Required Libraries to Install

sh !pip install pandas !pip install matplotlib !pip install plotly !pip install seaborn !pip install statsmodels !pip install numpy !pip install scikit-learn !pip install!pip install transformers torch yfinance

```
!pip install pandas
!pip install matplotlib
!pip install plotly
!pip install seaborn
!pip install statsmodels
!pip install numpy
!pip install scikit-learn
!pip install transformers torch yfinance
```

```
Requirement already satisfied: tzdata>=2022.7 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from p
Requirement already satisfied: six>=1.5 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from python-dateutil>=2
Requirement already satisfied: statsmodels in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fr
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (f
Requirement already satisfied: patsy>=0.5.6 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from sta
Requirement already satisfied: packaging>=21.3 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from statsmodels
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from pand
Requirement already satisfied: pytz>=2020.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from par
Requirement already satisfied: tzdata>=2022.7 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from p
Requirement already satisfied: six>=1.5 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from python-dateutil>=2
Requirement already satisfied: numpy in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (2.2.1)
Requirement already satisfied: scikit-learn in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (1.6.0)
Requirement already satisfied: numpy>=1.19.5 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from sc
Requirement already satisfied: scipy>=1.6.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from sci
Requirement already satisfied: joblib>=1.2.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from sc
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages
Requirement already satisfied: transformers in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (4.47.1)
Requirement already satisfied: torch in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (2.5.1)
Requirement already satisfied: yfinance in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (0.2.51)
Requirement already satisfied: filelock in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from transfo
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-pa
Requirement already satisfied: numpy>=1.17 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from tran
Requirement already satisfied: packaging>=20.0 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from transformer
Requirement already satisfied: pyyaml>=5.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from tran
Requirement already satisfied: regex!=2019.12.17 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fro
Requirement already satisfied: requests in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from transfo
Requirement already satisfied: tokenizers<0.22,>=0.21 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages
Requirement already satisfied: safetensors>=0.4.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fr
Requirement already satisfied: tqdm>=4.27 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from trans
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from to
Requirement already satisfied: networkx in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from torch)
Requirement already satisfied: jinja2 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from torch) (3
Requirement already satisfied: fsspec in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from torch) (2
Requirement already satisfied: sympy==1.13.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from to
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fr
Requirement already satisfied: pandas>=1.3.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from yf
Requirement already satisfied: multitasking>=0.0.7 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (f
Requirement already satisfied: lxml>=4.9.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from yfir
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from yfinanc
Requirement already satisfied: pytz>=2022.5 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from yf
Requirement already satisfied: frozendict>=2.3.4 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fro
Requirement already satisfied: peewee>=3.16.2 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from y
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages
Requirement already satisfied: html5lib>=1.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from yf
Requirement already satisfied: soupsieve>1.2 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from be
Requirement already satisfied: six>=1.9 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from html5lib>=1.1->yf
Requirement already satisfied: webencodings in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from htm
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sruth\appdata\roaming\python\python311\site-packages (from pand
Requirement already satisfied: tzdata>=2022.7 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from p
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packag
Requirement already satisfied: idna<4,>=2.5 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from req
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fr
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (fr
Requirement already satisfied: colorama in c:\users\sruth\appdata\roaming\python\python311\site-packages (from tqdm>=4.27->transf
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sruth\appdata\local\programs\python\python311\lib\site-packages (from
```

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from statsmodels.tsa.seasonal import STL
import plotly.express as px
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import yfinance as yf
from datetime import datetime, timedelta
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import os
```

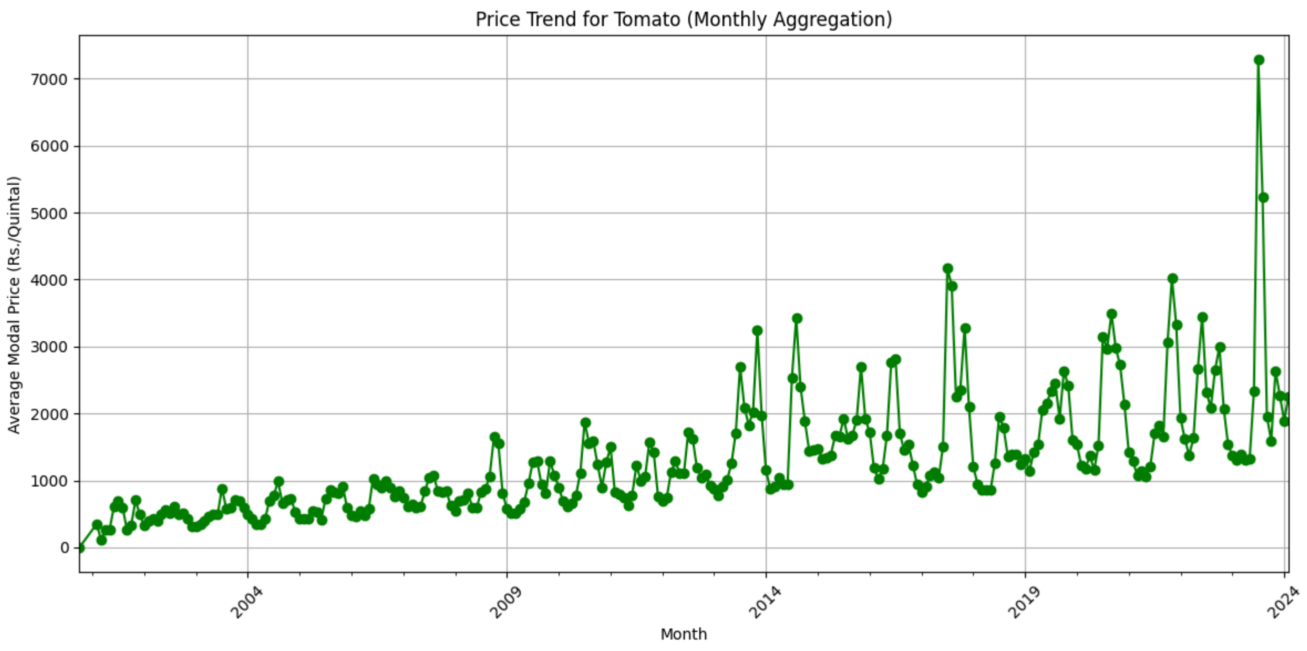
Enter the Commodity Name by checking the name of csv.

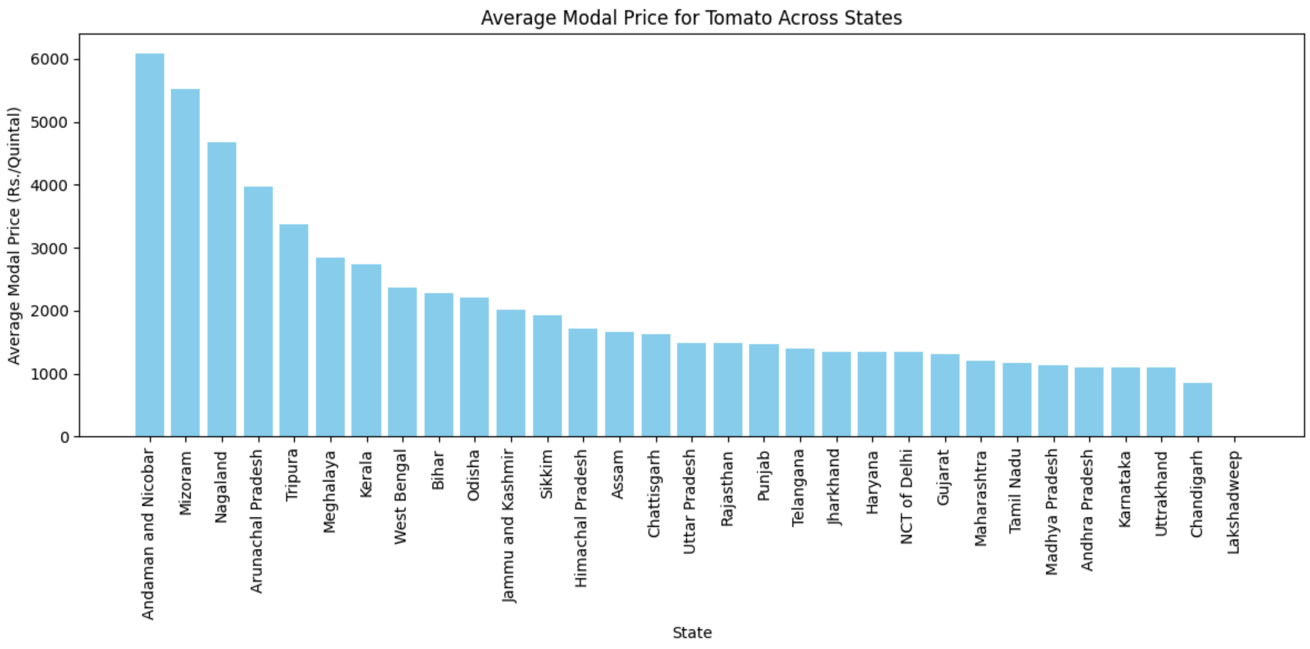
```
Crop_Name='Tomato'
```

Visualizing Commodity Price Data: Trends, Market Distribution, and State Analysis

```
# Step 1: Function to Visualize Data for a Given Commodity
def visualize_commodity_data(folder_path, commodity_name):
    """
    Plots a graphical representation of the data for the given commodity.
    - Line graph of modal prices over time (monthly aggregation).
    - Pie chart of market price distribution for top N markets.
    - Bar graph of average modal price across states.
    """
    # Construct the file path for the commodity
    file_path = os.path.join(folder_path, f"{commodity_name}.csv")

    if not os.path.exists(file_path):
        print(f"Dataset for {commodity_name} not found in {folder_path}")
        return

    # Load the dataset
    df = pd.read_csv(file_path)

    # Ensure the reported_date column is in datetime format
    df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')

    # Drop rows with invalid or missing dates
    df = df.dropna(subset=['reported_date'])

    # Step 2: Aggregated Price Trend Over Time (Monthly Aggregation)
    df['month'] = df['reported_date'].dt.to_period('M')
    monthly_prices = df.groupby('month')['modal_price_(rs./quintal)'].mean()

    # Plot the aggregated price trend
    plt.figure(figsize=(12, 6))
    monthly_prices.plot(kind='line', marker='o', color='green')
    plt.title(f'Price Trend for {commodity_name} (Monthly Aggregation)')
    plt.xlabel('Month')
    plt.ylabel('Average Modal Price (Rs./Quintal)')
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    # Step 3: Interactive Pie Chart with Slicer for Top N Markets
    market_avg_price = df.groupby('market_name')['modal_price_(rs./quintal)'].mean().sort_values(ascending=False)

    # Function to update pie chart based on top N markets
    def create_pie_chart(top_n):
        top_markets = market_avg_price.head(top_n)
        fig = px.pie(values=top_markets.values, names=top_markets.index,
                     title=f'Market Price Distribution for {commodity_name} (Top {top_n} Markets)', hole=0.3)
        fig.update_traces(textinfo='percent+label', pull=[0.1]*len(top_markets))
        fig.show()

    # Create Pie Chart with a dynamic slicer
    for top_n in [5, 10, 15, 20]:  # Adjust the range as needed
        create_pie_chart(top_n)

    # Step 4: Average Modal Price Across States (Bar Chart)
    plt.figure(figsize=(12, 6))
    state_avg_price = df.groupby('state_name')['modal_price_(rs./quintal)'].mean().sort_values(ascending=False)

    # Create the bar chart
    plt.bar(state_avg_price.index, state_avg_price.values, color='skyblue')

    # Title and labels
    plt.title(f'Average Modal Price for {commodity_name} Across States')
    plt.xlabel('State')
```

```
        plt.ylabel('Average Modal Price (Rs./Quintal)')

        # Rotate x-axis labels for better readability
        plt.xticks(rotation=90, ha='center')

        # Adjust layout to avoid label overlap
        plt.tight_layout()

        # Show the plot
        plt.show()

    # Specify the folder containing cleaned datasets
    cleaned_folder = './Cleaned_Datasets'

    # Input commodity name
    commodity = Crop_Name  # Replace with the desired commodity name

    # Visualize the data
    visualize_commodity_data(cleaned_folder, commodity)
```

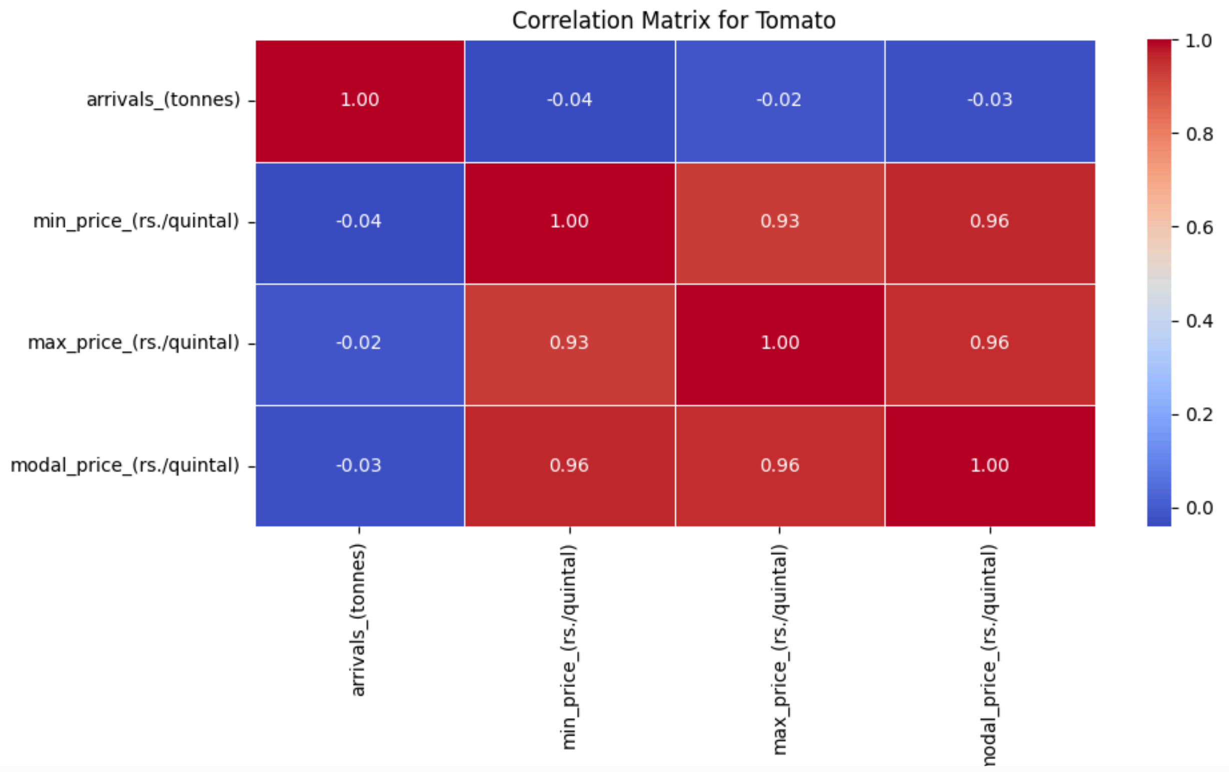

Price Trend for Tomato (Monthly Aggregation)
Average Modal Price (Rs./Quintal)
0
1000
2000
3000
4000
5000
6000
7000
Month
2004
2009
2014
2019
2024



Average Modal Price for Tomato Across States
Average Modal Price (Rs./Quintal)
State
0
1000
2000
3000
4000
5000
6000
Andaman and Nicobar
Mizoram
Nagaland
Arunachal Pradesh
Tripura
Meghalaya
Kerala
West Bengal
Bihar
Odisha
Jammu and Kashmir
Sikkim
Himachal Pradesh
Assam
Chattisgarh
Uttar Pradesh
Rajasthan
Punjab
Telangana
Jharkhand
Haryana
NCT of Delhi
Gujarat
Maharashtra
Tamil Nadu
Madhya Pradesh
Andhra Pradesh
Karnataka
Uttrakhand
Chandigarh
Lakshadweep

Loading Data and Performing Exploratory Data Analysis (EDA) for Commodity Prices

```
# Step 1: Load and Prepare the Data
def load_data(folder_path, commodity_name):
    """
    Load the dataset for a given commodity.
    """
    file_path = os.path.join(folder_path, f"{commodity_name}.csv")
    if not os.path.exists(file_path):
        print(f"Dataset for {commodity_name} not found in {folder_path}")
        return None

    # Load the dataset
    df = pd.read_csv(file_path)

    # Ensure date is in datetime format
    df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')

    # Drop rows with invalid or missing dates
    df = df.dropna(subset=['reported_date'])

    return df

# Step 2: Exploratory Data Analysis (EDA)
def perform_eda(df, commodity_name):
    """
    Perform EDA including price trends, seasonality, price distribution, and correlation analysis.
    """
    # Ensure modal price column is numeric
    df['modal_price_(rs./quintal)'] = pd.to_numeric(df['modal_price_(rs./quintal)'], errors='coerce')

    # 2.1 Visualize Price Trends
    df['month'] = df['reported_date'].dt.to_period('M')  # Group by month
    monthly_prices = df.groupby('month')['modal_price_(rs./quintal)'].mean()

    # 2.3 Identify Correlations
    # Correlation Matrix
    if 'arrivals_(tonnes)' in df.columns and 'min_price_(rs./quintal)' in df.columns:
        correlation_columns = ['arrivals_(tonnes)', 'min_price_(rs./quintal)', 'max_price_(rs./quintal)', 'modal_price_(rs./quintal)']
        correlation_df = df[correlation_columns].corr()

        plt.figure(figsize=(10, 6))
        sns.heatmap(correlation_df, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
        plt.title(f'Correlation Matrix for {commodity_name}')
        plt.tight_layout()
        plt.show()

# Main Function
def main():
    # Specify the folder containing cleaned datasets
    cleaned_folder = './Cleaned_Datasets'

    # Input commodity name
    commodity = Crop_Name  # Replace with desired commodity name

    # Load the data
    df = load_data(cleaned_folder, commodity)
    if df is not None:
        perform_eda(df, commodity)

# Run the main function
if __name__ == "__main__":
    main()
```

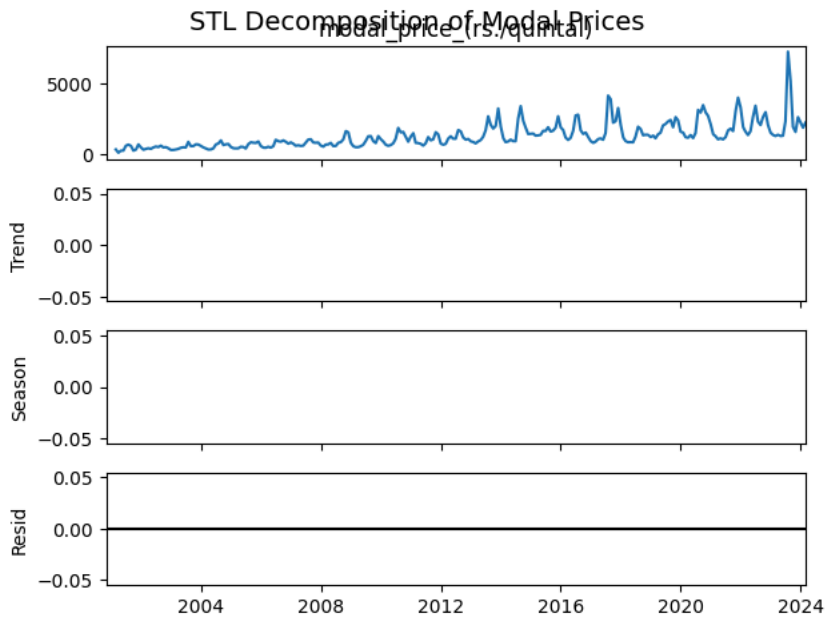


Analyzing Seasonal Trends, Volatility, and Decomposition of Commodity Prices

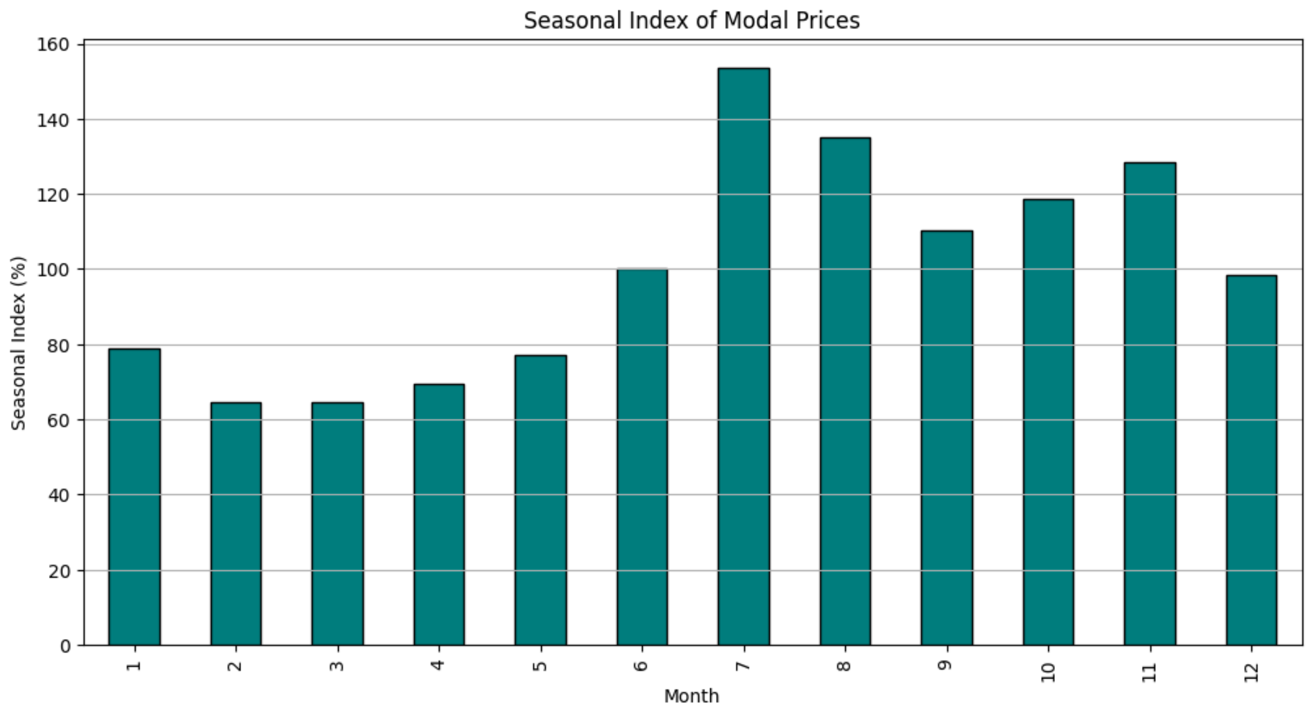
```
# Step 1: Load the Data
def load_data(folder_path, commodity_name):
    file_path = os.path.join(folder_path, f"{commodity_name}.csv")
    if not os.path.exists(file_path):
        print(f"Dataset for {commodity_name} not found in {folder_path}")
        return None
    df = pd.read_csv(file_path)
    df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')
    df.dropna(subset=['reported_date'], inplace=True)
    return df

# Step 2: STL Decomposition
def perform_stl_decomposition(df, column='modal_price_(rs./quintal)'):
    df = df.set_index('reported_date').sort_index()
    resampled_data = df[column].resample('ME').mean()  # Use 'ME' for monthly end
    stl = STL(resampled_data, seasonal=13)
    result = stl.fit()

    # Plot the components
    result.plot()
    plt.suptitle('STL Decomposition of Modal Prices', fontsize=14)
    plt.show()

    return result

# Step 3.1: Seasonal Index Calculation
def calculate_seasonal_index(df, column='modal_price_(rs./quintal)'):
    df['month'] = df['reported_date'].dt.month
    monthly_avg = df.groupby('month')[column].mean()
    yearly_avg = df[column].mean()
    seasonal_index = (monthly_avg / yearly_avg) * 100

    # Plot Seasonal Index
    plt.figure(figsize=(12, 6))
    seasonal_index.plot(kind='bar', color='teal', edgecolor='black')
    plt.title('Seasonal Index of Modal Prices')
    plt.xlabel('Month')
    plt.ylabel('Seasonal Index (%)')
    plt.grid(axis='y')
    plt.show()

    return seasonal_index

# Step 3.2: Interactive Heatmap
def create_interactive_heatmap(df, column='modal_price_(rs./quintal)'):
    df['year'] = df['reported_date'].dt.year
```

```
        df['month'] = df['reported_date'].dt.month
        pivot_table = df.pivot_table(values=column, index='year', columns='month', aggfunc='mean')

        # Plot interactive heatmap
        fig = px.imshow(pivot_table, labels=dict(x="Month", y="Year", color="Price"),
                        x=[f"Month {i}" for i in range(1, 13)], y=pivot_table.index,
                        title="Seasonal Heatmap of Modal Prices")
        fig.update_layout(coloraxis_colorbar=dict(title="Modal Price (Rs./Quintal)"))
        fig.show()

    # Step 3.3: Analyze Seasonal Trends and Volatility
    def analyze_seasonal_trends_and_volatility(df, commodity_name):
        df['modal_price_(rs./quintal)'] = pd.to_numeric(df['modal_price_(rs./quintal)'], errors='coerce')
        df = df.dropna(subset=['modal_price_(rs./quintal)'])

        # Add Month and Season Columns
        df['month'] = df['reported_date'].dt.month
        df['season'] = df['month'].apply(lambda x: 'Winter' if x in [12, 1, 2] \
                                         else 'Spring' if x in [3, 4, 5] \
                                         else 'Summer' if x in [6, 7, 8] \
                                         else 'Autumn')

        # Boxplot by Season (Fix: Pass 'month' for hue)
        plt.figure(figsize=(12, 6))
        sns.boxplot(x='season', y='modal_price_(rs./quintal)', data=df, palette='Set2', hue='season')
        plt.title(f'Price Distribution Across Seasons for {commodity_name}')
        plt.xlabel('Season')
        plt.ylabel('Modal Price (Rs./Quintal)')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

        # Identify Peaks and Lows
        monthly_avg_prices = df.groupby('month')['modal_price_(rs./quintal)'].mean()
        print("\n### Peaks and Lows ###")
        print(f"Highest Average Price: {monthly_avg_prices.idxmax()} (Month), Price: {monthly_avg_prices.max()}")
        print(f"Lowest Average Price: {monthly_avg_prices.idxmin()} (Month), Price: {monthly_avg_prices.min()}")

        # Volatility as Standard Deviation
        monthly_volatility = df.groupby('month')['modal_price_(rs./quintal)'].std()

        plt.figure(figsize=(12, 6))
        monthly_volatility.plot(kind='bar', color='coral')
        plt.title(f'Monthly Price Volatility for {commodity_name}')
        plt.xlabel('Month')
        plt.ylabel('Standard Deviation of Modal Price')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

        # Rolling Standard Deviation (Volatility)
        rolling_volatility = df.set_index('reported_date')['modal_price_(rs./quintal)'].rolling(window=30).std()

        plt.figure(figsize=(12, 6))
        rolling_volatility.plot(color='purple', linewidth=2)
        plt.title(f'Rolling Volatility (30-Day Window) for {commodity_name}')
        plt.xlabel('Date')
        plt.ylabel('Rolling Standard Deviation')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    # Main Function
    def main():
        # Specify the folder containing cleaned datasets
        cleaned_folder = './Cleaned_Datasets'

        # Input commodity name
        commodity = Crop_Name  # Replace with desired commodity name

        # Load the data
        df = load_data(cleaned_folder, commodity)
        if df is not None:
            # STL Decomposition
            print("Performing STL Decomposition...")
            perform_stl_decomposition(df)

            # Seasonal Index Calculation
            print("Calculating Seasonal Index...")
            calculate_seasonal_index(df)

            # Interactive Heatmap
            print("Creating Interactive Heatmap...")
            create_interactive_heatmap(df)

            # Seasonal Trends and Volatility Analysis
            print("Analyzing Seasonal Trends and Volatility...")
            analyze_seasonal_trends_and_volatility(df, commodity)

    # Run the main function
    if __name__ == "__main__":
        main()
```

```
Performing STL Decomposition...
```

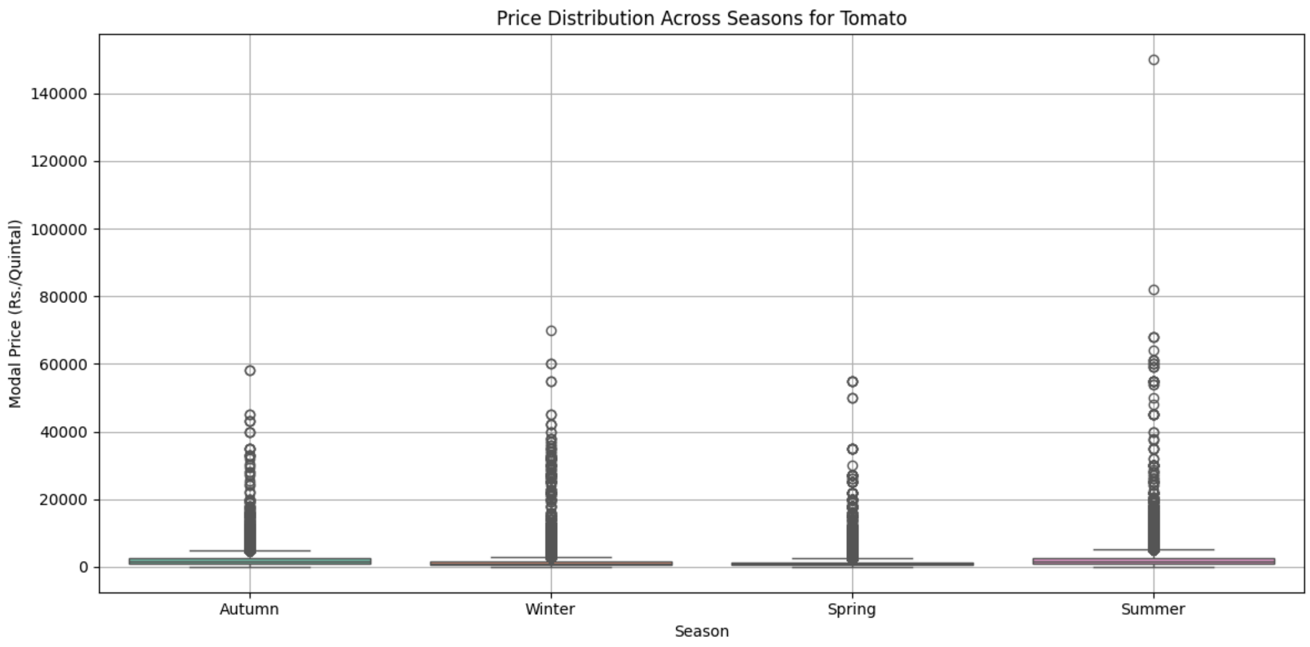


```
Calculating Seasonal Index...
```

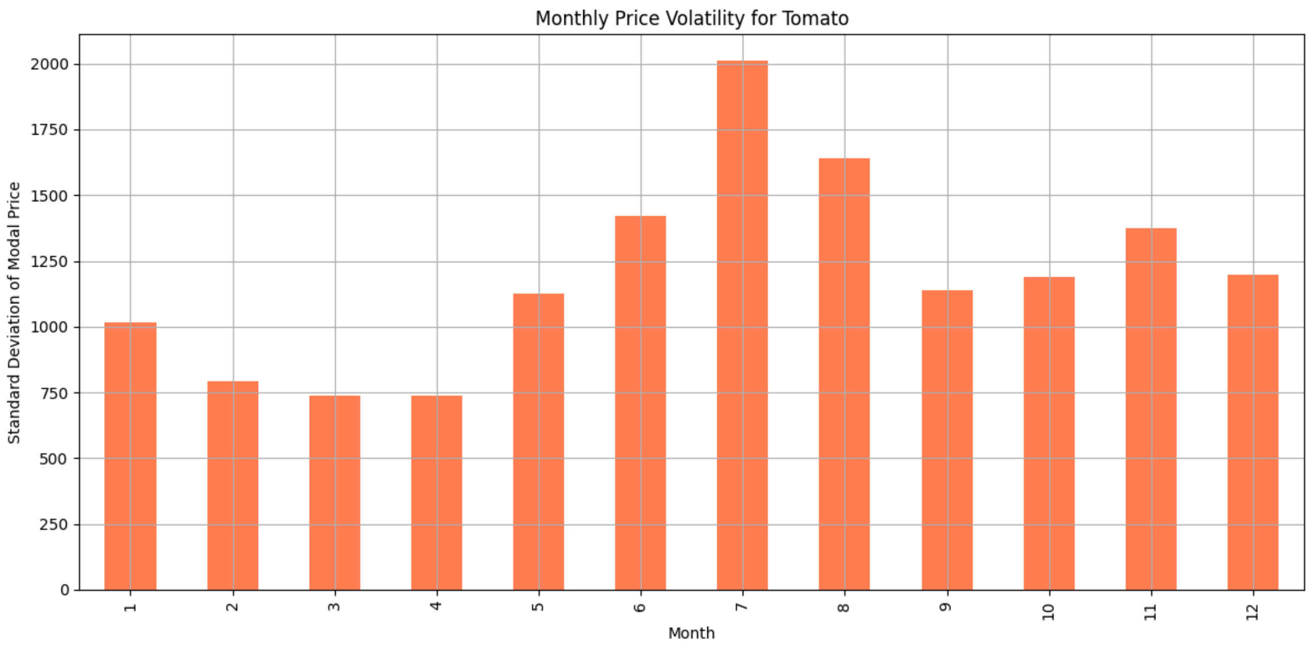


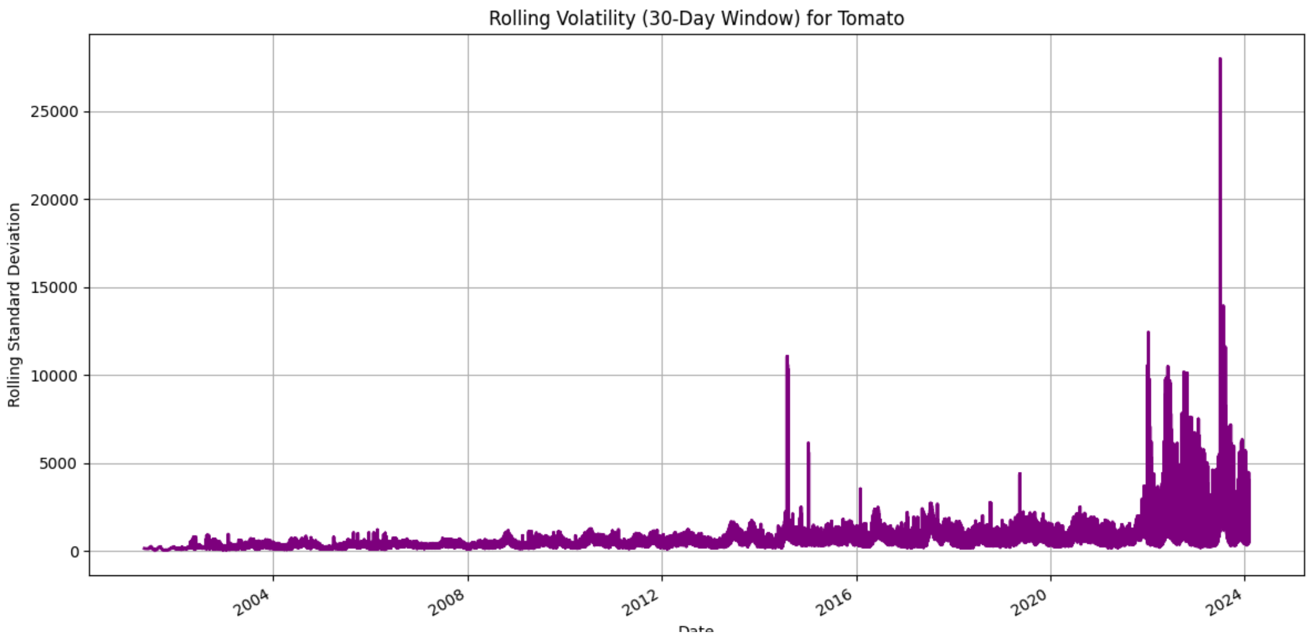
```
Creating Interactive Heatmap...
```

```
Analyzing Seasonal Trends and Volatility...
```



```
### Peaks and Lows ###
Highest Average Price: 7 (Month), Price: 2481.2884674549537
Lowest Average Price: 2 (Month), Price: 1040.4785877390418
```



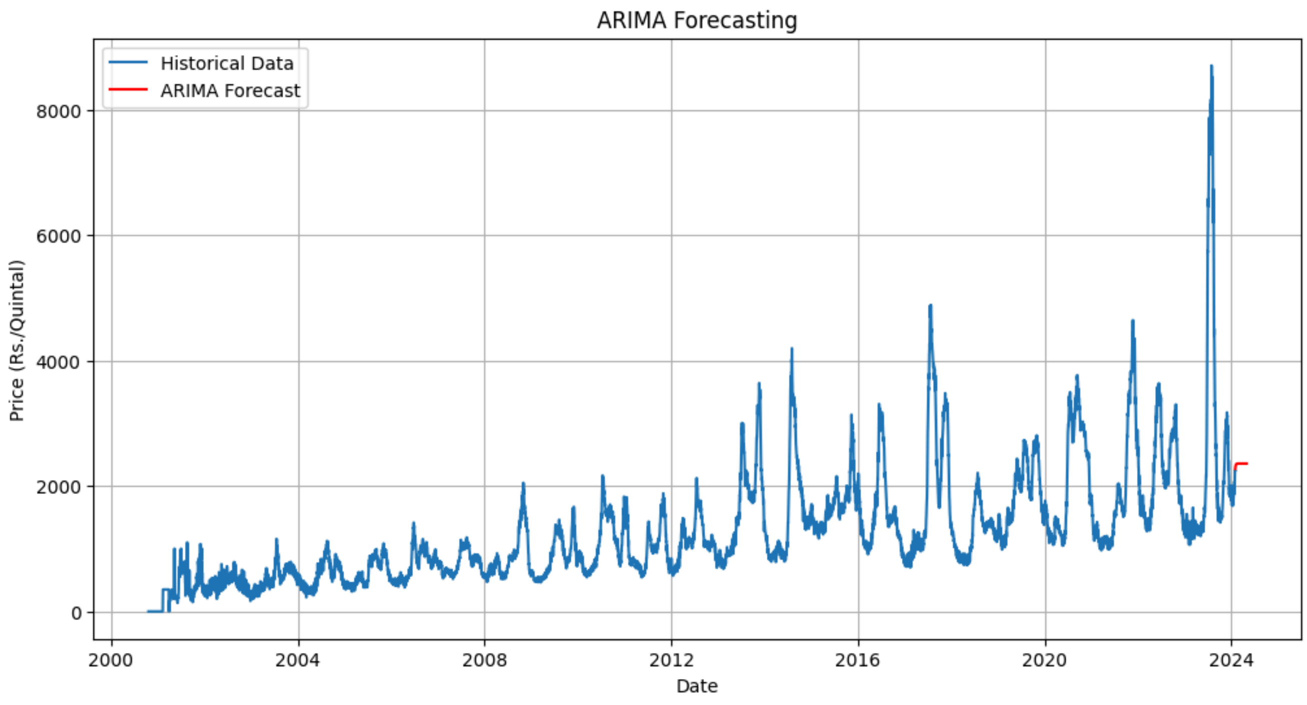


Date

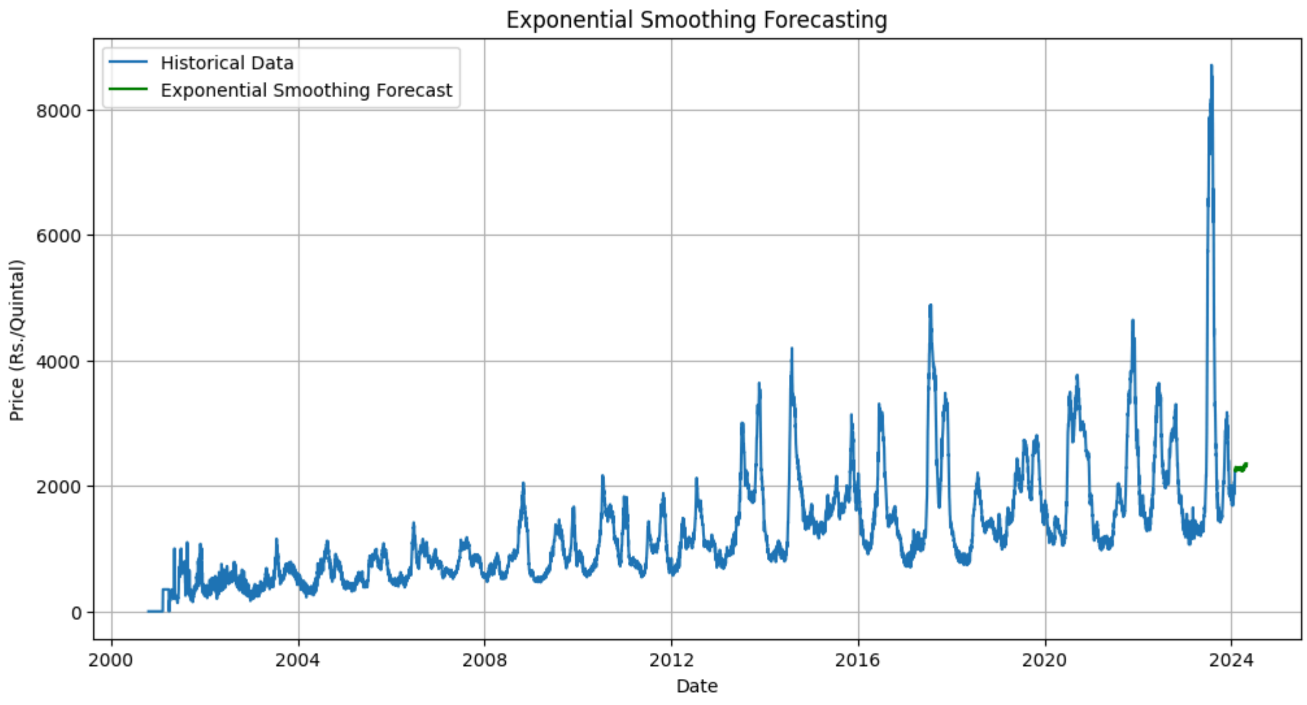
Price Forecasting using ARIMA, Exponential Smoothing, and Random Forest

```python
# ARIMA Forecasting
def arima_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    """
    Perform ARIMA forecasting and plot results.
    """
    df = df.set_index('reported_date').sort_index()
    time_series = df[column].resample('D').mean().ffill()  # Updated line to avoid FutureWarning

    # Train ARIMA model
    model = ARIMA(time_series, order=(5, 1, 0))  # Adjust order based on dataset
    model_fit = model.fit()

    # Forecast for specified periods
    forecast = model_fit.forecast(steps=forecast_periods)
    forecast_index = pd.date_range(start=time_series.index[-1] + pd.Timedelta(days=1), periods=forecast_periods, freq='D')

    # Plot results
    plt.figure(figsize=(12, 6))
    plt.plot(time_series, label='Historical Data')
    plt.plot(forecast_index, forecast, label='ARIMA Forecast', color='red')
    plt.title('ARIMA Forecasting')
    plt.xlabel('Date')
    plt.ylabel('Price (Rs./Quintal)')
    plt.legend()
    plt.grid()
    plt.show()

    return forecast

# Exponential Smoothing Forecasting
def exponential_smoothing_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    """
    Perform Exponential Smoothing forecasting and plot results.
    """
    df = df.set_index('reported_date').sort_index()
    time_series = df[column].resample('D').mean().ffill()  # Updated line to avoid FutureWarning

    # Train Exponential Smoothing model
    model = ExponentialSmoothing(time_series, seasonal='add', seasonal_periods=365)
    model_fit = model.fit()

    # Forecast for specified periods
    forecast = model_fit.forecast(forecast_periods)
    forecast_index = pd.date_range(start=time_series.index[-1] + pd.Timedelta(days=1), periods=forecast_periods, freq='D')

    # Plot results
    plt.figure(figsize=(12, 6))
    plt.plot(time_series, label='Historical Data')
    plt.plot(forecast_index, forecast, label='Exponential Smoothing Forecast', color='green')
    plt.title('Exponential Smoothing Forecasting')
    plt.xlabel('Date')
    plt.ylabel('Price (Rs./Quintal)')
    plt.legend()
    plt.grid()
    plt.show()

    return forecast

def random_forest_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    """
    Perform Random Forest forecasting and plot results.
    """
    # Feature engineering for ML model
    df['year'] = df['reported_date'].dt.year
    df['month'] = df['reported_date'].dt.month
    df['day'] = df['reported_date'].dt.day
    df['day_of_week'] = df['reported_date'].dt.dayofweek

    # Define features and target
    X = df[['year', 'month', 'day', 'day_of_week']]
    y = df[column]

    # Split data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train Random Forest model with optimizations
    model = RandomForestRegressor(n_estimators=20, max_depth=10, n_jobs=-1, random_state=42)  # Reduced estimators and max_depth
    model.fit(X_train, y_train)
    # Predict on test set
    y_pred = model.predict(X_test)
    print(f"Random Forest R2 Score: {r2_score(y_test, y_pred):.2f}")
    print(f"Random Forest RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")

    # Generate forecast data for the next periods
    last_date = df['reported_date'].max()
    forecast_dates = [last_date + pd.Timedelta(days=i) for i in range(1, forecast_periods + 1)]
    forecast_features = pd.DataFrame({
        'year': [date.year for date in forecast_dates],
        'month': [date.month for date in forecast_dates],
        'day': [date.day for date in forecast_dates],
        'day_of_week': [date.dayofweek for date in forecast_dates],
    })
    forecast = model.predict(forecast_features)

    # Plot results
    plt.figure(figsize=(12, 6))
    plt.plot(df['reported_date'], df[column], label='Historical Data')
    plt.plot(forecast_dates, forecast, label='Random Forest Forecast', color='orange')
    plt.title('Random Forest Forecasting')
    plt.xlabel('Date')
    plt.ylabel('Price (Rs./Quintal)')
    plt.legend()
    plt.grid()
    plt.show()

    return forecast


# Main Forecasting Function
def price_forecasting(df, forecast_periods=90):
    """
    Perform price forecasting using ARIMA, Exponential Smoothing, and Random Forest.
    """
    print("Performing ARIMA Forecasting...")
    arima_forecast(df, forecast_periods=forecast_periods)

    print("Performing Exponential Smoothing Forecasting...")
    exponential_smoothing_forecast(df, forecast_periods=forecast_periods)

    print("Performing Random Forest Forecasting...")
    random_forest_forecast(df, forecast_periods=forecast_periods)

# Example Usage
folder_path = './Cleaned_Datasets'
commodity_name = Crop_Name

df = load_data(folder_path, commodity_name)
if df is not None:
    price_forecasting(df, forecast_periods=90)
```

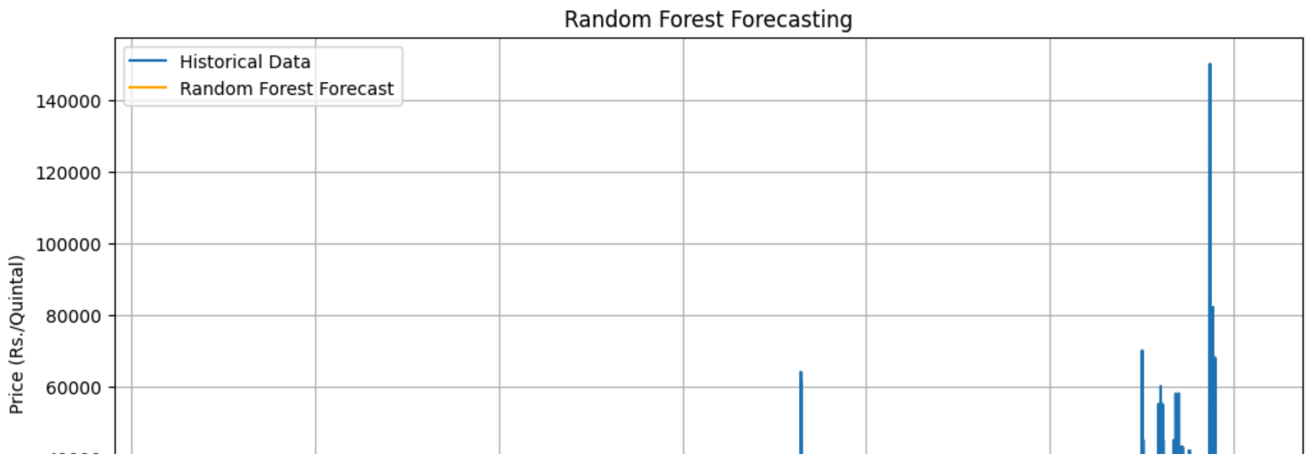
```
Performing ARIMA Forecasting...
```

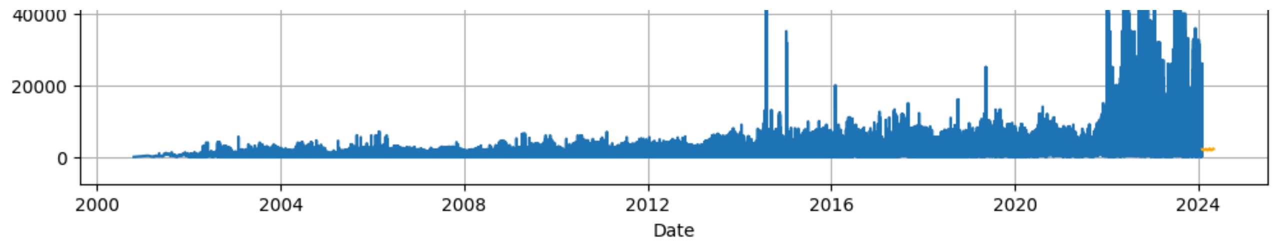


```
Performing Exponential Smoothing Forecasting...
```



```
Performing Random Forest Forecasting...
Random Forest R2 Score: 0.48
Random Forest RMSE: 955.02
C:\Users\sruth\AppData\Roaming\Python\Python311\site-packages\IPython\core\pylabtools.py:170: UserWarning:

Creating legend with loc="best" can be slow with large amounts of data.
```





Price Forecasting Using ARIMA, Exponential Smoothing, and Random Forest for Multiple Forecast Periods

```
# ARIMA Forecasting
def arima_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    df = df.set_index('reported_date').sort_index()
    time_series = df[column].resample('D').mean().ffill()  # Updated line

    model = ARIMA(time_series, order=(5, 1, 0))
    model_fit = model.fit()

    forecast = model_fit.forecast(steps=forecast_periods)
    forecast_index = pd.date_range(start=time_series.index[-1] + pd.Timedelta(days=1), periods=forecast_periods, freq='D')

    return forecast, forecast_index

# Exponential Smoothing Forecasting
def exponential_smoothing_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    df = df.set_index('reported_date').sort_index()
    time_series = df[column].resample('D').mean().ffill()  # Updated line

    model = ExponentialSmoothing(time_series, seasonal='add', seasonal_periods=365)
    model_fit = model.fit()

    forecast = model_fit.forecast(forecast_periods)
    forecast_index = pd.date_range(start=time_series.index[-1] + pd.Timedelta(days=1), periods=forecast_periods, freq='D')

    return forecast, forecast_index

# Random Forest Forecasting
def random_forest_forecast(df, column='modal_price_(rs./quintal)', forecast_periods=30):
    df['year'] = df['reported_date'].dt.year
    df['month'] = df['reported_date'].dt.month
    df['day'] = df['reported_date'].dt.day
    df['day_of_week'] = df['reported_date'].dt.dayofweek

    X = df[['year', 'month', 'day', 'day_of_week']]
    y = df[column]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = RandomForestRegressor(n_estimators=20, max_depth=10, n_jobs=-1, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    print(f"Random Forest R2 Score: {r2_score(y_test, y_pred):.2f}")
    print(f"Random Forest RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")

    last_date = df['reported_date'].max()
    forecast_dates = [last_date + pd.Timedelta(days=i) for i in range(1, forecast_periods + 1)]
    forecast_features = pd.DataFrame({
        'year': [date.year for date in forecast_dates],
        'month': [date.month for date in forecast_dates],
        'day': [date.day for date in forecast_dates],
        'day_of_week': [date.dayofweek for date in forecast_dates],
    })
    forecast = model.predict(forecast_features)

    return forecast, forecast_dates

# Main Forecasting Function
def price_forecasting(df, forecast_periods_list=[30, 90, 365]):
    """
    Perform price forecasting using ARIMA, Exponential Smoothing, and Random Forest for multiple forecast periods.
    """
    for forecast_periods in forecast_periods_list:
        print(f"\nAverage Forecast for {forecast_periods} Days:")

        # ARIMA Forecast
        arima_forecast_result, arima_forecast_index = arima_forecast(df, forecast_periods=forecast_periods)
        arima_avg_price = np.mean(arima_forecast_result)
        print(f"ARIMA Average Price for {forecast_periods} Days: {arima_avg_price:.2f} Rs./Quintal")

        # Exponential Smoothing Forecast
        exponential_smoothing_forecast_result, exponential_smoothing_forecast_index = exponential_smoothing_forecast(df, forecast_perioc
        exponential_smoothing_avg_price = np.mean(exponential_smoothing_forecast_result)
        print(f"Exponential Smoothing Average Price for {forecast_periods} Days: {exponential_smoothing_avg_price:.2f} Rs./Quintal")

        # Random Forest Forecast
        random_forest_forecast_result, random_forest_forecast_dates = random_forest_forecast(df, forecast_periods=forecast_periods)
        random_forest_avg_price = np.mean(random_forest_forecast_result)
        print(f"Random Forest Average Price for {forecast_periods} Days: {random_forest_avg_price:.2f} Rs./Quintal")

# Example Usage
folder_path = './Cleaned_Datasets'
commodity_name = Crop_Name

df = load_data(folder_path, commodity_name)
if df is not None:
    price_forecasting(df, forecast_periods_list=[30, 90, 365])
```

```
Average Forecast for 30 Days:
ARIMA Average Price for 30 Days: 2345.54 Rs./Quintal
Exponential Smoothing Average Price for 30 Days: 2266.88 Rs./Quintal
Random Forest R2 Score: 0.48
Random Forest RMSE: 955.02
Random Forest Average Price for 30 Days: 2058.49 Rs./Quintal

Average Forecast for 90 Days:
ARIMA Average Price for 90 Days: 2353.25 Rs./Quintal
Exponential Smoothing Average Price for 90 Days: 2279.13 Rs./Quintal
Random Forest R2 Score: 0.48
Random Forest RMSE: 955.02
Random Forest Average Price for 90 Days: 2056.62 Rs./Quintal

Average Forecast for 365 Days:
ARIMA Average Price for 365 Days: 2356.16 Rs./Quintal
Exponential Smoothing Average Price for 365 Days: 2401.92 Rs./Quintal
Random Forest R2 Score: 0.48
Random Forest RMSE: 955.02
Random Forest Average Price for 365 Days: 2864.74 Rs./Quintal
```

Fetching Gold Price Data for the Last 20 Years Using Yahoo Finance API

This script downloads the gold price data for the past 20 years (from the start date) using the Yahoo Finance API. The data is then saved into a CSV file containing the date and closing price for each day.

```
# Define the ticker symbol for gold (e.g., 'GC=F' for gold futures)
gold_ticker = 'GC=F'

# Get the current date and the date two years ago
end_date = datetime.today()
start_date = end_date - timedelta(days=365*20)

# Download gold price data for the last 2 years
gold_data = yf.download(gold_ticker, start=start_date, end=end_date, progress=False)

# Check if gold data is not empty
if not gold_data.empty:
    # Reset the index to make 'Date' a column
    gold_data.reset_index(inplace=True)

    # Select only the Date and Close columns
    gold_data = gold_data[['Date', 'Close']]

    # Save the data to a CSV file
    gold_data.to_csv('gold.csv', index=False)
    print("Gold price data (Date and Close) for the last 2 years has been saved to 'gold.csv'.")
else:
    print("Failed to fetch gold price data.")
```

```
Gold price data (Date and Close) for the last 2 years has been saved to 'gold.csv'.
```

Start coding or generate with AI.

Analyzing the Correlation Between Gold Price Change and Commodity Price Change

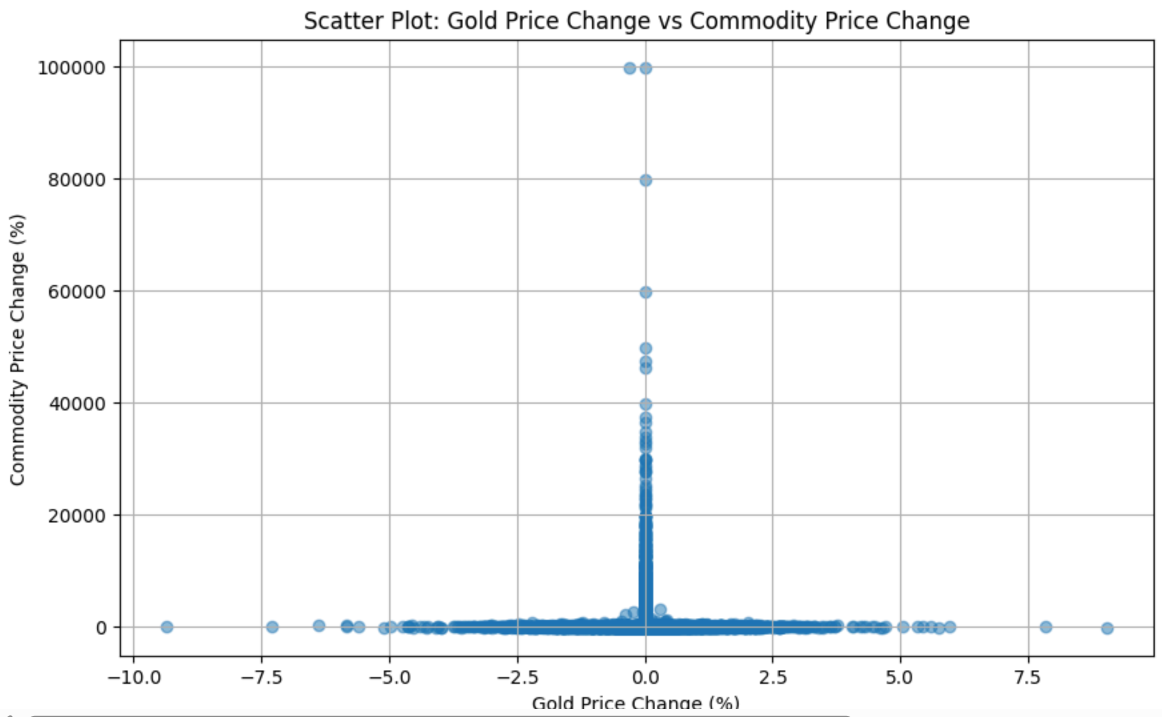
This script performs the following steps:

1. Loads commodity and gold price datasets.
2. Converts date columns to datetime format for both datasets.
3. Merges the datasets based on matching dates.
4. Calculates the percentage change in both commodity prices and gold prices.
5. Computes the correlation between gold price change and commodity price change.
6. Visualizes the relationship using a scatter plot.

The correlation value helps determine how changes in gold prices may affect commodity prices over the analyzed period.

```python
# Load the commodity dataset
commodity_df = pd.read_csv(f'./Cleaned_Datasets/{Crop_Name}.csv')

# Load the gold dataset
gold_df = pd.read_csv('gold.csv')

# Ensure the 'reported_date' is in datetime format for commodity data
commodity_df['reported_date'] = pd.to_datetime(commodity_df['reported_date'])

# Ensure the 'Date' is in datetime format for gold data
gold_df['Date'] = pd.to_datetime(gold_df['Date'])

# Convert 'Close' column in gold data to numeric, coercing errors to NaN (in case of non-numeric values)
gold_df['Close'] = pd.to_numeric(gold_df['Close'], errors='coerce')

# Drop rows with NaN values in 'Close' or 'modal_price_(rs./quintal)' columns
gold_df.dropna(subset=['Close'], inplace=True)
commodity_df.dropna(subset=['modal_price_(rs./quintal)'], inplace=True)

# Merge the commodity data and gold price data on the 'reported_date' (commodity) and 'Date' (gold data)
merged_df = pd.merge(commodity_df, gold_df, left_on='reported_date', right_on='Date', how='inner')

# Check if merge is successful and data is not empty
if not merged_df.empty:
    # Calculate the percentage change in the commodity price and gold price
    merged_df['commodity_price_change'] = (merged_df['modal_price_(rs./quintal)'].pct_change()) * 100
    merged_df['gold_price_change'] = (merged_df['Close'].pct_change()) * 100

    # Drop rows with NaN values resulting from percentage change calculation
    merged_df.dropna(subset=['commodity_price_change', 'gold_price_change'], inplace=True)

    # Check for infinite values in the data and replace them with NaN
    merged_df.replace([float('inf'), -float('inf')], float('nan'), inplace=True)

    # Drop rows with NaN values after replacing infinite values
    merged_df.dropna(subset=['commodity_price_change', 'gold_price_change'], inplace=True)

    # Calculate the correlation between gold price change and commodity price change
    correlation = merged_df['commodity_price_change'].corr(merged_df['gold_price_change'])

    print(f"Correlation between gold price change and commodity price change: {correlation}")

    # Scatter plot to visualize the relationship
    plt.figure(figsize=(10, 6))
    plt.scatter(merged_df['gold_price_change'], merged_df['commodity_price_change'], alpha=0.5)
    plt.title('Scatter Plot: Gold Price Change vs Commodity Price Change')
    plt.xlabel('Gold Price Change (%)')
    plt.ylabel('Commodity Price Change (%)')
    plt.grid(True)
    plt.show()

else:
    print("No matching dates found between gold and commodity data.")
```

```
Correlation between gold price change and commodity price change: -0.0010213494741488042
```



Analyzing the Impact of Gold Price Increases on Commodity Price Increases

This script performs the following tasks:

1. Loads commodity and gold price datasets.
2. Converts date columns to datetime format for both datasets.
3. Merges the datasets based on matching dates.
4. Identifies the days when gold price increased (i.e., today's price is higher than yesterday's price).
5. Checks if commodity price also increased on those days.
6. Calculates the percentage of times commodity prices increased when gold prices increased.

The result gives insights into how often commodity prices follow gold price increases.

```python
# Load the commodity dataset
commodity_df = pd.read_csv(f'./Cleaned_Datasets/{Crop_Name}.csv')

# Load the gold dataset
gold_df = pd.read_csv('gold.csv')

# Ensure the 'reported_date' is in datetime format for commodity data
commodity_df['reported_date'] = pd.to_datetime(commodity_df['reported_date'])

# Ensure the 'Date' is in datetime format for gold data
gold_df['Date'] = pd.to_datetime(gold_df['Date'])

# Merge the commodity data and gold price data on the 'reported_date' (commodity) and 'Date' (gold data)
merged_df = pd.merge(commodity_df, gold_df, left_on='reported_date', right_on='Date', how='inner')

# Check if merge is successful and data is not empty
if not merged_df.empty:
    # Identify where gold price increased (today's price > yesterday's price)
    merged_df['gold_price_increase'] = merged_df['Close'] > merged_df['Close'].shift(1)

    # Check if commodity price increased on these days
    merged_df['commodity_price_increase'] = merged_df['modal_price_(rs./quintal)'] > merged_df['modal_price_(rs./quintal)'].shift(1)

    # Find the days where gold price increased and check if commodity price also increased
    gold_increase_and_commodity_increase = merged_df[merged_df['gold_price_increase'] == True]['commodity_price_increase']

    # Calculate the percentage of times commodity price increased when gold price increased
    percentage_increase = gold_increase_and_commodity_increase.mean() * 100

    print(f"Percentage of times commodity price increased when gold price increased: {percentage_increase}%")

else:
    print("No matching dates found between gold and commodity data.")
```

```
Percentage of times commodity price increased when gold price increased: 50.19952114924182%
```

Enter the Names of Commodities for Analysis

Please provide the list of commodity names you want to analyze. These names will be used to load the respective datasets and perform the analysis.

```
commodity_names = ['Green Gram', 'Bengal Gram', 'Black Gram']  # List of your commodity names
```

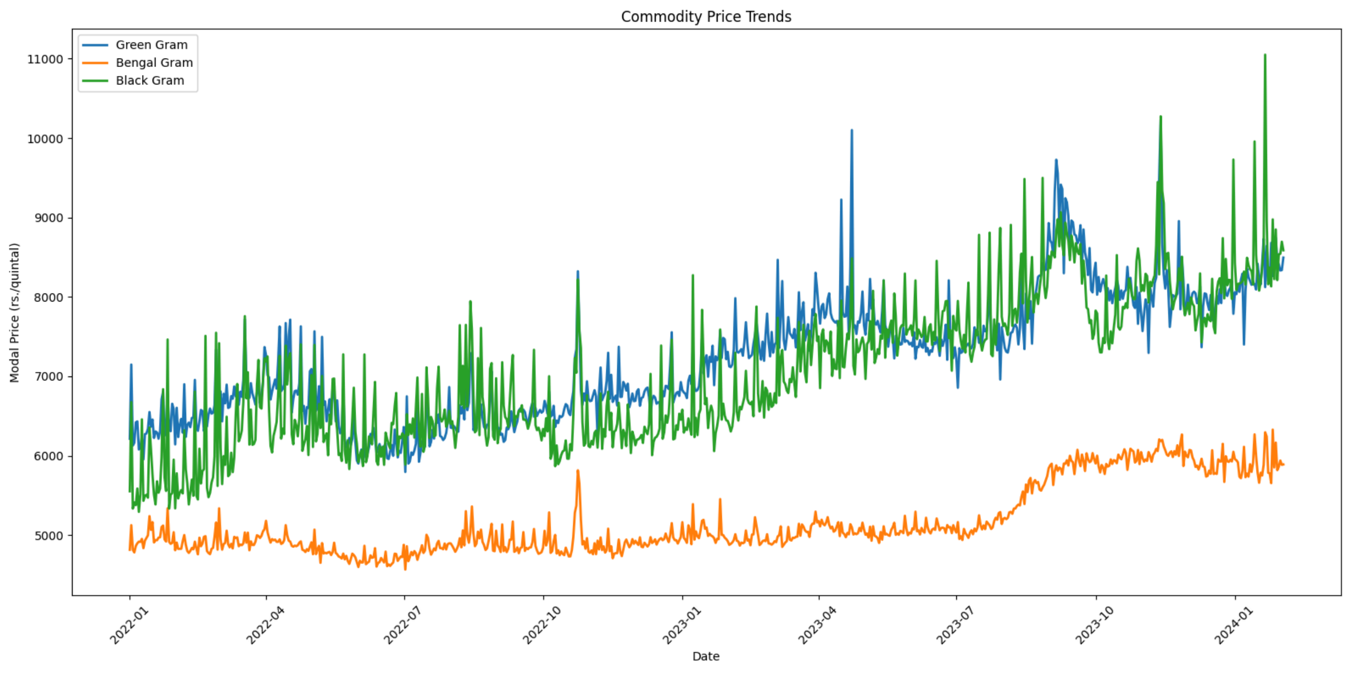
Comparing Commodity Prices and Regional Data

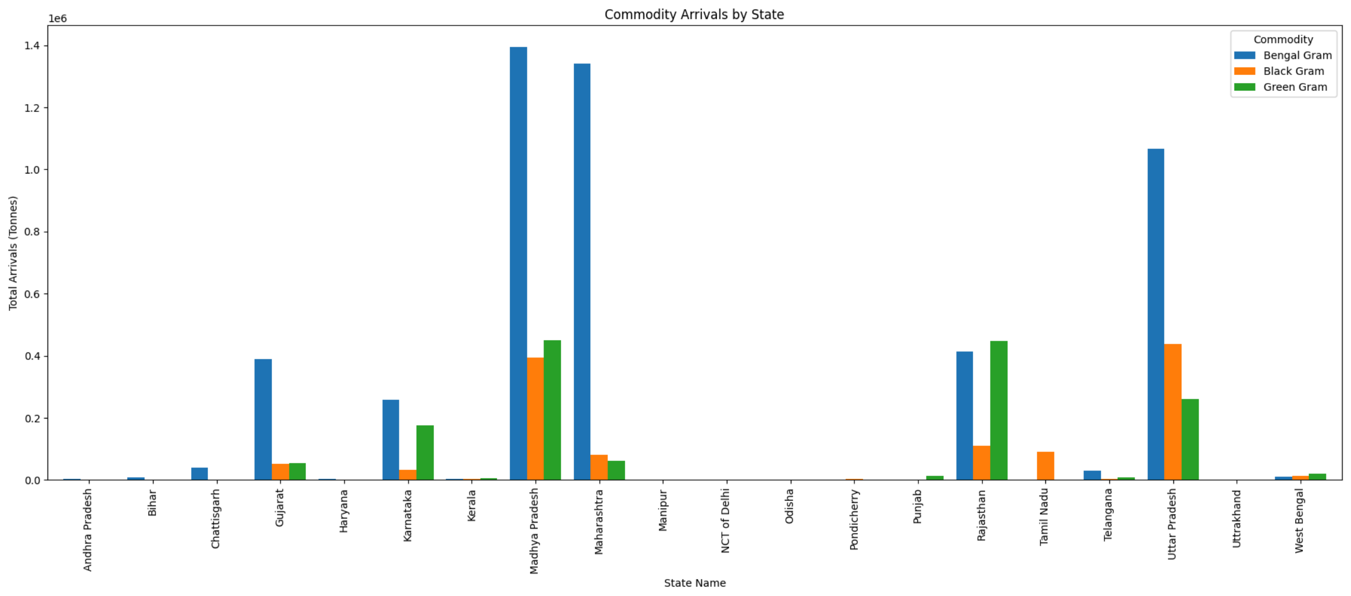
This function compares commodity prices over time and analyzes regional data by aggregating commodity arrivals across different states. You can specify a list of commodity names, the folder where the data is stored, and a date range for filtering the data. The function returns two outputs: one for commodity price trends and another for regional data based on state and arrivals in tonnes. The results can be visualized using multi-line graphs and bar charts.

Please provide the required input parameters, and the function will generate the visualizations based on the given data.

```
def compare_commodity_prices_and_regions(folder_path, commodity_names, start_date=None, end_date=None):
    commodity_data = []
    region_data = []

    for commodity in commodity_names:
        file_path = f"{folder_path}/{commodity}.csv"
        try:
            # Load the CSV file for the commodity
            df = pd.read_csv(file_path)

            # Check if necessary columns exist
            if 'modal_price_(rs./quintal)' not in df.columns or 'reported_date' not in df.columns or 'state_name' not in df.columns or
                print(f"Error: Missing necessary columns in {commodity} data.")
                continue

            # Clean 'modal_price_(rs./quintal)' by forcing errors to NaN
            df['modal_price_(rs./quintal)'] = pd.to_numeric(df['modal_price_(rs./quintal)'], errors='coerce')

            # Check for NaN or infinite values and clean them
            df = df.dropna(subset=['modal_price_(rs./quintal)', 'arrivals_(tonnes)'])
            df = df[~df['modal_price_(rs./quintal)'].isin([float('inf'), -float('inf')])]  # Remove infinite values

            # Remove invalid dates
            df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')
            df = df.dropna(subset=['reported_date'])

            # Filter the data based on the date range, if provided
            if start_date:
                df = df[df['reported_date'] >= pd.to_datetime(start_date)]
            if end_date:
                df = df[df['reported_date'] <= pd.to_datetime(end_date)]

            # Add the commodity name as a new column (commodity is the file name here)
            df['commodity'] = commodity

            # Aggregate region data based on state_name and sum the arrivals_(tonnes)
            region_counts = df.groupby(['state_name', 'commodity'])['arrivals_(tonnes)'].sum().reset_index()

            # Store the region data for plotting
            region_data.append(region_counts)

            # Add commodity to the price data
            df_resampled = df.groupby('reported_date').agg({'modal_price_(rs./quintal)': 'mean'}).reset_index()
            df_resampled['commodity'] = commodity
            commodity_data.append(df_resampled)
        except Exception as e:
            print(f"Error processing file {commodity}: {e}")
            continue

    # Concatenate the data for all commodities
    if commodity_data:
        df_all = pd.concat(commodity_data, ignore_index=True)
    else:
        df_all = None

    # Concatenate region data for all commodities
```

```
        if region_data:
            region_all = pd.concat(region_data, ignore_index=True)
        else:
            region_all = None

        return df_all, region_all

    # Example usage:
    folder_path = './Cleaned_Datasets'
    start_date = '2022-01-01'  # Start date
    end_date = '2024-12-31'    # End date

    result, region_result = compare_commodity_prices_and_regions(folder_path, commodity_names, start_date, end_date)

    # 1. Multi-Line Graph to Compare Commodity Prices
    if result is not None:
        plt.figure(figsize=(16, 8))  # Larger figure size to accommodate multiple commodities
        for commodity in result['commodity'].unique():
            df_commodity = result[result['commodity'] == commodity]
            plt.plot(df_commodity['reported_date'], df_commodity['modal_price_(rs./quintal)'], label=commodity, linewidth=2)  # Thicker line
        plt.xlabel('Date')
        plt.ylabel('Modal Price (rs./quintal)')
        plt.title('Commodity Price Trends')
        plt.legend()
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

    # 2. Compare Region Prices (State-level with arrivals_(tonnes))
    if region_result is not None:
        # Pivot the data to have states as rows and commodities as columns
        pivot_data = region_result.pivot_table(index='state_name', columns='commodity', values='arrivals_(tonnes)', aggfunc='sum', fill_valu

        # Plotting the bar chart
        pivot_data.plot(kind='bar', figsize=(18, 8), width=0.8)
        plt.xlabel('State Name')
        plt.ylabel('Total Arrivals (Tonnes)')
        plt.title('Commodity Arrivals by State')
        plt.xticks(rotation=90)
        plt.tight_layout()
        plt.legend(title='Commodity')
        plt.show()
```





Analyzing Commodity Price Trends and Volatility

This script performs an analysis on the commodity data, focusing on two main tasks: plotting monthly average price trends and calculating price volatility (standard deviation).

Functions:

- `load_data(file_path, commodity_name)` : Loads the data for a specific commodity and adds a 'commodity' column to the DataFrame.
- `plot_monthly_price_trends(df)` : Plots the monthly average price trends for each commodity in the dataset.
- `calculate_volatility(df)` : Calculates the price volatility (standard deviation of prices) for each commodity.
- `analyze_commodity_data(folder_path, commodity_names)` : Main function that processes the dataset for each commodity, plotting the monthly price trends and calculating volatility.

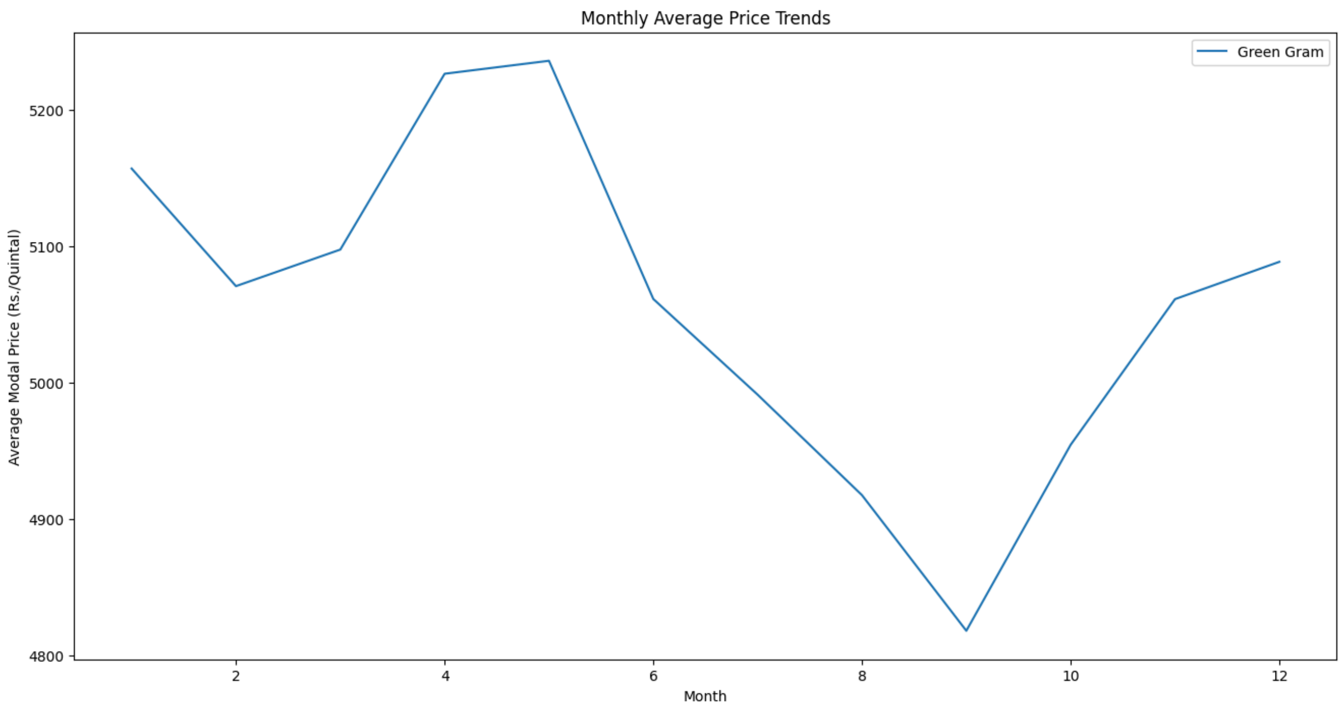
Inputs:

- A folder path containing commodity data files (CSV format).
- A list of commodity names whose data is stored in separate files.

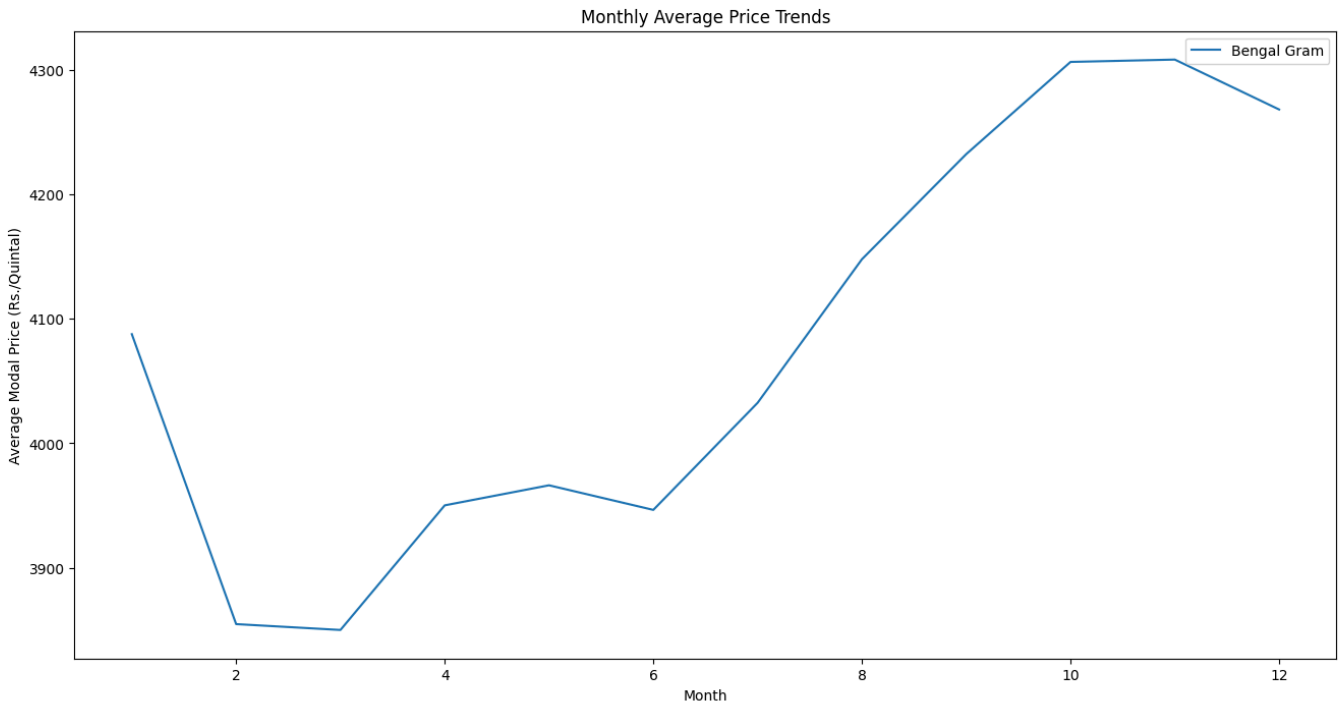
The function will display monthly price trends for each commodity and print the price volatility for each commodity.

```
# Function to load data and add commodity name based on the filename
def load_data(file_path, commodity_name):
    try:
        df = pd.read_csv(file_path)
        df['commodity'] = commodity_name  # Add a 'commodity' column
        df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')  # Ensure dates are in datetime format
        return df
    except Exception as e:
        print(f"Error loading {file_path}: {e}")
        return None

# Function to plot monthly price trends
def plot_monthly_price_trends(df):
    df['month'] = df['reported_date'].dt.month
    monthly_avg_price = df.groupby(['month', 'commodity'])['modal_price_(rs./quintal)'].mean().reset_index()

    plt.figure(figsize=(16, 8))
    for commodity in df['commodity'].unique():
        commodity_data = monthly_avg_price[monthly_avg_price['commodity'] == commodity]
        plt.plot(commodity_data['month'], commodity_data['modal_price_(rs./quintal)'], label=commodity)

    plt.title('Monthly Average Price Trends')
    plt.xlabel('Month')
    plt.ylabel('Average Modal Price (Rs./Quintal)')
    plt.legend()
    plt.show()

# Function to calculate price volatility (standard deviation of prices)
def calculate_volatility(df):
    volatility = df['modal_price_(rs./quintal)'].std()
    return volatility

# Main analysis function
def analyze_commodity_data(folder_path, commodity_names):
    for commodity_name in commodity_names:
        file_path = os.path.join(folder_path, f'{commodity_name}.csv')  # Assuming the files are named by commodity
        print(f"Processing file: {file_path}")

        df = load_data(file_path, commodity_name)

        if df is not None:
            # Plot monthly price trends
            plot_monthly_price_trends(df)

            # Calculate price volatility
            volatility = calculate_volatility(df)
            print(f"Price volatility for {commodity_name}: {volatility}")
        else:
            print(f"Skipping {commodity_name} due to loading errors.")

# Define folder path and commodity names
folder_path = './Cleaned_Datasets'
# Run the analysis
analyze_commodity_data(folder_path, commodity_names)
```

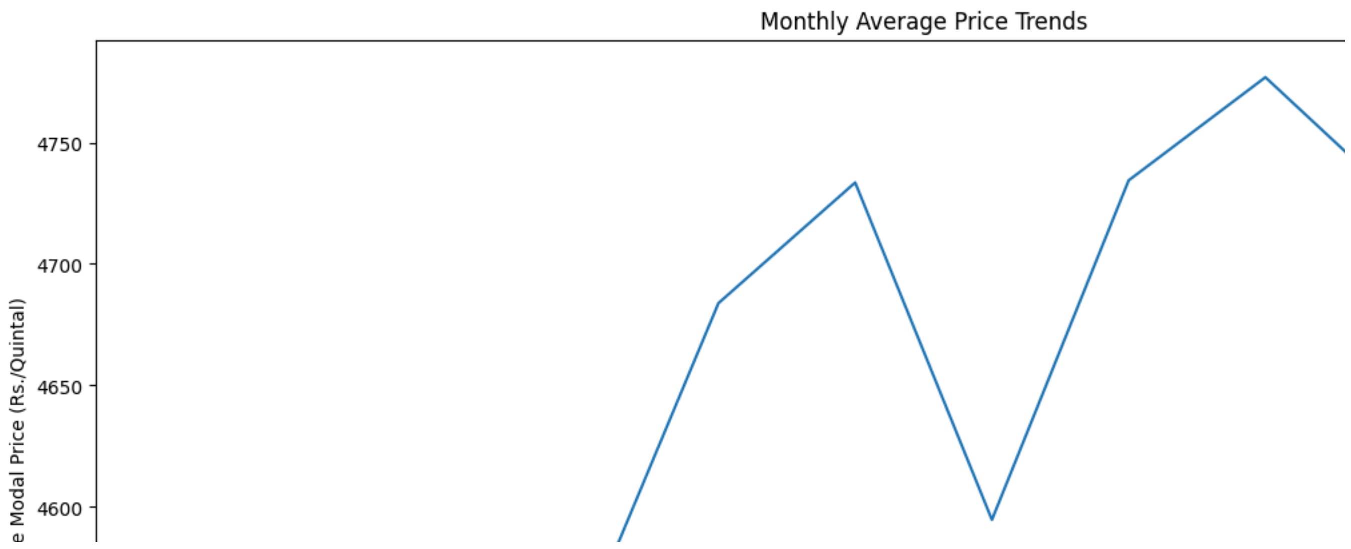
```
Processing file: ./Cleaned_Datasets\Green Gram.csv
```

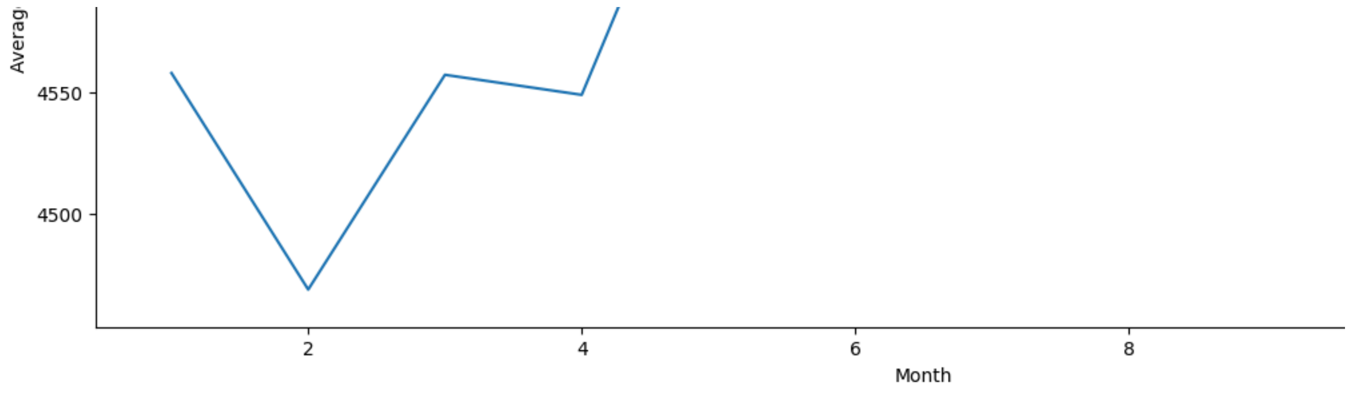


```
Price volatility for Green Gram: 2148.3220437237474
Processing file: ./Cleaned_Datasets\Bengal Gram.csv
```



```
Price volatility for Bengal Gram: 1558.540647856356
Processing file: ./Cleaned_Datasets\Black Gram.csv
```





```
Price volatility for Black Gram: 2573.012167774457
```

Commodity Price Prediction Analysis

Overview

This analysis aims to predict commodity prices for the next 12 months based on historical data, specifically using the relationship between commodity prices and gold prices. The analysis involves loading commodity price data, merging it with gold price data, training a linear regression model, and making future price predictions.

Steps Involved:

1. **Data Loading**

- **Commodity Data**: Loads commodity data from CSV files. Each file is named after the commodity (e.g., `commodity_name.csv`) and contains columns such as `modal_price_(rs./quintal)` and `reported_date`.
- **Gold Price Data**: Loads historical gold price data from a CSV file (`gold.csv`). The file must contain columns like `Date` and `Close` (gold price).

2. **Data Merging**

- The commodity price data is merged with the gold price data based on matching dates (i.e., `reported_date` for commodities and `Date` for gold). Rows with missing or invalid values are removed during the merging process.

3. **Model Training**

- **Features**: The linear regression model is trained using the gold price (`Close`) and the lagged commodity price (i.e., previous day's price) as features.
- The data is split into training and testing sets, and the model is trained to predict the commodity price based on these features.
- **Evaluation**: The model's performance is evaluated using Mean Squared Error (MSE) to assess the prediction accuracy.

4. **Price Prediction**

- Future gold prices are randomly generated (in this example), and the trained model is used to predict commodity prices for the next 12 months.
- The predicted prices are plotted on a graph for each commodity.

5. **Plotting**

- A line plot is generated to visualize the predicted commodity prices for the next 12 months. Each commodity is represented by a separate line on the plot.

Expected Output

- A line plot showing the predicted commodity prices over the next 12 months for each commodity.
- A printed Mean Squared Error (MSE) value for each commodity to evaluate the model's performance.

Example Usage

To run this analysis:

1. Prepare your commodity data CSV files, where each file is named after the commodity (e.g., `commodity_name.csv`).
2. Ensure that the gold price data is in a file named `gold.csv` with columns `Date` and `Close`.
3. Run the script to generate predictions and visualize the results.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import warnings  # Import the warnings module
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Suppress warnings
warnings.filterwarnings("ignore")

# Function to load data and add commodity name based on the filename
def load_data(file_path, commodity_name):
    try:
        df = pd.read_csv(file_path)
        df['commodity'] = commodity_name  # Add a 'commodity' column
        df['reported_date'] = pd.to_datetime(df['reported_date'], errors='coerce')  # Ensure dates are in datetime format
        return df
    except Exception as e:
        print(f"Error loading {file_path}: {e}")
        return None

# Function to load gold price data
def load_gold_data(gold_price_file):
    try:
        gold_df = pd.read_csv(gold_price_file)
        print(gold_df.columns)  # Check the column names
        gold_df['Date'] = pd.to_datetime(gold_df['Date'], errors='coerce')  # Ensure date is properly parsed
        return gold_df
    except Exception as e:
        print(f"Error loading gold price data: {e}")
        return None

# Function to merge commodity and gold price data
def merge_data(commodity_df, gold_df):
    # Merge on reported_date and Date columns (ensure both are datetime)
    merged_df = pd.merge(commodity_df, gold_df, left_on='reported_date', right_on='Date', how='left')
    merged_df.dropna(subset=['modal_price_(rs./quintal)', 'Close'], inplace=True)
    return merged_df

# Train a simple predictive model (Linear Regression)
def train_predictive_model(df):
    # Feature: gold price and past commodity price (lag 1)
    df['lag_price'] = df['modal_price_(rs./quintal)'].shift(1)
    df.dropna(subset=['lag_price'], inplace=True)

    # Features (Gold price + Lag of commodity price)
    X = df[['Close', 'lag_price']]  # Using gold price ('Close') and lag of commodity price
    y = df['modal_price_(rs./quintal)']

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train a linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Evaluate model
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")

    return model

# Function to predict future prices using the trained model
def predict_prices(model, df, future_gold_prices):
    # Use the model to predict commodity prices based on future gold prices
    last_known_price = df['modal_price_(rs./quintal)'].iloc[-1]  # Last known price of commodity
    last_known_gold_price = df['Close'].iloc[-1]  # Last known gold price

    predicted_prices = []
    for gold_price in future_gold_prices:
        # Predict based on lag price and gold price
        predicted_price = model.predict([[gold_price, last_known_price]])[0]
        predicted_prices.append(predicted_price)
        last_known_price = predicted_price  # Update lag for next prediction

    return predicted_prices

# Main analysis function
def analyze_commodity_data(folder_path, commodity_names, gold_price_file):
    # Load gold price data
    gold_df = load_gold_data(gold_price_file)

    if gold_df is None:
        print("Unable to load gold price data.")
        return

    # Create a single plot for all commodities
    plt.figure(figsize=(10, 6))  # Set the size of the plot

    for commodity_name in commodity_names:
        file_path = os.path.join(folder_path, f'{commodity_name}.csv')  # Assuming the files are named by commodity
        print(f"Processing file: {file_path}")

        commodity_df = load_data(file_path, commodity_name)

        if commodity_df is not None:
            # Merge the commodity data with gold price data
            merged_df = merge_data(commodity_df, gold_df)

            # Train a predictive model
            model = train_predictive_model(merged_df)

            # Predict future prices (for example, for the next 12 months)
            future_gold_prices = np.random.uniform(50000, 60000, size=12)  # Example: random gold prices for the next 12 months
            predicted_prices = predict_prices(model, merged_df, future_gold_prices)

            print(f"Predicted prices for next 12 months for {commodity_name}: {predicted_prices}")

            # Plotting the predicted prices for each commodity
            months = [f'Month {i+1}' for i in range(12)]  # Labels for 12 months
            plt.plot(months, predicted_prices, marker='o', label=f'Predicted Prices for {commodity_name}')

        else:
            print(f"Skipping {commodity_name} due to loading errors.")

    # Final plot configuration
    plt.xlabel('Months')
    plt.ylabel('Predicted Price (Rs./quintal)')
    plt.title(f"Predicted Commodity Prices Over the Next 12 Months")
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.legend()
    plt.show()

# Define folder path, commodity names, and gold price file
folder_path = './Cleaned_Datasets'
gold_price_file = './gold.csv'  # Path to your gold price data

# Run the analysis
analyze_commodity_data(folder_path, commodity_names, gold_price_file)
```