Wholesale Price Index Analysis and Forecasting

Abstract

This report presents an analysis and forecasting of the Wholesale Price Index (WPI) from 1990 to 2024 using a dataset provided in tabular form. The study employs statistical and machine learning techniques to understand trends, stationarity, and future predictions of WPI. The methodology includes data preprocessing, exploratory analysis, and the application of models to forecast price fluctuations using a comparative analysis of ARMA (statistical), Random Forest, XGBoost (machine learning), and LSTM (deep learning) models. Leveraging a dataset of 35 annual observations, we assess model performance over the 2020-2024 test period using Root Mean Squared Error (RMSE). The results provide insights into WPI trends and predictive performance, contributing to economic planning and sustainability goals.

Introduction

From the statement, we understand that agricultural markets are influenced by multiple factors, including climate variability, supply chain disruptions, and economic conditions, which lead to unpredictable price fluctuations. Understanding and forecasting these price movements is crucial for farmers, traders, and policymakers to make informed decisions. While traditional models have been widely used for price prediction, they often struggle with capturing non-linear trends and sudden market shifts. With advancements in statistical and machine learning techniques, more sophisticated approaches can enhance forecasting accuracy.

However, in this study, we analyse the wholesale price index (WPI) of potatoes from 1990 to 2024 and find that the traditional **AutoRegressive Moving Average (ARMA) model** outperforms other predictive approaches. Despite the increasing interest in complex machine-learning models, ARMA proves to be the most effective in capturing price trends and fluctuations in our dataset. This highlights the continued relevance of classical time series models in agricultural price forecasting.

Objectives

- To perform an Exploratory Data Analysis(EDA) on the dataset so as to understand the data's structure and characteristics.
- To check if the time series is stationary or non-stationary in order to know if the statistical properties of the data remain constant over time.
- To develop models that can help predict future WPI values (like ARMA, XG Boost, LSTM, etc.)

- To make a comparison among different models to know which is the most efficient in performing its task.
- To understand what the models are implying and assess their significance in economic planning.

Methodology

1. Data Acquisition & Preprocessing

The dataset comprises the Wholesale Price Index (WPI) of potatoes from 1990 to 2024. Before modelling, necessary preprocessing steps were undertaken:

- **Handling Missing Values:** Checked for and addressed any missing or inconsistent data points.
- **Outlier Detection:** Identified outliers using Boxplot.

2. Exploratory Data Analysis (EDA)

To gain insights into the underlying patterns and volatility in the data, the following analyses were conducted:

- **Summary Statistics:** Checked the basic descriptive statistics of the data
- **Stationarity Check:** The Augmented Dickey-Fuller (ADF) test was used to determine stationarity; differencing was applied if required.
- **Fourier Transform:** Applied Fourier analysis to identify dominant frequency components in the time series and assess seasonal patterns.
- **Line Plot:** Plotted the WPI data over time to observe trends, seasonality, and anomalies.
- **Autocorrelation Analysis:** Examined Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots to identify lag dependencies.
- **Volatility Assessment:** Measured fluctuations in WPI to understand price instability.
- **Yearly Percentage Change:** Calculated year-over-year variations to identify major price shifts.

3. Model Selection & Implementation

To forecast price fluctuations, a combination of statistical and machine-learning models were employed, each offering unique advantages:

- **ARMA (AutoRegressive Moving Average):** Captures linear dependencies in the time series by modelling both autoregressive and moving average components.
- **Random Forest Regression:** Uses an ensemble of decision trees to identify patterns and relationships in price movements.

- **XGBoost Regression:** A gradient boosting model designed to capture complex, non-linear price fluctuations effectively.
- **LSTM (Long Short-Term Memory Networks):** A deep learning model capable of learning long-term dependencies in sequential data, making it well-suited for time series forecasting.
- **Hybrid Model (ARMA + LSTM):** Combines the strengths of statistical and deep learning approaches to enhance predictive accuracy by leveraging both historical trends and sequential dependencies.

The dataset was split into training (1990-2019) and testing (2020-2024) sets for model validation, ensuring robust performance evaluation. Hyperparameter tuning was applied where necessary to enhance performance and accuracy.

4. Model Evaluation & Forecasting

The models were evaluated using **Root Mean Squared Error (RMSE)** to measure predictive accuracy. The model with the lowest RMSE was selected for final forecasting. Based on this, future price trends were predicted, providing valuable insights for risk assessment, policy planning, and market decision-making.

Result and Discussion

Exploratory Data Analysis Findings

1. Data Quality and Summary Statistics

- The dataset was thoroughly examined for completeness, and no missing values were found.
- Outlier analysis did not reveal any extreme values significantly deviating from the distribution.
- The **mean WPI was 215.4**, providing a central measure of the price index over the years.

2. Seasonality and Trend Analysis

- **Autocorrelation Function (ACF) and Fourier analysis** were conducted to detect seasonality patterns.
- The results indicated **no significant recurring seasonal patterns**, suggesting that price fluctuations were not cyclical.

3. Volatility Analysis

- The dataset was assessed for **volatility clustering** using statistical tests.
- The findings showed **consistent price fluctuations over time**, meaning there were no prolonged periods of high or low volatility.

4. **Stationarity Testing**

- The **Augmented Dickey-Fuller (ADF) test** was applied to check for stationarity.
- Since our $p < 0.05$, the test results confirmed that the data was **already stationary**, removing the need for differencing before model implementation.

5. **Model Parameter Selection**

- **Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots** were analysed to determine the appropriate lag values.
- These plots helped in selecting optimal parameters for the **ARMA model**, ensuring better predictive performance.

<u>**Model Performance**</u>

After implementing and evaluating multiple forecasting models, their performance was compared using **Root Mean Squared Error (RMSE)** to determine the most effective approach for predicting WPI fluctuations.

Model Performance Comparison

The RMSE values for the models were as follows:

- **ARMA**: **27.96**
- **XGBoost**: **28.92**
- **Random Forest**: **31.78**
- **LSTM**: **35.69**
- **Hybrid Model (ARMA + LSTM)**: **40.48**

Each model was selected for the following reasons:

- **ARMA** was used as a benchmark statistical model, leveraging its strength in capturing linear dependencies and short-term correlations in time-series data.
- **XGBoost** was implemented to explore non-linear relationships and feature importance in price fluctuations using a gradient-boosting approach.
- **Random Forest** was chosen as another ensemble learning method to analyse how well tree-based models could capture complex patterns in WPI data.

- **LSTM** was applied to assess deep learning's ability to model sequential dependencies and long-term patterns.
- **A hybrid Model (ARMA + LSTM)** was tested to combine ARMA's statistical efficiency with LSTM's capacity to learn complex, long-range dependencies.

Among these models, **ARMA** achieved the lowest RMSE of **27.96**, indicating that it provided the most accurate forecasts for WPI fluctuations. While machine learning models like **XGBoost** and **Random Forest** also performed well, deep learning-based **LSTM** and the **hybrid model (ARMA + LSTM)** showed higher RMSE values, suggesting they struggled to capture patterns effectively in this dataset. The relatively poorer performance of the hybrid model indicates that ARMA and LSTM did not complement each other effectively in this case.

Conclusion

This study aimed to develop an accurate forecasting model for the Wholesale Price Index (WPI) of potatoes (1990-2024) by comparing statistical and machine learning approaches. Through exploratory data analysis (EDA), we identified key trends, stationarity, volatility, and price fluctuations. Various models, including **ARMA, XGBoost, Random Forest, LSTM**, and a **Hybrid ARMA + LSTM model**, were implemented and evaluated using **Root Mean Squared Error (RMSE).**

Among the tested models, **ARMA achieved the lowest RMSE (27.96)**, making it the most effective model for forecasting WPI fluctuations. While machine learning models like XGBoost and Random Forest also performed reasonably well, deep learning-based LSTM and the hybrid model struggled to capture price patterns effectively. These results suggest that traditional statistical models remain highly effective for time-series forecasting in agricultural price data.

Using the ARMA model, we forecasted WPI values for the next five years (2025-2029) to provide insights into expected price trends. However, since the dataset contained only 35 data points, the accuracy of the forecast was limited. The small dataset size impacted the model's ability to capture long-term trends effectively, leading to reduced predictive reliability. This highlights the challenges of time-series forecasting with limited historical data and suggests that incorporating additional data sources or external factors could improve future predictions.

How Our Forecasting Model Supports the UN Sustainable Development Goals (SDGs)

Our forecasting model contributes to multiple **Sustainable Development Goals (SDGs)** by enhancing agricultural and economic resilience through data-driven decision-making.

SDG 1: No Poverty

Reliable predictions help farmers anticipate market trends, optimise planting cycles, and time their sales effectively. This improves income stability, reduces financial losses, and supports economic security in rural communities.

SDG 2: Zero Hunger

Accurate price forecasts help predict market fluctuations, allowing stakeholders to adjust supply chains proactively. This ensures food availability and price stability, reducing hunger risks for low-income households reliant on staple crops.

SDG 12: Responsible Consumption and Production

By aligning production with demand, price forecasting helps prevent overproduction and reduce waste. This promotes sustainable resource use in agriculture, enhances market efficiency, and supports long-term environmental and economic sustainability.

Code

Packages and Libraries

```
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#ignore warnings
import warnings
warnings.filterwarnings("ignore")

import warnings
warnings.simplefilter("ignore", category=UserWarning)
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter("ignore", category=ConvergenceWarning)

# Time series analysis
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.signal import periodogram

# Scikit-learn
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# Machine learning models
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import xgboost as xgb

# TensorFlow/Keras for LSTM
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import itertools

# !pip install statsmodels xgboost scikit-learn

"""## **Reading Data**"""

data = '/content/WPI_Data.xlsx'

# Reading the xlsx sheet
df = pd.read_excel(data, sheet_name=1)

print("Data Shape:", df.shape)

print(df.head())
```

```
"""# **Exploratory Data Analysis**

### **WPI Over Time - Line Plot**
"""

plt.figure(figsize=(10, 5))
plt.plot(df["Year"], df["WPI"], marker="o", linestyle="-", color="b", label="WPI")
plt.xlabel("Year")
plt.ylabel("Wholesale Price Index (WPI)")
plt.title("WPI of Potatoes Over Time")
plt.legend()
plt.grid(True)
plt.show()

"""###**Summary Statistics**"""

summary_stats = df["WPI"].describe()
print("Summary Statistics\n", summary_stats)

"""### **Missing Values**"""

missing_values = df.isnull().sum()
print("\nMissing Values:\n", missing_values)

"""### **Box Plot**"""

plt.figure(figsize=(8, 5))
sns.boxplot(x=df["WPI"], color="skyblue")
plt.title("Boxplot of WPI to Identify Outliers")
plt.show()

"""### **ACF and Fourier Analysis**"""

#ACF Analysis
```

```
plt.figure(figsize=(10, 5))
plot_acf(df['WPI'], lags=10)
plt.title("Autocorrelation Function (ACF) for WPI")
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.show()

# Fourier Analysis
years = df['Year'].values
wpi_values = df['WPI'].values
freqs, power = periodogram(wpi_values, scaling='spectrum')

# Periodogram
plt.figure(figsize=(10, 5))
plt.plot(freqs, power, marker='o', linestyle='-')
plt.xlabel("Frequency (1/Years)")
plt.ylabel("Power Spectrum")
plt.title("Spectral Analysis (Fourier Transform) of WPI")
plt.xlim(0, 0.5)

plt.grid(True)
plt.show()

"""### **WPI Percentage Change**"""

# Year-over-year changes
yoy_change = df['WPI'].pct_change() * 100

# Plotting
plt.figure(figsize=(12, 6))
plt.bar(df.index[1:], yoy_change[1:])
plt.title('Year-over-Year Percentage Change in WPI')
plt.xlabel('Year')
plt.ylabel('Percentage Change (%)')
```

```
plt.grid(True)
plt.axhline(y=0, color='r', linestyle='-')
plt.tight_layout()
plt.show()
```

"""### **Volatility Analysis of WPI"""**

```
volatility = df['WPI'].rolling(window=5).std()

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(df.index[4:], volatility[4:])
plt.xlabel('Year', fontsize = 14)
plt.ylabel('Standard Deviation', fontsize = 14)
plt.grid(True)
plt.tight_layout()
plt.show()
```

"""### **ADF (Augmented Dickey Fuller Test)"""**

```
# ADF test
result = adfuller(df['WPI'])

print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])

# Stationarity check
if result[1] <= 0.05:
    print("Data is stationary")
else:
    print("Data is non-stationary")
```

"""## **Analysis and Modelling**

```
### **Splitting Data**
"""

df.set_index('Year', inplace=True)

train_data = df['WPI'][:28]  # First 28 years for training
test_data = df['WPI'][28:]   # Remaining 7 years for testing

# Scale
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_data.values.reshape(-1, 1))
test_scaled = scaler.transform(test_data.values.reshape(-1, 1))

"""### **ACF and PACF**"""

plt.figure(figsize=(12,5))
plot_acf(df["WPI"].dropna(), lags=10)
plt.xlabel('Lag')
plt.ylabel('ACF')
plot_pacf(df["WPI"].dropna(), lags=10)
plt.xlabel('Lag')
plt.ylabel('PACF')
plt.show()
print(df.shape)

"""### **Best ARMA order**"""

p_values = range(0, 10)
d_values = [0]  # since it is stationary
q_values = range(0, 9)
pdq_combinations = list(itertools.product(p_values, d_values, q_values))

best_rmse = float('inf')
```

```
best_aicc = float('inf')
best_order = None
results = []

# evaluate model
for order in pdq_combinations:
    try:
        model = sm.tsa.ARIMA(train_data, order=order).fit()
        predictions = model.predict(start=len(train_data), end=len(df)-1)
        rmse = np.sqrt(mean_squared_error(test_data, predictions))
        aicc = model.aicc

        results.append((order, rmse, aicc))

        if rmse < best_rmse:
            best_rmse = rmse
            best_aicc = aicc
            best_order = order
    except:
        continue

# Best Model
results_df = pd.DataFrame(results, columns=['Order', 'RMSE', 'AICc'])
results_df = results_df.sort_values(by='RMSE')
print(results_df.head())
print(f"Best ARMA Order: {best_order}, RMSE: {best_rmse:.2f}, AICc: {best_aicc:.2f}")
print(df.shape)
```

"""### **ARMA"""**

```
arma_model = ARIMA(train_data, order=(3,0,1))
arma_fit = arma_model.fit()
arma_forecast = arma_fit.predict(start=len(train_data), end=len(df)-1)
```

```
"""### **Checking Seasonality**"""

P_values = [0, 1, 2]
D_values = [0, 1]
Q_values = [0, 1, 2]
s = 1  # since it is yearly

# Generating all combinations of p,d,q
seasonal_combinations = list(itertools.product(P_values, D_values, Q_values))

best_rmse = float('inf')
best_order = (3,0,1)
best_seasonal_order = None

# Loop through combinations
for seasonal_order in seasonal_combinations:
    try:
        model = sm.tsa.SARIMAX(train_data, order=best_order,
                               seasonal_order=(seasonal_order[0], seasonal_order[1], seasonal_order[2], s))
        fit = model.fit()

        predictions = fit.predict(start=len(train_data), end=len(df)-1)
        rmse = np.sqrt(mean_squared_error(test_data, predictions))

        # best model
        if rmse < best_rmse:
            best_rmse = rmse
            best_seasonal_order = seasonal_order

    except:
        continue

print(f"Best SARIMA Order: {best_order}, Seasonal Order: {best_seasonal_order}, RMSE: {best_rmse:.4f}")
```

```
"""### **Random Forest and XGBoost**"""

# Lag Features based on ARMA (3,0,1)
df_lags = df.copy()  # Copy of original data
df_lags['Lag1'] = df_lags['WPI'].shift(1)
df_lags['Lag2'] = df_lags['WPI'].shift(2)
df_lags['Lag3'] = df_lags['WPI'].shift(3)
df_lags.dropna(inplace=True)

print("After lag features:", df_lags.shape)

df_lags.fillna(method="bfill", inplace=True)

# Splitting Data using df_lags
train_size = int(len(df_lags) * 0.8)
train, test = df_lags.iloc[:train_size], df_lags.iloc[train_size:]

# Scaling
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train[['WPI', 'Lag1', 'Lag2', 'Lag3']])
test_scaled = scaler.transform(test[['WPI', 'Lag1', 'Lag2', 'Lag3']])

X_train, y_train = train_scaled[:, 1:], train_scaled[:, 0]  # Lag features as X, WPI as y
X_test, y_test = test_scaled[:, 1:], test_scaled[:, 0]

# Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_forecast_scaled = rf_model.predict(X_test).reshape(-1, 1)
rf_forecast = scaler.inverse_transform(np.hstack((rf_forecast_scaled, X_test)))[:, 0]

# XGBoost
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.05,
objective='reg:squarederror', random_state=42)
```

```
xgb_model.fit(X_train, y_train)
xgb_forecast_scaled = xgb_model.predict(X_test).reshape(-1, 1)
xgb_forecast = scaler.inverse_transform(np.hstack((xgb_forecast_scaled, X_test)))[:, 0]

# RMSE
rmse_rf = np.sqrt(mean_squared_error(test['WPI'], rf_forecast))
rmse_xgb = np.sqrt(mean_squared_error(test['WPI'], xgb_forecast))

print(f"Random Forest RMSE: {rmse_rf:.2f}")
print(f"XGBoost RMSE: {rmse_xgb:.2f}")

test['RF_Prediction'] = rf_forecast
test['XGB_Prediction'] = xgb_forecast

df = df[['WPI']]
```

"""### **Long Short-Term Memory"""**

```
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length)])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

sequence_length = 3  # Matches ARMA p=3

full_data = df['WPI'].values.reshape(-1, 1)
scaler_lstm = MinMaxScaler()
scaled_data = scaler_lstm.fit_transform(full_data)

# Splitting data
train_size = int(len(full_data) * 0.8)
train_scaled_lstm = scaled_data[:train_size]
```

```
test_scaled_lstm = scaled_data[train_size:]

X_train, y_train = create_sequences(train_scaled_lstm, sequence_length)
X_test, y_test = create_sequences(test_scaled_lstm, sequence_length)

print(f"Total data points: {len(full_data)}")
print(f"Train data points: {len(train_scaled_lstm)}")
print(f"Test data points: {len(test_scaled_lstm)}")
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")

# LSTM model
model = Sequential([
    LSTM(10, input_shape=(sequence_length, 1)),
    Dropout(0.2),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model.fit(X_train, y_train, epochs=30, batch_size=8, validation_split=0.2,
callbacks=[early_stopping], verbose=0)

# RMSE
test_predictions = model.predict(X_test, verbose=0)
test_predictions = scaler_lstm.inverse_transform(test_predictions)
y_test_actual = scaler_lstm.inverse_transform(y_test.reshape(-1, 1))
lstm_rmse = np.sqrt(mean_squared_error(y_test_actual, test_predictions))
print(f'LSTM RMSE: {lstm_rmse}')

"""### **ARMA+LSTM**"""

np.random.seed(42)
tf.random.set_seed(42)
```

```
train_data = df['WPI'][:28]
test_data = df['WPI'][28:]

# ARMA model
arma_model = ARIMA(train_data, order=(3, 0, 1))
arma_fit = arma_model.fit()
arma_train_pred = arma_fit.predict(start=0, end=len(train_data) - 1)
arma_forecast = arma_fit.predict(start=len(train_data), end=len(df) - 1)
print(f"arma_forecast length before trimming: {len(arma_forecast)}")
arma_forecast = pd.Series(arma_forecast.values, index=test_data.index)
print(f"arma_forecast length after trimming: {len(arma_forecast)}")
arma_rmse = np.sqrt(mean_squared_error(test_data, arma_forecast))

# scale residuals
train_residuals = train_data - arma_train_pred
scaler_lstm = MinMaxScaler()
train_residuals_scaled = scaler_lstm.fit_transform(train_residuals.values.reshape(-1, 1))

# sequences from scaled residuals
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length)])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

sequence_length = 3
X_train, y_train = create_sequences(train_residuals_scaled, sequence_length)

print(f"Total data points: {len(df)}")
print(f"Train points: {len(train_data)}")
print(f"Test points: {len(test_data)}")
print(f"X_train shape: {X_train.shape}")
```

```
# LSTM
model = Sequential([
    LSTM(10, input_shape=(sequence_length, 1)),
    Dropout(0.3),
    Dense(1)
])
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model.fit(X_train, y_train, epochs=50, batch_size=8, validation_split=0.2, shuffle=False,
          callbacks=[early_stopping], verbose=0)

last_sequence = train_residuals_scaled[-sequence_length:]
test_residuals_pred_scaled = []
current_sequence = last_sequence.copy()
for _ in range(len(test_data)):
    current_sequence_reshaped = current_sequence.reshape((1, sequence_length, 1))
    next_pred = model.predict(current_sequence_reshaped, verbose=0)
    test_residuals_pred_scaled.append(next_pred[0, 0])
    current_sequence = np.roll(current_sequence, -1)
    current_sequence[-1] = next_pred[0, 0]

test_residuals_pred =
scaler_lstm.inverse_transform(np.array(test_residuals_pred_scaled).reshape(-1, 1))
print(f"test_residuals_pred length: {len(test_residuals_pred)}")

# Combine ARMA with LSTM residuals
hybrid_forecast = arma_forecast + test_residuals_pred.flatten()

# RMSE
ARMA_LSTM_rmse = np.sqrt(mean_squared_error(test_data, hybrid_forecast))

test_residuals_actual = test_data - arma_forecast
print(f"test_residuals_actual length: {len(test_residuals_actual)}")
```

```
residual_rmse = np.sqrt(mean_squared_error(test_residuals_actual, test_residuals_pred.flatten()))

print(f'Hybrid ARMA-LSTM RMSE: {ARMA_LSTM_rmse:.4f}')
```

"""### **Conclusion"""**

```
arma_rmse = np.sqrt(mean_squared_error(test_data, arma_forecast))
xgb_rmse = np.sqrt(mean_squared_error(test_data, xgb_forecast))
rf_rmse = np.sqrt(mean_squared_error(test_data, rf_forecast))

rmse_values = [arma_rmse, rmse_rf, rmse_xgb, lstm_rmse, ARMA_LSTM_rmse]
model_names = ["ARMA", "Random Forest", "XGBoost", "LSTM", "ARMA+LSTM"]

# Sorting results
sorted_indices = np.argsort(rmse_values)
print("RMSE Results:\n")
for index in sorted_indices:
    print(f"{model_names[index]}: {rmse_values[index]}")

"""### **Actual VS Predicted WPI Values for ARMA**"""

plt.figure(figsize=(10, 6))
plt.plot(train_data.index, train_data, label='Training Data', color='blue')
plt.plot(test_data.index, test_data, label='Actual Test Data', color='green')
plt.plot(test_data.index, arma_forecast, label='ARMA Predicted', color='red', linestyle='dashed')
plt.xlabel('Year')
plt.ylabel('WPI')
plt.title('Actual vs Predicted WPI (ARMA)')
plt.legend()
plt.show()

comparison_df = pd.DataFrame({'Actual': test_data, 'Predicted': arma_forecast})
print(comparison_df)
```

```
"""### **Predicted Values of All Models VS Actual Values of Test Data**"""

plt.figure(figsize=(12, 6))
plt.plot(df.index, df['WPI'], label='Actual')
plt.plot(test_data.index, arma_forecast, label='ARMA')
plt.plot(test_data.index, xgb_forecast, label='XGBoost')
plt.plot(test_data.index, rf_forecast, label='Random Forest')
plt.plot(test_data.index[-len(test_predictions):], test_predictions, label='LSTM', color='orange')
plt.plot(test_data.index, hybrid_forecast, label='ARMA+LSTM', color='cyan')
plt.xlabel('Year')
plt.ylabel('WPI Value')
plt.legend()
plt.title('WPI Forecasting')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(test_data.index, test_data, label='Actual', color='black')
plt.plot(test_data.index, arma_forecast, label='ARMA', color='blue')
plt.plot(test_data.index, xgb_forecast, label='XGBoost', color='red')
plt.plot(test_data.index, rf_forecast, label='Random Forest', color='green')
plt.plot(test_data.index[-len(test_predictions):], test_predictions, label='LSTM', color='orange')
plt.plot(test_data.index, hybrid_forecast, label='ARMA+LSTM', color='cyan')

plt.xlabel('Year')
plt.ylabel('WPI')
plt.title('Forecasting WPI using Various Models')
plt.legend()
plt.grid(True)
plt.show()

model = ARIMA(df['WPI'], order=(3,0,1))
results = model.fit()

# Forecasting for the next 5 years
```

```
forecast_steps = 5
forecast = results.forecast(steps=forecast_steps)

last_year = df.index[-1]

forecast_index = range(last_year + 1, last_year + forecast_steps + 1)

forecast = np.array(forecast)

# 95% Confidence intervals
forecast_ci = results.get_forecast(steps=forecast_steps).conf_int(alpha=0.05)
lower_ci = forecast_ci.iloc[:, 0]
upper_ci = forecast_ci.iloc[:, 1]

# Plot
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['WPI'], label='Historical Data', marker='o')
plt.plot(forecast_index, forecast, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_index, lower_ci, upper_ci, color='red', alpha=0.2, label='95% Confidence Interval')

plt.title('WPI Forecast for Next 5 Years')
plt.xlabel('Year')
plt.ylabel('WPI Value')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Forecasted values
print("\nForecasted values for the next 5 years:")
for i, year in enumerate(forecast_index):
    print(f"Year {year}: {forecast[i]:.4f}")
```

References

https://www.geeksforgeeks.org/arma-time-series-model/

https://sdgs.un.org/goals

https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

https://www.analyticsvidhya.com/blog/2021/06/statistical-tests-to-check-stationarity-in-time-series-part-1/

https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling

https://www.geeksforgeeks.org/autocorrelation-and-partial-autocorrelation/

Group Members:

1. Pratik Rai
2. Aanshi Rathi
3. G. Samyukta
4. Mesam Laloo