UN MODÈLE DE PRÉVISION SUR 60 JOURS UTILISANT SARIMAX POUR LES PM10

UN PROJET GÉNÉRATIF ASSISTÉ PAR L'IA GÉNÉRATIVE PROMPT ENGINEERING

GARE MEDA - MONZA LOMBARDIE

PAR

ING. JAD GHANTOUS

M. NAD GHANTOUS

Ensemble de données extraites de l'EEE pour la station Meda entre 2017 et 2023 pour le polluant 5 (PM10)

Importer les bibliothèques

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import requests
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVR
from sklearn.preprocessing import PowerTransformer
import os
import pyarrow
from scipy.stats import zscore
import seaborn as sns
from xgboost import XGBRegressor
import requests
import csv
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import warnings
from pmdarima import auto_arima
from datetime import timedelta
from statsmodels.tsa.statespace.sarimax import SARIMAX
import warnings
import plotly.graph_objects as go
import numpy as np
from sklearn.metrics import mean_absolute_error
from math import sqrt
from matplotlib.colors import ListedColormap
```

Chargez l'ensemble de données et convertissez-le en fichier csv

In [2]:
```
# Read Parquet file into a pandas DataFrame
parquet_file_path = r'SPO.IT1034A_5_BETA_1998-02-05_00_00_00.parquet'
df = pd.read_parquet(parquet_file_path)
df.head()
```

Out[2]:

	Samplingpoint	Pollutant	Start	End	Value	Unit	AggType	Validity	Verification	ResultTime	DataCapture	FkObservationLog
0	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-01	2017-01-02	83.099998000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
1	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-02	2017-01-03	80.300003000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
2	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-03	2017-01-04	70.500000000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
3	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-04	2017-01-05	35.900002000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
4	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-05	2017-01-06	6.100000000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c

In [3]:
```
# Specify CSV file path
csv_file_path = r'Milan_Air_Quality_2017_2023_PM10_2.csv'

# Write DataFrame to CSV file
df.to_csv(csv_file_path, index=False)

print(f'Data written to {csv_file_path}')
df.head()
```

```
Data written to Milan_Air_Quality_2017_2023_PM10_2.csv
```

Out[3]:

	Samplingpoint	Pollutant	Start	End	Value	Unit	AggType	Validity	Verification	ResultTime	DataCapture	FkObservationLog
0	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-01	2017-01-02	83.099998000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
1	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-02	2017-01-03	80.300003000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
2	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-03	2017-01-04	70.500000000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
3	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-04	2017-01-05	35.900002000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c
4	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2017-01-05	2017-01-06	6.100000000000000000	ug.m-3	day	1	1	2018-07-10 13:10:30	None	406a5a07-a266-48aa-b90b-6da37804466c

In [4]:
```
df.tail()
```

Out[4]:

	Samplingpoint	Pollutant	Start	End	Value	Unit	AggType	Validity	Verification	ResultTime	DataCapture	FkObservationLog
1821	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2022-12-27	2022-12-28	59.000000000000000000	ug.m-3	day	1	1	2023-03-31 08:56:00	None	1ea7d119-b55b-4f7f-8b8e-30bf6724f92d
1822	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2022-12-28	2022-12-29	60.700000000000000000	ug.m-3	day	1	1	2023-03-31 08:56:00	None	1ea7d119-b55b-4f7f-8b8e-30bf6724f92d
1823	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2022-12-29	2022-12-30	78.000000000000000000	ug.m-3	day	1	1	2023-03-31 08:56:00	None	1ea7d119-b55b-4f7f-8b8e-30bf6724f92d
1824	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2022-12-30	2022-12-31	-9999.000000000000000000	ug.m-3	day	-1	1	2023-03-31 08:56:00	None	1ea7d119-b55b-4f7f-8b8e-30bf6724f92d
1825	IT/SPO.IT1034A_5_BETA_1998-02-05_00:00:00	5	2022-12-31	2023-01-01	71.800000000000000000	ug.m-3	day	1	1	2023-03-31 08:56:00	None	1ea7d119-b55b-4f7f-8b8e-30bf6724f92d

In [5]:
```
#read info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1826 entries, 0 to 1825
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   Samplingpoint     1826 non-null   object        
 1   Pollutant         1826 non-null   int32         
 2   Start             1826 non-null   datetime64[ns]
 3   End               1826 non-null   datetime64[ns]
 4   Value             1826 non-null   object        
 5   Unit              1826 non-null   object        
 6   AggType           1826 non-null   object        
 7   Validity          1826 non-null   int32         
 8   Verification      1826 non-null   int32         
 9   ResultTime        1826 non-null   datetime64[ns]
 10  DataCapture       0 non-null      object        
 11  FkObservationLog  1826 non-null   object        
dtypes: datetime64[ns](3), int32(3), object(6)
memory usage: 149.9+ KB
```

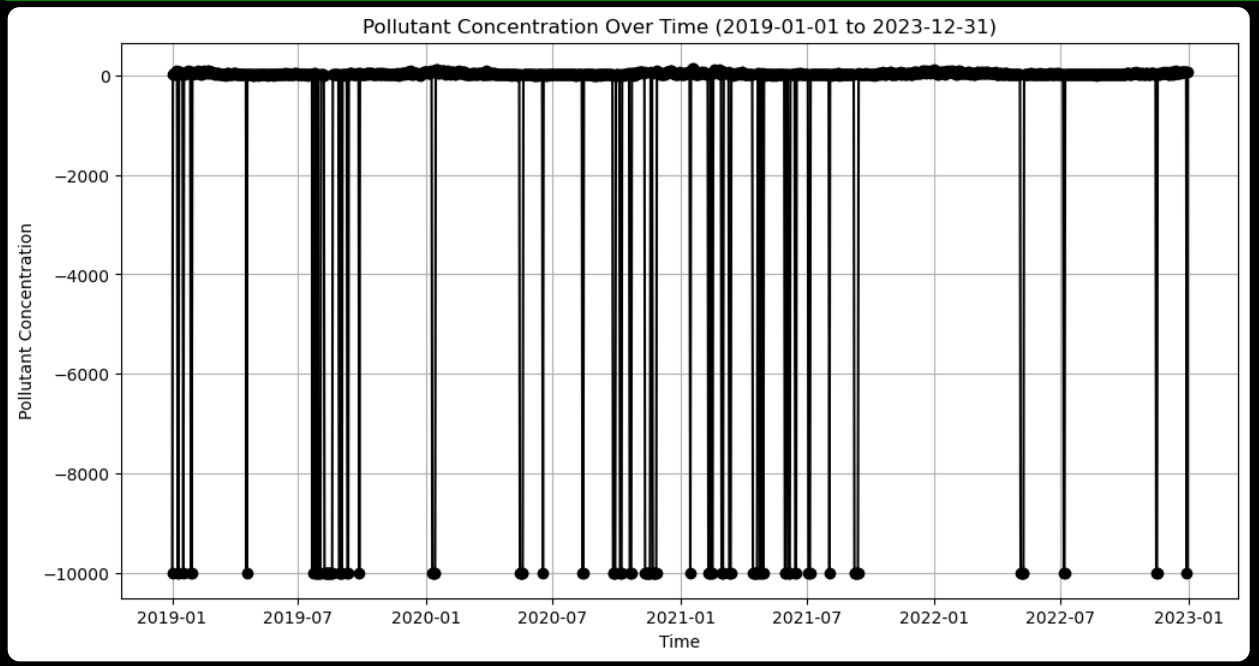
Traitez les données en ajustant les dates, en fixant des limites et en supprimant les valeurs aberrantes

In [6]:
```
#drop missing values
df.dropna()
```

Out[6]:

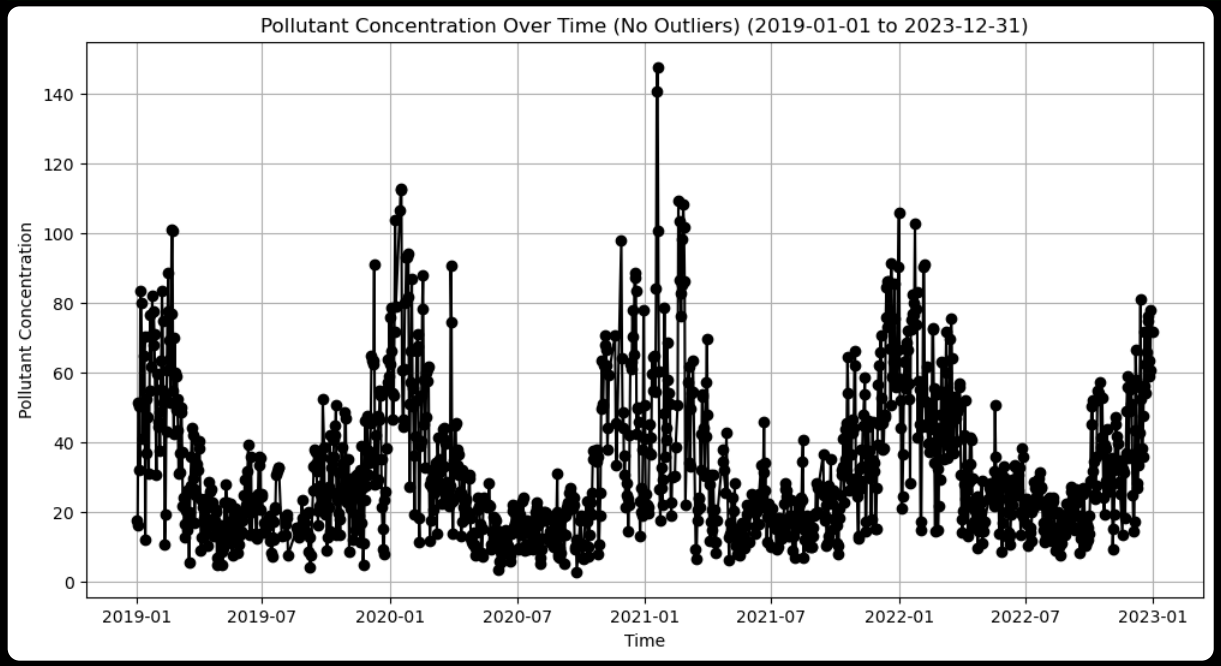
Samplingpoint	Pollutant	Start	End	Value	Unit	AggType	Validity	Verification	ResultTime	DataCapture	FkObservationLog

In [7]:
```
# Assuming 'Start' represents the time and 'Value' represents the pollutant concentration
df['Start'] = pd.to_datetime(df['Start'])  # Ensure 'Start' column is in datetime format

# Convert 'Value' to numeric type
df['Value'] = pd.to_numeric(df['Value'], errors='coerce')  # 'coerce' will convert non-numeric values to NaN

# Define the specific start and end dates
start_date = '2019-01-01'
end_date = '2023-12-31'

# Filter the DataFrame based on the specified date range
filtered_df = df[(df['Start'] >= start_date) & (df['Start'] <= end_date)]

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(filtered_df['Start'], filtered_df['Value'], marker='o', linestyle='-', color='black')
plt.title(f'Pollutant Concentration Over Time ({start_date} to {end_date})')
plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.grid(True)
plt.show()
```



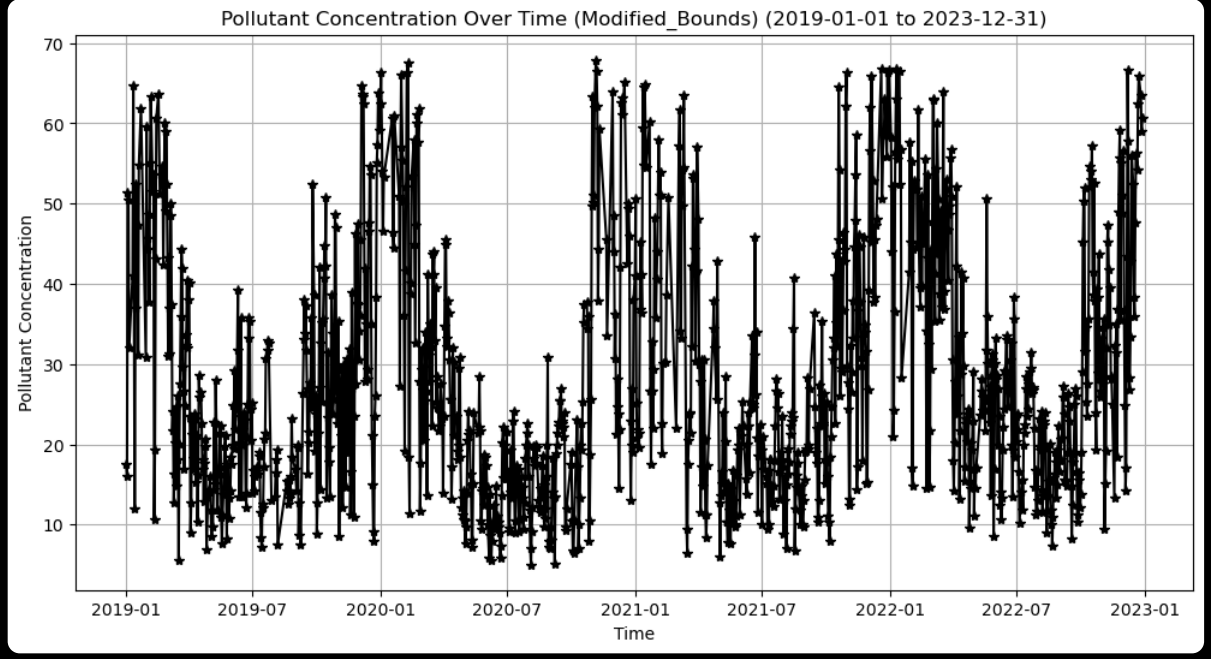
In [8]:
```
# Calculate Z-scores for the 'Value' column
z_scores = zscore(filtered_df['Value'].astype(float))

# Define a threshold for Z-scores to identify outliers
z_threshold = 1  # Adjust as needed

# Create a boolean mask for outliers
outliers_mask = abs(z_scores) > z_threshold
```

```
# Drop rows with outliers
filtered_df_no_outliers = filtered_df[~outliers_mask]

# Plotting without outliers
plt.figure(figsize=(12, 6))
plt.plot(filtered_df_no_outliers['Start'], filtered_df_no_outliers['Value'], marker='o', linestyle='-', color='black')
plt.title(f'Pollutant Concentration Over Time (No Outliers) ({start_date} to {end_date})')
plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.grid(True)
plt.show()
```



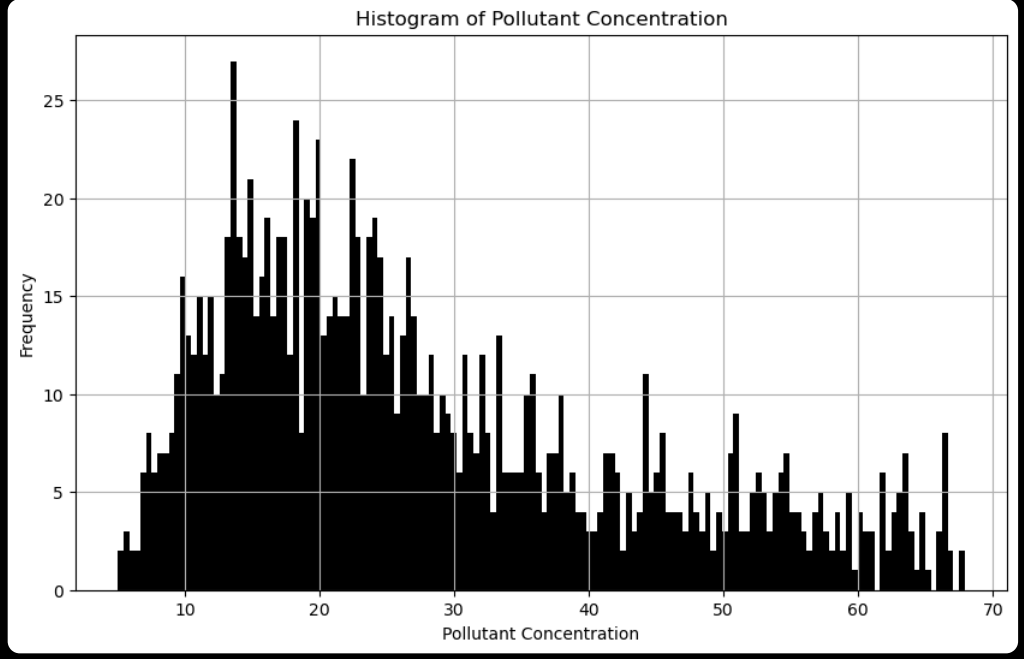
In [9]:

```
# Define your custom range for acceptable values
lower_bound = 5  # Adjust as needed
upper_bound = 68  # Adjust as needed

# Filter the DataFrame based on the custom range
filtered_df_custom_range = filtered_df.loc[
    (filtered_df['Value'].astype(float) >= lower_bound) & (filtered_df['Value'].astype(float) <= upper_bound)
].copy()

# Plotting without outliers using the custom range
plt.figure(figsize=(12, 6))
plt.plot(filtered_df_custom_range['Start'], filtered_df_custom_range['Value'], marker='*', linestyle='-', color='black')
plt.title(f'Pollutant Concentration Over Time (Modified_Bounds) ({start_date} to {end_date})')
plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.grid(True)
plt.show()
```

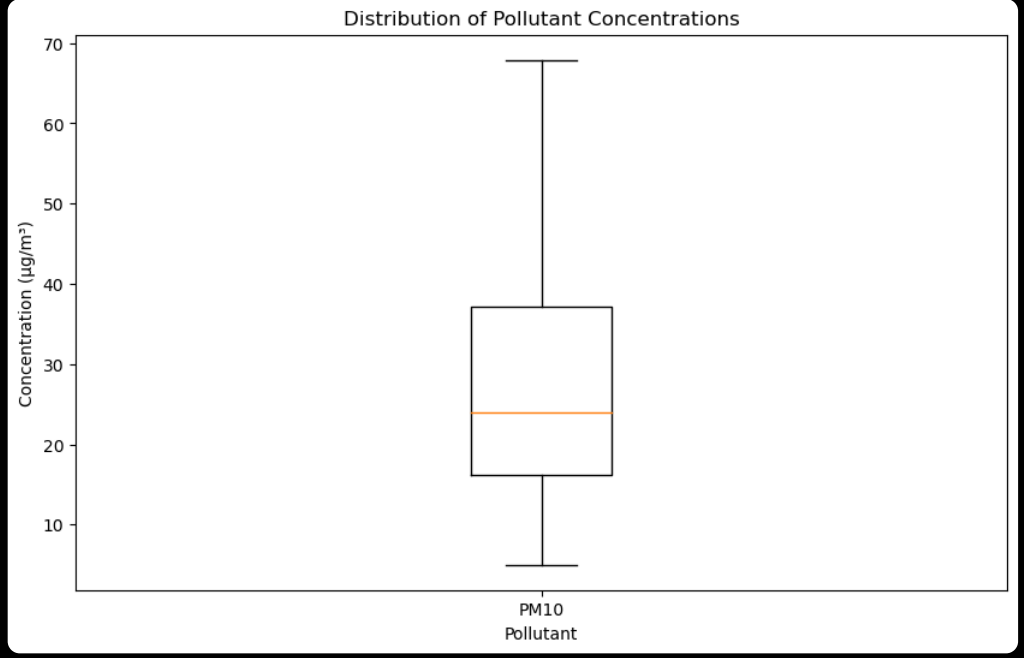


In [10]:

```
# Plot a histogram to visualize the distribution of 'Value' column
plt.figure(figsize=(10, 6))
plt.hist(filtered_df_custom_range['Value'].astype(float), bins=150, color='black')
plt.title('Histogram of Pollutant Concentration')
plt.xlabel('Pollutant Concentration')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

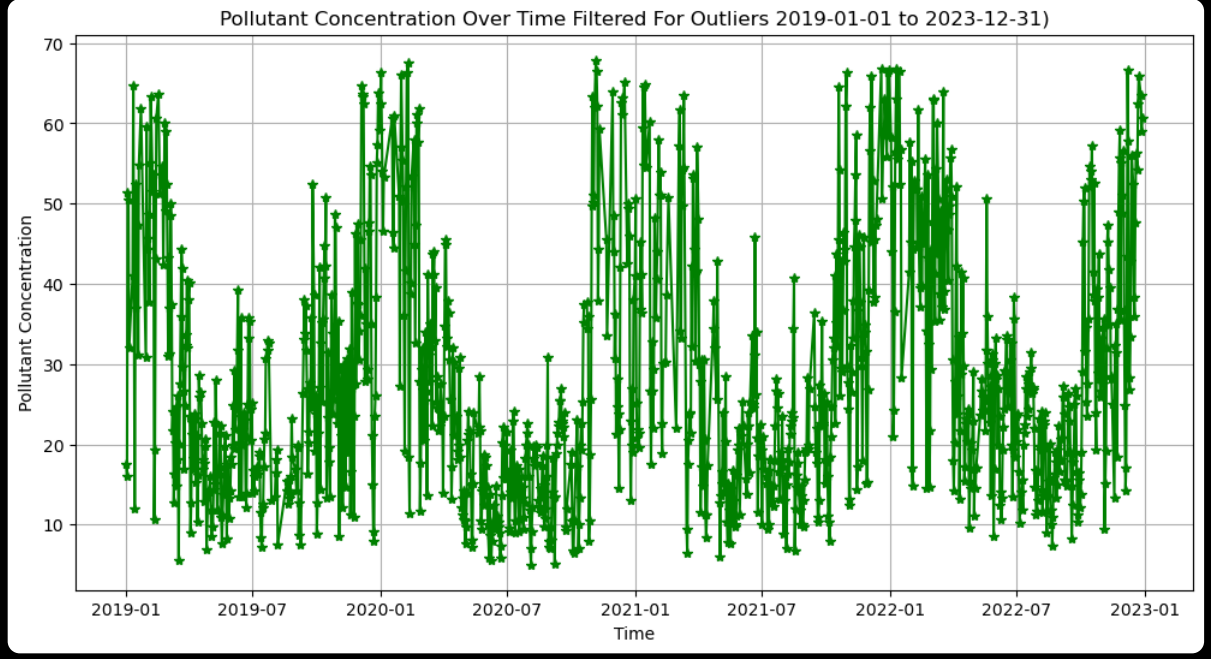


In [11]:
```
#Create a box plot of the pollutants
plt.figure(figsize=(10, 6))
plt.boxplot([filtered_df_custom_range['Value']], labels=['PM10'])
plt.xlabel('Pollutant')
plt.ylabel('Concentration (µg/m³)')
plt.title('Distribution of Pollutant Concentrations')
plt.show()
```



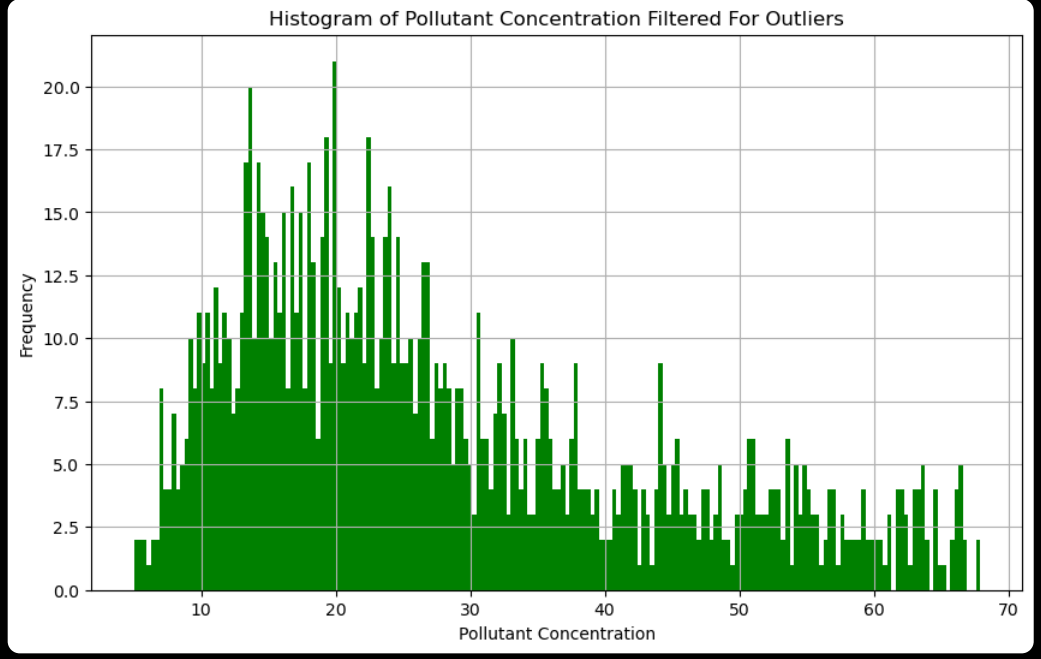
In [12]:
```python
# Apply outlier detection using the IQR method
Q1 = filtered_df_custom_range['Value'].quantile(0.25)
Q3 = filtered_df_custom_range['Value'].quantile(0.75)
IQR = Q3 - Q1

# Define the bounds for acceptable values using IQR
lower_bound_iqr = Q1 - 1.5 * IQR
upper_bound_iqr = Q3 + 1.5 * IQR

# Filter the DataFrame based on the IQR bounds
filtered_df_final = filtered_df_custom_range[
    (filtered_df_custom_range['Value'] >= lower_bound_iqr) & (filtered_df_custom_range['Value'] <= upper_bound_iqr)
]
```

In [13]:
```python
# Plotting without outliers using the custom range
plt.figure(figsize=(12, 6))
plt.plot(filtered_df_final['Start'], filtered_df_final['Value'], marker='*', linestyle='-', color='green')
plt.title(f'Pollutant Concentration Over Time Filtered For Outliers {start_date} to {end_date})')
plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.grid(True)
plt.show()
```

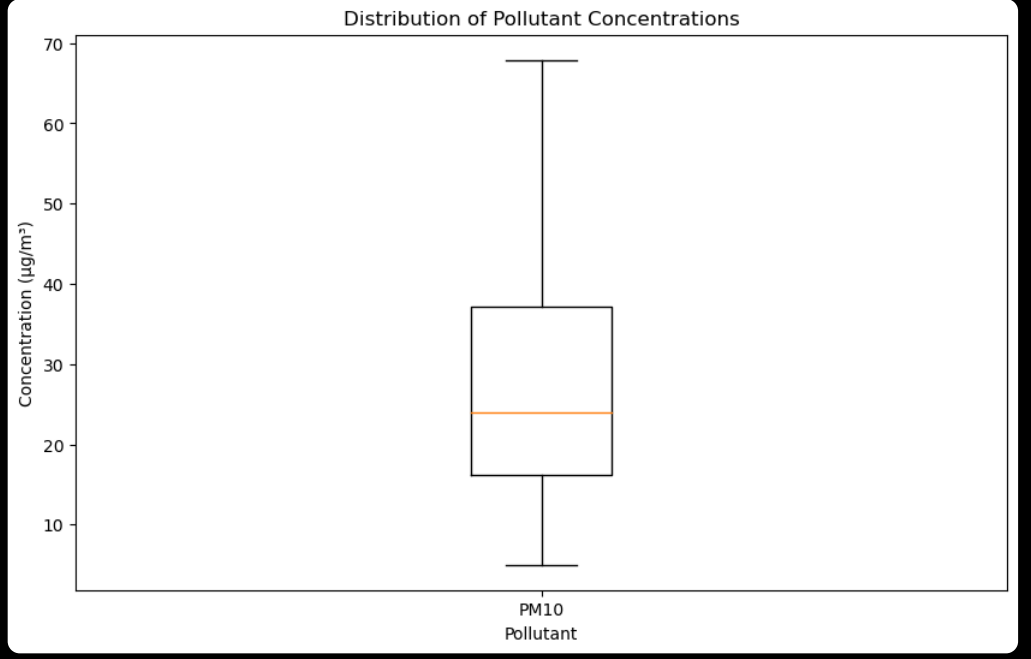


In [14]:

```
# Plot a histogram to visualize the distribution of 'Value' column
plt.figure(figsize=(10, 6))
plt.hist(filtered_df_final['Value'].astype(float), bins=200, color='green')
plt.title('Histogram of Pollutant Concentration Filtered For Outliers')
plt.xlabel('Pollutant Concentration')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



In [15]:
```
#Create a box plot of the pollutants
plt.figure(figsize=(10, 6))
plt.boxplot([filtered_df_final['Value']], labels=['PM10'])
plt.xlabel('Pollutant')
plt.ylabel('Concentration (µg/m³)')
plt.title('Distribution of Pollutant Concentrations')
plt.show()
```



In [16]:
```
# calcualte mean and std dev

pm10 = filtered_df_final['Value']
pm10_avg = pm10.mean()
pm10_std = pm10.std()
pm10_med = pm10.median()
pm10_skew = pm10.skew()
pm10_kurt = pm10.kurt()

# Print the results
print(f"Mean: {pm10_avg:.2f}")
print(f"Standard Deviation: {pm10_std:.2f}")
print(f"Median: {pm10_med:.2f}")
print(f"Skewness: {pm10_skew:.2f}")
print(f"Kurtosis: {pm10_kurt:.2f}")
```

```
Mean: 28.10
Standard Deviation: 15.28
Median: 24.00
Skewness: 0.81
Kurtosis: -0.29
```

SARIMAX MODEL

In [17]:
```
print(filtered_df_final.columns)
```

```
Index(['Samplingpoint', 'Pollutant', 'Start', 'End', 'Value', 'Unit',
       'AggType', 'Validity', 'Verification', 'ResultTime', 'DataCapture',
       'FkObservationLog'],
      dtype='object')
```

In [18]:
```
warnings.filterwarnings("ignore")
# Assuming 'final_df_modified' is the DataFrame you created earlier

# Assuming a DataFrame named 'filtered_df_final'
filtered_df_final['Start'] = pd.to_datetime(filtered_df_final['Start'])
filtered_df_final['Timestamp'] = filtered_df_final['Start']
filtered_df_final.set_index('Start', inplace=True)
```

```
filtered_df_final.head()
# Drop unnecessary columns
filtered_df_final.drop(['Samplingpoint', 'Pollutant', 'End', 'Unit', 'AggType', 'Validity',
                        'Verification', 'ResultTime', 'DataCapture', 'FkObservationLog'],
                       axis=1, inplace=True)

# Create a new DataFrame for the SARIMA model
sarima_df = filtered_df_final[['Timestamp', 'Value']].copy()

# Print information about NaN values
print("Number of NaN values in the dataset:")
print(filtered_df_final.isnull().sum())
```

```
Number of NaN values in the dataset:
Value        0
Timestamp    0
dtype: int64
```

In [19]:

```
# Create 'final_df_modified' containing the last values of the filtered DataFrame
final_df_modified = sarima_df.groupby(sarima_df.index.date).last().reset_index()

# Drop NaN values in the 'Value' column
final_df_modified.dropna(subset=['Value'], inplace=True)

#Create a SARIMA Model
sarima_order = (7, 1, 3, 24)  # Example SARIMA order, adjust based on your data
sarima_model = SARIMAX(final_df_modified['Value'], order=(4, 0, 1), seasonal_order=sarima_order)
sarima_results = sarima_model.fit()

# Split the data into train and test sets
train_size = int(len(final_df_modified) * 0.5)
train, test = final_df_modified[0:train_size].copy(), final_df_modified[train_size:].copy()
# Forecast future values with a continuous date range index
forecast_steps = len(test)
forecast_index = pd.date_range(test.index[0], periods=forecast_steps, freq='D')
forecast = sarima_results.get_forecast(steps=forecast_steps, index=forecast_index)

# Check for NaN values in the forecasted values
if forecast.predicted_mean.isnull().any():
    # Handle NaN values in the forecasted values
    forecast.predicted_mean.fillna(final_df_modified['Value'].mean(), inplace=True)

# Align lengths of 'test' and forecasted values
forecast.predicted_mean = forecast.predicted_mean[:len(test)]

# Check for NaN values in 'test' again
test.isnull().sum()
final_df_modified.head()
```

Out[19]:

	index	Timestamp	Value
0	2019-01-02	2019-01-02	17.5
1	2019-01-03	2019-01-03	16.1
2	2019-01-04	2019-01-04	51.4
3	2019-01-05	2019-01-05	50.5
4	2019-01-06	2019-01-06	32.0

In [20]:

```
# Handle NaN values in 'test' (if any)
test['Value'].fillna(test['Value'].mean(), inplace=True)
test['Timestamp'].interpolate(inplace=True)
test.head()
```

Out[20]:

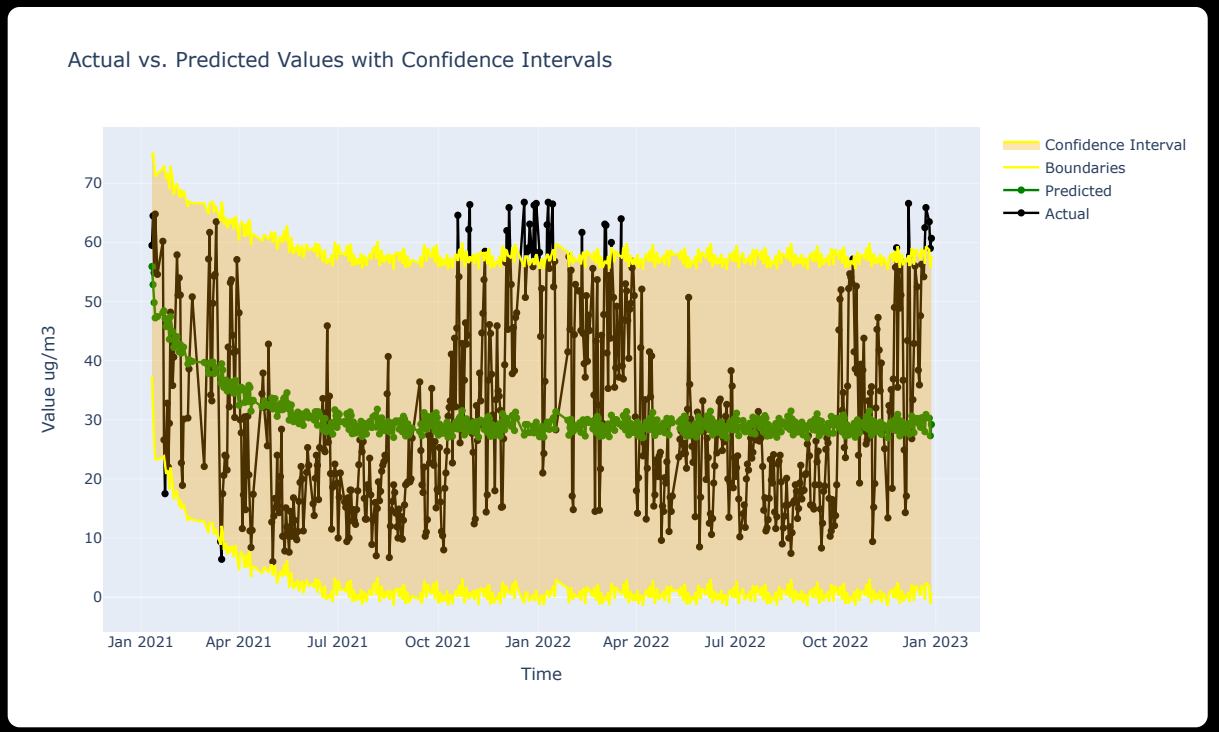
	index	Timestamp	Value
616	2021-01-11	2021-01-11	59.5
617	2021-01-12	2021-01-12	64.5
618	2021-01-13	2021-01-13	54.9
619	2021-01-14	2021-01-14	64.8
620	2021-01-16	2021-01-16	54.6

In [21]:
```python
def evaluate_sarimax(model, actual_values, forecast_values):
    """
    Calculates MAE and Theil's U statistic for a SARIMAX model given actual and forecast values.

    Args:
        model: The fitted SARIMAX model object.
        actual_values: A list or array of actual values.
        forecast_values: A list or array of forecast values.

    Returns:
        Dictionary containing MAE and Theil's U statistic.
    """

    # Convert to arrays
    actual_values = np.array(actual_values)
    forecast_values = np.array(forecast_values)

    # Calculate Mean Absolute Error
    mae = mean_absolute_error(actual_values, forecast_values)

    # Calculate Theil's U statistic
    rmse_forecast = sqrt(mean_squared_error(actual_values, forecast_values))
    rmse_naive = sqrt(mean_squared_error(actual_values, [actual_values[0]] * len(actual_values)))
    theil_u = rmse_forecast / rmse_naive


    # Return results as a dictionary
    return {"MAE": mae, "Theil's U": theil_u}


# Assuming 'test' contains your test set with actual values
# Forecast future values
forecast_steps = len(test)
forecast = sarima_results.get_forecast(steps=forecast_steps)
forecast_values = forecast.predicted_mean

# Actual values from your test set
actual_values = test['Value']

# Evaluate the SARIMAX model
evaluation_results = evaluate_sarimax(sarima_results, actual_values, forecast_values)
```

In [22]:
```python
# Print mean and predicted mean
print(f'Mean of Actual Values: {test["Value"].mean()}')
print(f'Mean of Predicted Values: {forecast.predicted_mean.mean()}')
print("The Mean Absolute Error is:", evaluation_results["MAE"])
print("The Theil's U is:", evaluation_results["Theil's U"])


# Create a time series plot
fig = go.Figure()

# Add traces for actual and predicted values
fig.add_trace(go.Scatter(x=test['Timestamp'], y=actual_values, mode='lines+markers', name='Actual', line=dict(color='black')))
fig.add_trace(go.Scatter(x=test['Timestamp'], y=forecast_values, mode='lines+markers', name='Predicted', line=dict(color='green')))
fig.add_trace(go.Scatter(x=test['Timestamp'], y=forecast.conf_int().iloc[:, 0], mode='lines', line=dict(color='yellow'), name='Boundaries'))
fig.add_trace(go.Scatter(x=test['Timestamp'], y=forecast.conf_int().iloc[:, 1], mode='lines', line=dict(color='yellow'), fill='tonexty', fillcolor='rgba(255,165,0,0.3)', name='Confidence Interval'))

# Update layout
fig.update_layout(title='Actual vs. Predicted Values with Confidence Intervals',
```

```
                  xaxis_title='Time',
                  yaxis_title='Value ug/m3',
                  showlegend=True,
                  height=600,  # Adjust the height as needed
                  width=1000)  # Adjust the width as needed

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

```
Mean of Actual Values: 29.705519480519477
Mean of Predicted Values: 30.33155692144122
The Mean Absolute Error is: 12.66574730128063
The Theil's U is: 0.45561103668331765
```



Prédire les valeurs pour janvier et janvier février 2023

In [23]:

```
# Predict the value for a specific timestamp (replace 'desired_timestamp' with your desired timestamp)
desired_timestamp = pd.Timestamp('2023-01-01')
steps_to_forecast = 60
forecast_final = sarima_results.get_forecast(steps=steps_to_forecast)
#forecast_values = forecast.predicted_mean
#forecast_at_timestamp = sarima_results.get_forecast(steps=1, index=[desired_timestamp])
forecasted_value = forecast_final.predicted_mean
# Print the forecasted value
#print(f"Forecasted value at {desired_timestamp}: {forecast_at_timestamp.predicted_mean.iloc[0]}")
# Assuming 'final_df_modified' is the DataFrame you created earlier

# Set the 'Timestamp' column as the index and convert it to a DatetimeIndex
final_df_modified.set_index('Timestamp', inplace=True)
final_df_modified.index = pd.to_datetime(final_df_modified.index)

# Now the last index of final_df_modified is a Timestamp
last_date = final_df_modified.index[-1]

# Create a date range starting from the day after the last day in your training data
```

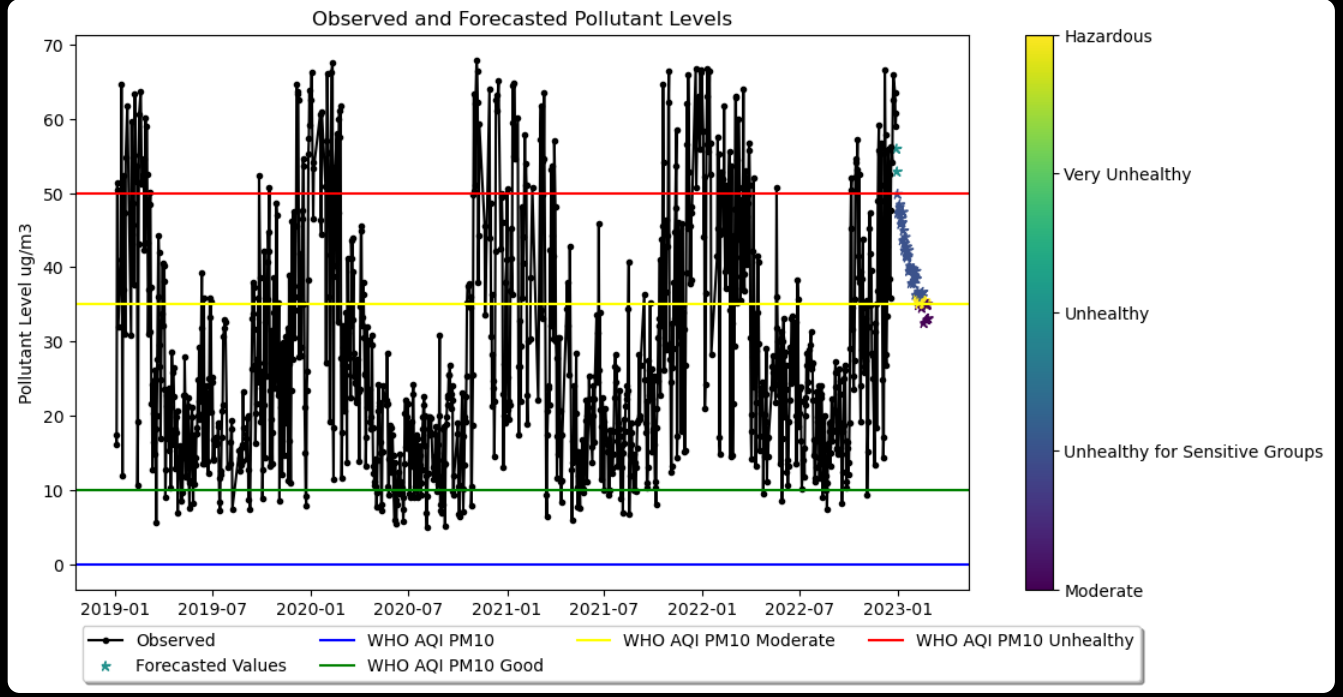
```
date_range = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=steps_to_forecast)

# Set the date range as the index of your forecasted values
forecasted_value.index = date_range

print(forecasted_value)
```

```
2022-12-29    55.932374
2022-12-30    52.862473
2022-12-31    49.821928
2023-01-01    47.242225
2023-01-02    47.457761
2023-01-03    48.092482
2023-01-04    48.429910
2023-01-05    47.143387
2023-01-06    45.780732
2023-01-07    46.575457
2023-01-08    46.019283
2023-01-09    43.605595
2023-01-10    47.434350
2023-01-11    44.921977
2023-01-12    42.257302
2023-01-13    43.215898
2023-01-14    44.043364
2023-01-15    42.707274
2023-01-16    41.504460
2023-01-17    42.749037
2023-01-18    41.287367
2023-01-19    41.516532
2023-01-20    42.290985
2023-01-21    39.453480
2023-01-22    40.183784
2023-01-23    39.872112
2023-01-24    39.699810
2023-01-25    37.826759
2023-01-26    39.594737
2023-01-27    39.619080
2023-01-28    39.597932
2023-01-29    39.783947
2023-01-30    37.816366
2023-01-31    38.847920
2023-02-01    38.659769
2023-02-02    36.241533
2023-02-03    39.474590
2023-02-04    38.437741
2023-02-05    35.483332
2023-02-06    35.988187
2023-02-07    36.713205
2023-02-08    34.881994
2023-02-09    35.071347
2023-02-10    36.246287
2023-02-11    35.482460
2023-02-12    35.660661
2023-02-13    36.403510
2023-02-14    34.496885
2023-02-15    36.034056
2023-02-16    36.657724
2023-02-17    35.071710
2023-02-18    32.460625
2023-02-19    35.017865
2023-02-20    35.500888
2023-02-21    35.326648
2023-02-22    35.103694
2023-02-23    32.940479
2023-02-24    34.882989
2023-02-25    34.997708
2023-02-26    33.090516
Freq: D, Name: predicted_mean, dtype: float64
```

In [24]:

```
# Function to classify the forecasted value and assign a color category
def classify_and_color(value):
    if value <= 10:
        return 1  # Green
```

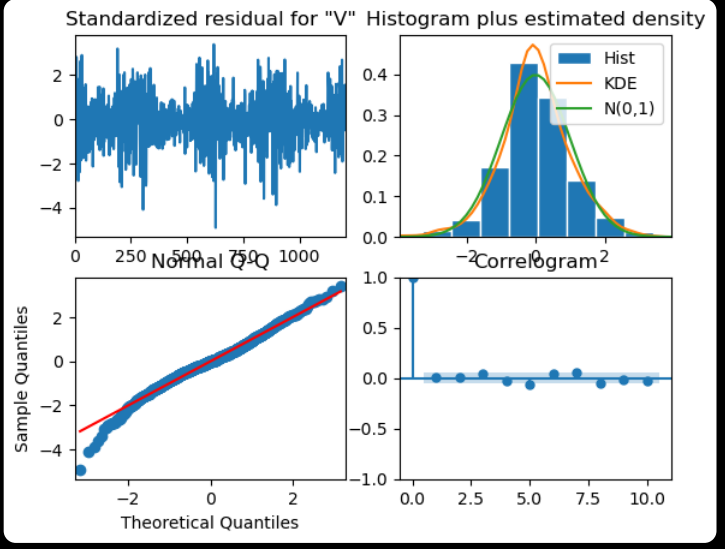
```
    elif 11 <= value <= 35:
        return 2  # Yellow
    elif 36 <= value <= 50:
        return 3  # Orange
    elif 51 <= value <= 75:
        return 4  # Red
    elif 76 <= value <= 100:
        return 5  # Purple
    else:
        return 6  # Maroon

# Classify the forecasted values based on WHO standards
classification = forecasted_value.apply(classify_and_color)

# Filter the data to include only the dates from January 1 to January 30, 2023
filtered_data = final_df_modified[(final_df_modified.index >= '2019-01-01') & (final_df_modified.index <= '2023-01-30')]

# Visualize the results with correct timestamp index
plt.figure(figsize=(12, 6))
plt.plot(filtered_data.index, filtered_data['Value'], label='Observed', marker='.', linestyle='-', color='black')

# Use plt.scatter to create scatter plot
scatter = plt.scatter(forecasted_value.index, forecasted_value, c=classification, marker='*', linestyle='--', label='Forecasted Values')

who_standard_low = 0
who_standard_green = 10
who_standard_yellow = 35
who_standard_high = 50

plt.axhline(y=who_standard_low, color='blue', linestyle='-', label='WHO AQI PM10')
plt.axhline(y=who_standard_green, color='green', linestyle='-', label='WHO AQI PM10 Good')
plt.axhline(y=who_standard_yellow, color='yellow', linestyle='-', label='WHO AQI PM10 Moderate')
plt.axhline(y=who_standard_high, color='red', linestyle='-', label='WHO AQI PM10 Unhealthy')

# Create a custom colormap
colors = ["Green", "Yellow", "Orange", "Red", "Purple", "Maroon"]
cmap = ListedColormap(colors)

# Add colorbar with a custom colormap
cbar = plt.colorbar(scatter, ticks=np.arange(1, 7), cmap=cmap)
cbar.set_ticklabels(['Good', 'Moderate', 'Unhealthy for Sensitive Groups', 'Unhealthy', 'Very Unhealthy', 'Hazardous'])

plt.title('Observed and Forecasted Pollutant Levels')
plt.xlabel('Time')
plt.ylabel('Pollutant Level ug/m3')
plt.legend(loc='upper center', bbox_to_anchor=(0.6, -0.05), fancybox=True, shadow=True, ncol=4)
plt.show()
```



Les valeurs prévues suivent la tendance des valeurs observées qui partent de malsaines et évoluent vers un AQI PM10 modéré en février 2023.

Évaluer le modèle

In [25]:
```
# Check diagnostic plots
sarima_results.plot_diagnostics()
plt.show()
```



In [26]:
```
# Check diagnostic summary
print(sarima_results.summary())
# Check residuals
residuals = sarima_results.resid
print("Descriptive Statistics of Residuals:")
print(residuals.describe())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                               Value   No. Observations:                 1232
Model:             SARIMAX(4, 0, 1)x(7, 1, [1, 2, 3], 24)   Log Likelihood               -4472.920
Date:                                Wed, 24 Jan 2024   AIC                           8977.841
Time:                                        23:30:20   BIC                           9059.388
Sample:                                             0   HQIC                          9008.549
                                                - 1232
Covariance Type:                                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.5577      0.042     36.768      0.000       1.475       1.641
ar.L2         -0.6799      0.054    -12.697      0.000      -0.785      -0.575
ar.L3          0.1226      0.050      2.459      0.014       0.025       0.220
ar.L4         -0.0138      0.030     -0.456      0.649      -0.073       0.045
ma.L1         -0.8800      0.034    -26.148      0.000      -0.946      -0.814
ar.S.L24      -1.0754      0.188     -5.729      0.000      -1.443      -0.707
ar.S.L48      -0.9071      0.195     -4.658      0.000      -1.289      -0.525
ar.S.L72       0.0698      0.066      1.050      0.294      -0.061       0.200
ar.S.L96       0.0405      0.065      0.621      0.535      -0.087       0.169
ar.S.L120      0.0038      0.066      0.058      0.954      -0.126       0.134
ar.S.L144     -0.0333      0.057     -0.580      0.562      -0.146       0.079
ar.S.L168     -0.0295      0.039     -0.763      0.445      -0.105       0.046
ma.S.L24       0.1057      0.707      0.149      0.881      -1.281       1.492
ma.S.L48      -0.1475      0.771     -0.191      0.848      -1.660       1.365
ma.S.L72      -0.9553      0.696     -1.373      0.170      -2.319       0.408
sigma2        87.8267     59.038      1.488      0.137     -27.887     203.540
===================================================================================
Ljung-Box (L1) (Q):                   0.21   Jarque-Bera (JB):               84.10
Prob(Q):                              0.64   Prob(JB):                        0.00
Heteroskedasticity (H):               0.99   Skew:                           -0.17
Prob(H) (two-sided):                  0.89   Kurtosis:                        4.25
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
Descriptive Statistics of Residuals:
count    1232.000000
mean        0.795917
std        11.764559
min       -46.988637
25%        -5.213756
50%        -0.003671
75%         6.090494
max        64.678563
dtype: float64
```

Le sommaire:

Le modèle est globalement bien adapté, comme l'indiquent ses statistiques et ses graphiques de prévision. [Le U de Theil < 1, ce qui signifie qu'il fonctionne mieux qu'une prédiction naïve]

Les concentrations de PM10 semblent connaître des pics en hiver. [Peut-être à cause de la température froide qui provoque une inversion et des émissions de chaleur]

Le modèle Sarima n'est plus précis après 60 jours. [Cela commence à devenir moins précis et s'écarte de la tendance historique]

Il y a place à amélioration, principalement en trouvant un meilleur ensemble de données contenant des valeurs de température et des vitesses de vent qui ont un impact sur les concentrations de PM10.

Des améliorations sont possibles en choisissant une meilleure combinaison de paramètres pouvant être calculés avec de meilleures spécifications matérielles informatiques.

Ce modèle sert de prévision préliminaire des concentrations de PM10 de janvier et janvier. Février 2023.