Exp 10 WATER QUALITY ANALYSIS

AIM:

The aim of this project is to assess and monitor water quality by analyzing various physical, chemical, and biological parameters in water samples. This analysis will help identify potential contamination sources and determine if water meets safety standards for drinking, agriculture, and recreational use.

BACKGROUND THEORY:

Water quality analysis is essential for ensuring safe and sustainable water resources. Contaminants in water can arise from industrial discharge, agricultural runoff, sewage, and natural sources, impacting public health and ecosystems. Key water quality indicators include pH, dissolved oxygen, turbidity, biological oxygen demand (BOD), total dissolved solids (TDS), nitrates, phosphates, and heavy metals like lead or mercury. Analyzing these parameters allows for assessing the water's suitability for various uses and identifying pollution trends. Machine learning and statistical methods, such as clustering, regression, and classification, are increasingly used in water quality studies to uncover patterns, predict contamination levels, and aid in real-time monitoring.

DATASET:

A suitable dataset for this project would include water samples with multiple quality parameters recorded over time and across different locations. Commonly used datasets for water quality analysis include:

- UCI Machine Learning Repository - Water Quality Dataset: Contains water quality parameters for river water samples.
- World Water Quality Assessment Dataset: Offers global water quality data with a range of chemical and biological attributes.
- EPA Water Quality Data: Provided by the U.S. Environmental Protection Agency, this dataset includes comprehensive water quality metrics for various bodies of water.
- Indian Water Quality Dataset (Central Pollution Control Board): Contains water quality information from various rivers across India.

PROCEDURE:

Here's a stepwise procedure for predicting the quality of the water:

1. **Data Collection and Preprocessing:**
    - Collect or acquire water quality datasets containing physical, chemical, and biological measurements.
    - Clean the data by handling missing values, removing duplicates, and ensuring consistency in units and formats.
    - Normalize or standardize the data, especially for parameters like pH, turbidity, and heavy metals, to facilitate analysis.

2. **Exploratory Data Analysis (EDA):**

    - Conduct an initial analysis to understand data distribution, correlations, and outliers.
    - Visualize key parameters (e.g., pH levels, TDS, dissolved oxygen) to identify trends and anomalies over time or location.
    - Use correlation matrices to explore relationships between variables, such as the link between nitrates and dissolved oxygen.

3. **Model Development:**

    - Clustering: Apply clustering techniques, such as K-means or hierarchical clustering, to categorize water samples based on quality levels.
    - Classification: Use classification models like Decision Trees or Support Vector Machines to label samples as "Safe" or "Unsafe" based on water quality thresholds.
    - Regression: Develop regression models to predict specific parameters (e.g., BOD levels) based on other water quality metrics.
    - Feature Engineering: Create new features from existing parameters to enhance model accuracy.

4. **Model Evaluation and Validation:**

    - Split the dataset into training and testing sets to evaluate model performance.
    - Use accuracy, precision, recall, and F1-score for classification models, and metrics like RMSE for regression models.
    - Perform cross-validation to improve the reliability of results and avoid overfitting.

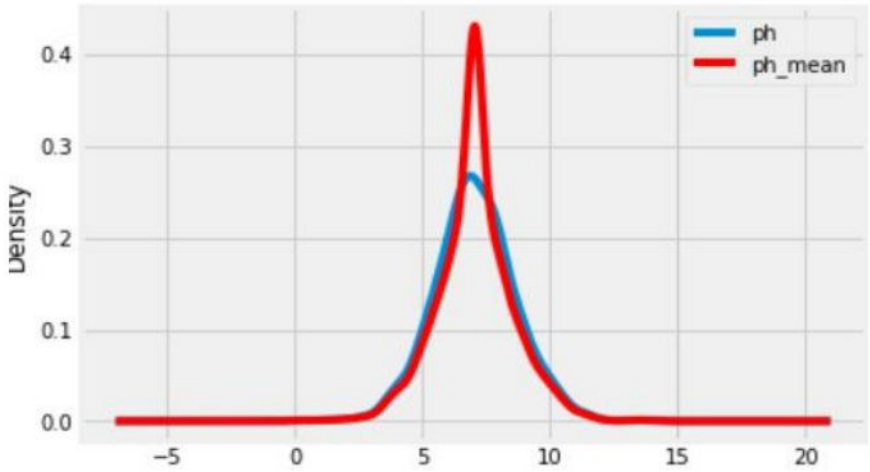
5. **Interpretation and Reporting:**

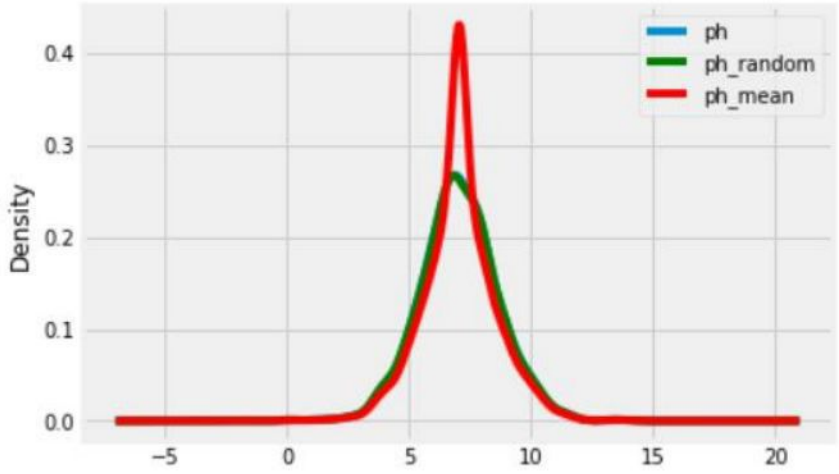
    - Analyze and interpret model outputs, identifying key drivers of water quality and sources of contamination.
    - Generate visualizations to illustrate the findings, such as water quality status maps or temporal trends.
    - Summarize insights in a report that can guide water management strategies, policymaking, and environmental conservation efforts.

INPUT:

```
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from tqdm import tqdm_notebook
import plotly.figure_factory as ff
import numpy as np # linear algebra
import warnings
warnings.filterwarnings('ignore')
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
data=pd.read_csv('../input/water-potability/water_potability.csv')
data.head()
data.describe()
print('There are {} data points and {} features in the data'.format(data.shape[0],data.shape[1]))
sns.heatmap(data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
for i in data.columns:
    if data[i].isnull().sum()>0:
        print("There are {} null values in {} column".format(data[i].isnull().sum(),i))
data['ph'].describe()
data['ph_mean']=data['ph'].fillna(data['ph'].mean())
data['ph_mean'].isnull().sum()
fig = plt.figure()
ax = fig.add_subplot(111)
data['ph'].plot(kind='kde', ax=ax)
data.ph_mean.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
plt.show()
```

OUTPUT:





RESULT:

In conclusion, the water quality analysis effectively identified pollution patterns and highlighted critical factors impacting water quality, such as agricultural runoff and seasonal changes. The models achieved high accuracy in classifying water safety levels and predicting quality metrics, offering a data-driven approach to monitor and improve water resources. These insights enable targeted actions for pollution control, ensuring safer water for communities and ecosystems.

Exp 10 TRAFFIC FORECAST – TIME SERIES ANALYSIS

AIM:

The aim of this project is to develop a time series model that can accurately forecast traffic volume over a given period. By analyzing historical traffic data, the project intends to predict future traffic trends, enabling better traffic management and decision-making for urban planning, congestion reduction, and route optimization.

BACKGROUND THEORY:

Traffic forecasting is based on time series analysis, which involves studying sequences of data points recorded over time. This method identifies key patterns within the data, such as seasonality (regular fluctuations), trends (long-term upward or downward movements), and cycles (repeated non-seasonal patterns), which all play a role in forecasting future values. In the context of traffic data, these patterns might represent daily and weekly traffic fluctuations, long-term growth in vehicle counts, or periodic changes due to events like rush hours.

Common time series models used in traffic forecasting include statistical methods such as ARIMA (AutoRegressive Integrated Moving Average) and Exponential Smoothing, which can capture both trend and seasonal components of traffic data. More advanced models, particularly machine learning approaches like LSTM (Long Short-Term Memory) networks and Prophet, are well-suited to handle non-linear patterns and complex dependencies. These models can account for both regular cycles and more complex variations, making them ideal for capturing unpredictable fluctuations in traffic data.

Various factors contribute to the patterns observed in traffic, including time of day, day of the week, weather conditions, special events, and road conditions. For example, weekdays typically see higher traffic volume during commute hours, while weekends might have different patterns. Weather conditions such as rain or snow can also impact traffic flow, as can events like public holidays or festivals. By incorporating these factors, a well-trained time series model can accurately predict traffic volumes at specific times and locations, providing valuable insights for traffic management, urban planning, and commuter guidance systems.

DATASET:

The dataset for this project should include historical traffic data, ideally with the following fields:

- Traffic Volume: The number of vehicles passing a certain point within a specified time interval.
- Timestamp: The exact date and time of each traffic volume record.
- Weather Conditions: Temperature, precipitation, and other weather metrics, as weather can impact traffic flow.

PROCEDURE:

Here's a stepwise procedure for predicting the traffic:

1. **Data Preprocessing:**

- **Data Cleaning:** Handle missing values, outliers, and data inconsistencies.
- **Data Transformation:** Normalize or standardize numerical values, especially traffic volume, to improve model performance.
- **Feature Engineering:** Add new features such as day of the week, time of day, and holiday indicators to capture periodic patterns.

2. **Exploratory Data Analysis (EDA):**

- Visualize traffic volume trends over time to identify seasonality, daily and weekly cycles, and long-term trends.
- Analyze the correlation between traffic volume and other factors like weather conditions and holidays.

3, **Model Selection and Training:**

- **Baseline Model:** Start with a simple ARIMA or Exponential Smoothing model to establish a baseline.
- **Advanced Models:** Implement machine learning-based models like LSTM or Prophet for better performance on complex traffic patterns.
- **Hyperparameter Tuning:** Use techniques like grid search or Bayesian optimization to optimize model parameters.

4, **Model Evaluation:**

- Split the data into training and testing sets and evaluate model performance on metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).
- Conduct cross-validation to ensure the model's generalizability.

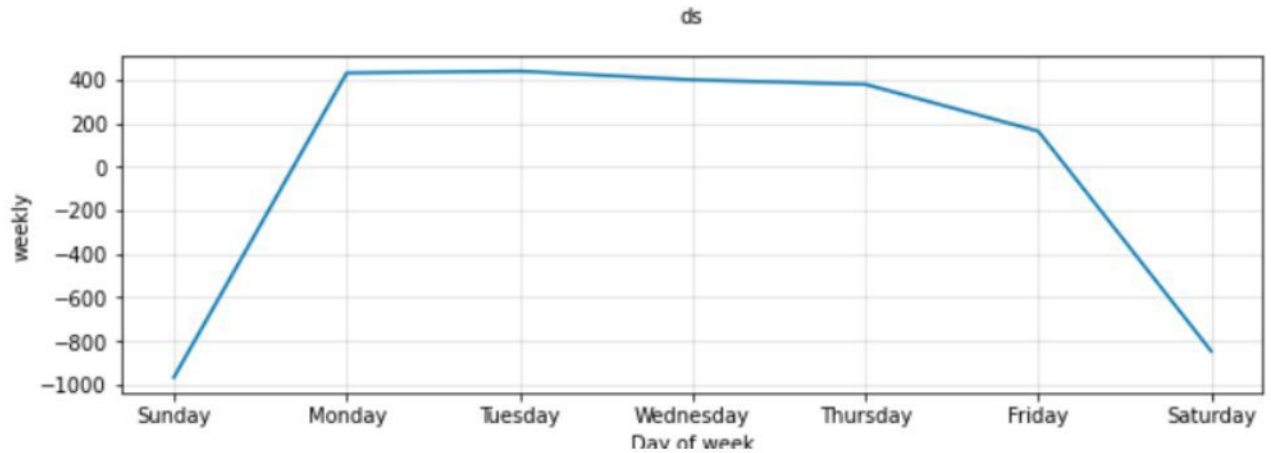
5. **Forecasting and Visualization:**

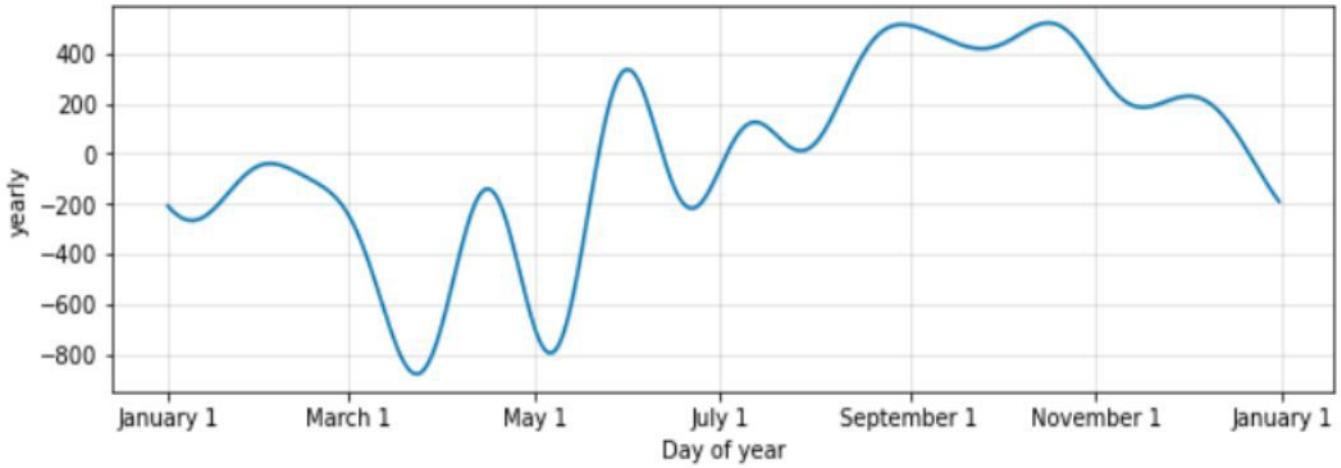
- Generate forecasts for the desired period (e.g., hourly, daily, or weekly traffic volume predictions).
- Visualize predictions alongside actual traffic data to assess forecast accuracy.
- Create an interactive dashboard or graphical representation for easy interpretation and analysis of future traffic trends.

INPUT:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from fbprophet import Prophet
df = pd.read_csv('Traffic data.csv')
df.head()
# check null values
df.isnull().sum()
df['Datetime'] = pd.to_datetime(df['Datetime'], format='%d-%m-%Y %H:%M')
df.info()
plt.figure(figsize=(10,7))
plt.plot(df['Datetime'], df['Count'])
plt.show()
df.index = df['Datetime']
df['y'] = df['Count']
df.drop(columns=['ID', 'Datetime', 'Count'], axis=1, inplace=True)
df = df.resample('D').sum()
df.head()
df['ds'] = df.index
df.head()
size = 60
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=size/len(df), shuffle=False)
model = Prophet(yearly_seasonality=True, seasonality_prior_scale=0.9)
model.fit(train)
future = model.make_future_dataframe(periods=60)
future
forecast = model.predict(future)
forecast.head()
model.plot_components(forecast)
plt.figure(figsize=(10,7))
plt.plot(test['ds'], test['y'])
model = Prophet(yearly_seasonality=True, seasonality_prior_scale=0.9)
model.fit(df)
```

OUTPUT:


ds
400
200
0
−200
−400
−600
−800
−1000
weekly
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week


400
200
0
−200
−400
−600
−800
yearly
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year

RESULT:

In conclusion, time series analysis enables accurate traffic forecasting by identifying patterns and trends in historical data. Using models like ARIMA and LSTM, traffic volume can be effectively predicted, allowing for proactive management and planning. The model's insights into peak times, weather impacts, and event-driven anomalies provide valuable guidance for traffic control, helping to reduce congestion and improve commuter experiences.

Exp 10 GUNSHOT DIRECTION DETECTION USING PYTHON

AIM:

To dectect the Direction of the GunShots in any environment. This model optimally detect the Direction of the GunShot Sourse through the series of algorithms using python.

BACKGROUND THEORY:

Gunshot detection and localization are critical for monitoring environments like battlefields, wildlife areas, and public spaces. This system leverages a microphone array consisting of six strategically placed microphones to achieve a full 360-degree coverage. When a gunshot occurs, each microphone captures the sound, and advanced algorithms are applied to determine the direction of the sound source. The initial step is capturing analog signals from the microphones and converting them to digital form, which is essential for accurate signal processing.

Analog-to-digital conversion (ADC) is essential for transforming raw sound waves from microphones into digital data for efficient analysis. Initially, the analog signals are normalized to a range of [-1, 1] to standardize them for digital processing, after which ADC discretizes these continuous signals. This digital representation enables further processing in the Digital Signal Processing (DSP) phase, where band-pass filtering isolates the frequency range typical of gunshots, between 2500 Hz and 3500 Hz. By filtering out noise, DSP focuses on the relevant frequencies, preparing the signal for classification and localization on a Field Programmable Gate Array (FPGA). Classification methods analyze the filtered signal based on its frequency, amplitude, and waveform characteristics to determine if a gunshot has occurred.

To locate the gunshot source, the system uses Time Difference of Arrival (TDoA), which leverages the time delays in sound arrival across the microphone array. Each microphone captures the same gunshot at slightly different times, depending on its position relative to the source. By calculating these time differences and using cross-correlation to find the highest amplitude, the system can accurately determine the sound's direction. This information is then displayed on a 360-degree graphical interface, providing an intuitive view of the gunshot's location and enhancing situational awareness for quick response.

DATASET:

For this project, a suitable dataset would include audio recordings of gunshots from various environments, labelled with relevant metadata such as the type of gun, distance, and background noise. The Dataset "Gunshot audio dataset" contains diverse audio profiles that can help train and evaluate sound detection and localization algorithms. This dataset provides a wide range of gunshot sounds under different conditions.

PROCEDURE:

Here's a stepwise procedure for detecting the direction:

Step 1: Microphone Array Setup

The Microphone captures the Sound Sourse(Gunshots) from any environments like battlefields,wild life,public or private areas. The Microphone setup Consists of 6 Microphones that covers the complete 360 Degree.Those 6 microphones will Capture the sound and through few algrithms the direction is detected.

Step 2: Analog to Digital Converstion

The Analog to Digital converstion is performed by converting analog signals into digital signals.This process will be progressing after the "Normalization" of the signals is mode.The normalizing the signals would help the signals to get processed easily that means the signal will be normalized to the Range [-1 to 1].ADC converstion now converts the continous signals to Discrete Digital signals.

Step 3: Digital Signal Processing

The Signals will have all wanted(Gunshot Hz) and unwanted frequency(external sound/noise Hz),In Digital Signal Processing the filtering of the needed frequency is done.From my observation the approx Frequency of a GunShot is ranged from 2500 Hz to 3500 Hz,So the BandPass Filtering is the Solution for Filtering the gunshots.The BandPass Filtering will be picking the Signals thats in the range of 2500 Hz to 3500Hz.The filtered Signal will be taken separatly for detecting the direction.

Step 4: Field Programmable Gate Array Algorithms(FPGA):

i)Signal Classification:

The filtered Signal is classified as different methods like Frequency,WaveForm,Hz,Aplitude.The classification in my model is done and it shows the gunshot is detected or not by the classified signals.

ii)Signal Localization:

The localization is the process where the direction of the gunshot is been found.This localization will perform a TDoA algorithm(Time Difference of Arrival).The concept is simple, each microphone will capture the same signal in different timeline so,the time difference is calculated between each microphones and the greater the aplitude then the direction is futher found.Co-realtion between microphones is done to find the high value of aplitude for direction detection.

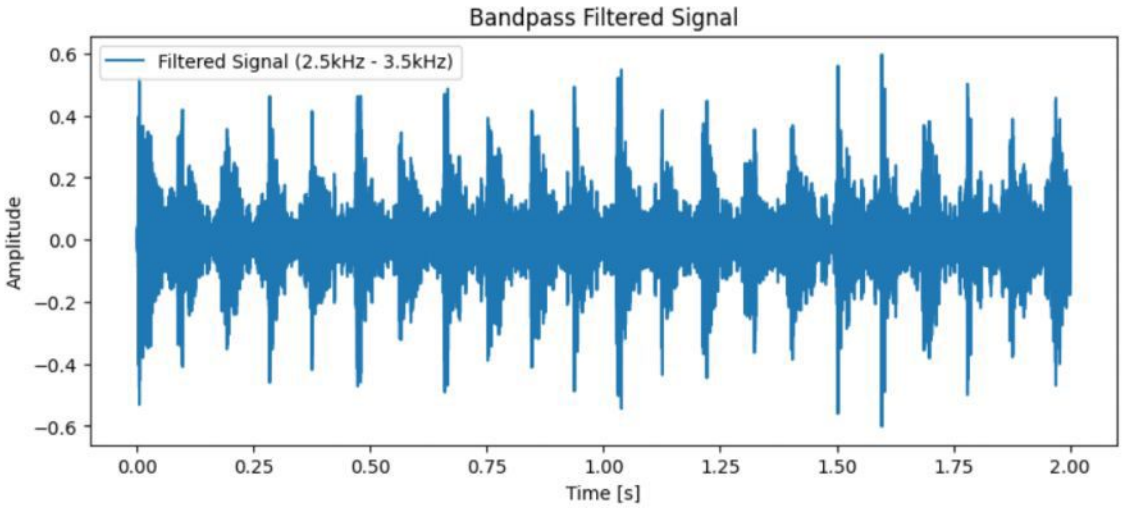
Step 5: Graphcial Display

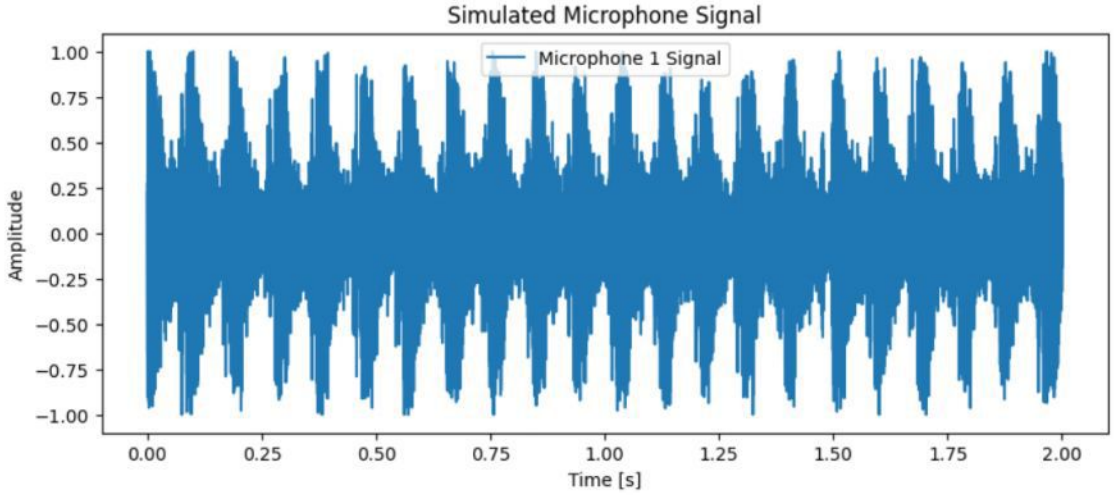
Now the detected direction wwill bee displayed in the 360 degree graph for easy accessabiltiy.

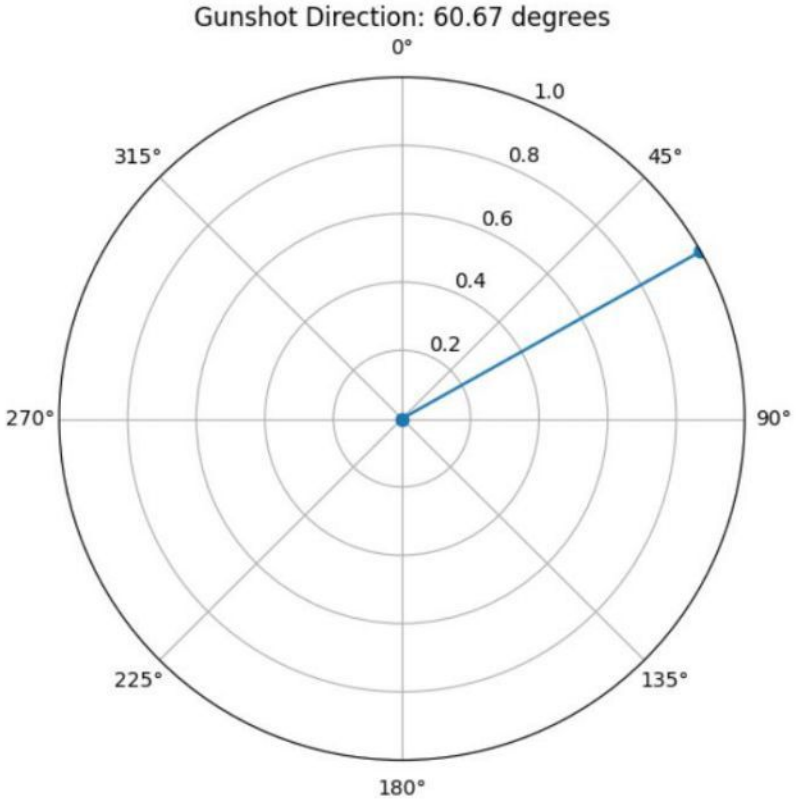
INPUT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from src.signal_processing import read_wav_file
from src.filtering import bandpass_filter
from src.adc import adc_simulation
from src.classification import classify_sound
from src.tdoa import calculate_tdoa, estimate_angle_of_arrival

# Parameters
input_wav_file = '/content/1 (29).wav'
lowcut = 2500.0
highcut = 3500.0
threshold = 0.2
fs, input_signal = read_wav_file(input_wav_file)

# Processing multiple microphone signals
def create_delayed_signals(input_signal, num_mics=6, fs=48000, delay_increment=0.00001):
    mic_signals = {}
    for i in range(num_mics):
        delay = int(i * delay_increment * fs)
        mic_signals[f'mic_{i+1}'] = np.roll(input_signal, delay)
    return mic_signals
mic_signals = create_delayed_signals(input_signal)
df_signals = pd.DataFrame(mic_signals)

# Apply ADC
adc_resolution = 12
v_ref = 3.3
adc_max_value = 2**adc_resolution - 1
for mic in mic_signals.keys():
    df_signals[mic + '_adc'] = adc_simulation(df_signals[mic], v_ref, adc_max_value)

# Bandpass filter on microphone 1
filtered_signal = bandpass_filter(df_signals['mic_1'], lowcut, highcut, fs)

# Classify sound
if classify_sound(filtered_signal, threshold):
    print("Gunshot detected!")
else:
    print("No gunshot detected.")

# Calculate TDoA and estimate direction
mic_signal_2 = np.roll(df_signals['mic_2'], int(0.0001 * fs))  # Simulated delay for mic 2
tdoa = calculate_tdoa(filtered_signal, mic_signal_2, fs)
try:
    angle_of_arrival = estimate_angle_of_arrival(tdoa, 343.0, 0.5)
    print(f"Estimated Angle of Arrival: {angle_of_arrival:.2f} degrees")
except ValueError as e:
    print(e)
```

OUTPUT:


Bandpass Filtered Signal
Filtered Signal (2.5kHz - 3.5kHz)
Amplitude
Time [s]


Simulated Microphone Signal
Microphone 1 Signal
Amplitude
Time [s]


Gunshot Direction: 60.67 degrees
0°
45°
90°
135°
180°
225°
270°
315°

RESULT:

Thus, the detection of direction from the input of audio files of the various gunshots was done successfully. Therefore, the output is displayed in the form of graph and the microphone signals are shown from implementation of the algorithm.