493381626
```

In [43]:
```
# Step 9: Cross-validation to check model performance
cv_scores = cross_val_score(model, features, target, cv=5, scoring='neg_mear
print(f'Cross-Validation MSE: {-cv_scores.mean()}')
```

```
Cross-Validation MSE: 559.7362432808002
```

In [44]:
```
# Step 10: Visualize Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Air Quality (PM 2.5) Levels')
plt.tight_layout()
plt.show()
```

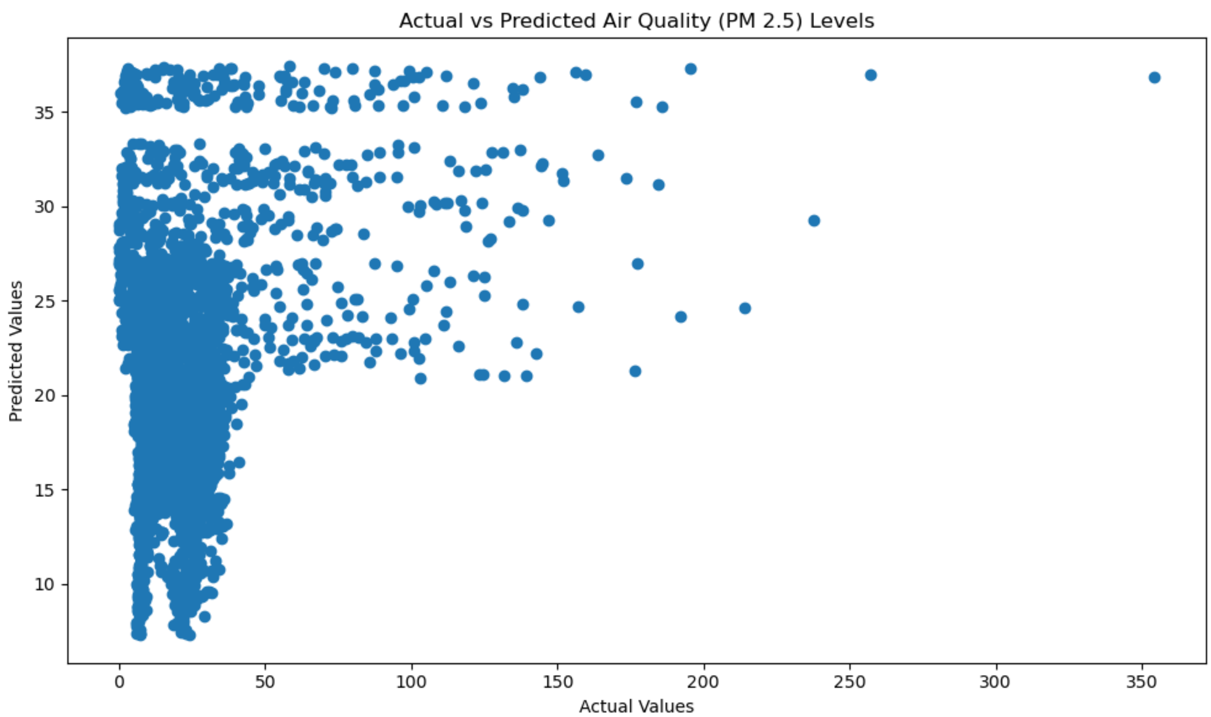


In [47]:
```
# Step 7: Model Building (Linear Regression)
model = LinearRegression()
model.fit(X_train, y_train)

# Step 8: Model Evaluation
# Predict on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared: {r2}')
```

Loading [MathJax]/extensions/Safe.js

```python
# Step 9: Cross-validation to check model performance
cv_scores = cross_val_score(model, features, target, cv=5, scoring='neg_mear
print(f'Cross-Validation MSE: {-cv_scores.mean()}')

# Step 10: Visualize Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Air Quality (PM 2.5) Levels')
plt.tight_layout()
plt.show()
```

```
Mean Squared Error (MSE): 502.83088692225306
R-squared: 0.07440969848178114
Cross-Validation MSE: 538.1963704018547
```

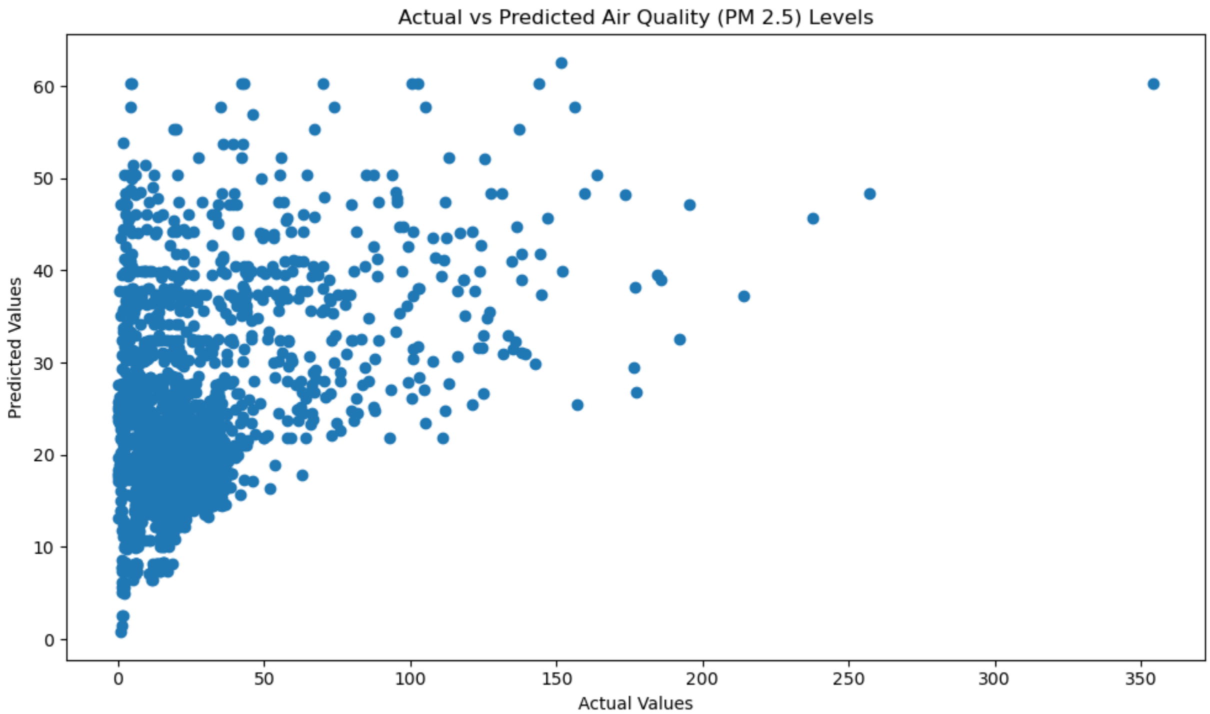


In [49]:

```python
# Step 7: Model Building (Gradient Boosting Regressor)
model = GradientBoostingRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 8: Model Evaluation
# Predict on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared: {r2}')

# Step 9: Cross-validation to check model performance
cv_scores = cross_val_score(model, features, target, cv=5, scoring='neg_mear
print(f'Cross-Validation MSE: {-cv_scores.mean()}')
```

```python
# Step 10: Visualize Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Air Quality (PM 2.5) Levels')
plt.tight_layout()
plt.show()
```

```
Mean Squared Error (MSE): 440.5397570618841
R-squared: 0.18907263421007925
Cross-Validation MSE: 500.4400652405528
```



In [ ]:

In [ ]: