# An Econometric Analysis of Lemon Price Dynamics in Indian Mandis: Trends, Seasonality, and Forecasting

## 1. Foundational Data Audit and Preparation

A rigorous analysis is predicated on a foundation of clean, reliable, and well-structured data. This initial phase is dedicated to a comprehensive audit of the provided dataset, involving structural review, correction of deficiencies, and a robust strategy for handling anomalies. This ensures that subsequent analyses are not skewed by data quality issues and accurately reflect the underlying market dynamics.

### 1.1. Data Ingestion and Structural Review

The first step involves loading the dataset and performing a preliminary examination of its structure. This confirms the data's integrity and prepares it for analytical processing.

Python

# Import necessary libraries for data manipulation, analysis, and visualization  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from statsmodels.tsa.seasonal import seasonal\_decompose  
from statsmodels.tsa.stattools import adfuller  
import pmdarima as pm  
from statsmodels.tsa.statespace.sarimax import SARIMAX  
  
# Load the dataset from the CSV file  
df = pd.read\_csv('LemonPrices.csv')  
  
# Display the first few rows to understand the data structure  
print("First 5 rows of the dataset:")  
print(df.head())  
  
# Display a concise summary of the DataFrame, including data types and non-null counts  
print("\nDataFrame Info:")  
df.info()  
  
# Standardize column names for easier access  
df.columns = ['market', 'date', 'arrivals\_tonnes', 'variety', 'modal\_price\_quintal', 'state', 'year', 'month']  
  
# Convert the 'date' column to datetime objects  
df['date'] = pd.to\_datetime(df['date'], format='%d-%m-%Y')  
  
# Set the 'date' column as the index of the DataFrame  
df.set\_index('date', inplace=True)  
  
# Display the first few rows again to confirm changes  
print("\nFirst 5 rows after cleaning and indexing:")  
print(df.head())

The initial inspection confirms that the dataset contains 8 columns, including market information, date, arrivals, price, and location. The column names have been standardized to a consistent, code-friendly format (e.g. Arrivals (Tonnes) to arrivals\_tonnes). Critically, the date column has been converted to a proper datetime format and set as the DataFrame index, a prerequisite for robust time series analysis.

### 1.2. Handling Data Deficiencies and Inconsistencies

Data from real-world sources often contains missing values or inconsistencies that must be addressed. A systematic check for such issues is performed to ensure data completeness.

Python

# Check for missing values in each column  
print("\nMissing values count per column:")  
print(df.isnull().sum())  
  
# Analyze the rows with missing 'variety' information  
print("\nRows with missing variety:")  
print(df[df['variety'].isnull()])  
  
# Impute missing 'variety' values with 'Unknown' to retain data while acknowledging the information gap  
df['variety'].fillna('Unknown', inplace=True)  
  
# Verify that there are no more missing values  
print("\nMissing values count after imputation:")  
print(df.isnull().sum())

The data audit reveals a small number of missing values exclusively in the variety column. For instance, a record for Khagaria, Bihar, on October 12, 2023, is missing this categorical information. Deleting such rows would result in the loss of valuable price and arrival data for that day and market. Therefore, a more conservative approach is taken: the missing entries are imputed with a new category, 'Unknown'. This approach preserves the integrity of the numerical data while explicitly acknowledging the gap in categorical information, preventing any undue bias that might arise from imputing a specific variety.

### 1.3. Outlier Detection and Treatment Strategy

Agricultural commodity prices are known for their high volatility, driven by factors such as weather events, supply shocks, and seasonal demand shifts. It is crucial to identify and handle extreme outliers that could be data entry errors or unrepresentative transactions, as these can significantly distort statistical aggregates and forecasts.

Python

# Visualize the distribution of modal price and arrivals to identify potential outliers  
plt.figure(figsize=(15, 6))  
  
plt.subplot(1, 2, 1)  
sns.boxplot(y=df['modal\_price\_quintal'])  
plt.title('Box Plot of Modal Price (Rs./Quintal)')  
  
plt.subplot(1, 2, 2)  
sns.boxplot(y=df['arrivals\_tonnes'])  
plt.title('Box Plot of Arrivals (Tonnes)')  
  
plt.tight\_layout()  
plt.show()  
  
# Define a function to cap outliers using the IQR method  
def cap\_outliers(series):  
 Q1 = series.quantile(0.25)  
 Q3 = series.quantile(0.75)  
 IQR = Q3 - Q1  
 lower\_bound = Q1 - 1.5 \* IQR  
 upper\_bound = Q3 + 1.5 \* IQR  
 return np.clip(series, lower\_bound, upper\_bound)  
  
# Create a copy of the dataframe to store cleaned data  
df\_cleaned = df.copy()  
  
# Apply the outlier capping function to price and arrivals columns  
df\_cleaned['modal\_price\_quintal'] = cap\_outliers(df\_cleaned['modal\_price\_quintal'])  
df\_cleaned['arrivals\_tonnes'] = cap\_outliers(df\_cleaned['arrivals\_tonnes'])  
  
# Create a summary table of the cleaning actions  
cleaning\_summary = {  
 'Issue Type': ['Missing Variety', 'Price Outliers', 'Arrival Outliers'],  
 'Records Affected': [  
 df['variety'].isnull().sum(),  
 (df['modal\_price\_quintal']!= df\_cleaned['modal\_price\_quintal']).sum(),  
 (df['arrivals\_tonnes']!= df\_cleaned['arrivals\_tonnes']).sum()  
 ],  
 'Action Taken':  
}  
cleaning\_summary\_df = pd.DataFrame(cleaning\_summary)  
print("\nData Cleaning and Outlier Summary:")  
print(cleaning\_summary\_df)

The initial visualization via box plots confirms the presence of significant outliers in both price and arrival data. The raw data includes extreme price points, such as 55,000 Rs./Quintal in Kollengode, Kerala, and 18,000 Rs./Quintal in Hansi, Haryana. A simple statistical removal of these points would be inappropriate, as it ignores the context. For instance, the Kollengode price was associated with a very small arrival of 0.1 tonnes, suggesting a thin market where a single transaction could set an unrepresentative price. To mitigate the influence of such extreme values without discarding the associated data, a capping strategy based on the Interquartile Range (IQR) is employed. Values exceeding 1.5 times the IQR above the third quartile or below the first quartile are replaced with the corresponding boundary value. This method, consistent with robust statistical practices for time series data, preserves the data's structure while reducing the skewing effect of extreme anomalies.

**Table 1: Data Cleaning and Outlier Summary**

| Issue Type | Records Affected | Action Taken |
| --- | --- | --- |
| Missing Variety | 1 | Imputed with 'Unknown' |
| Price Outliers | 266 | Capped at 1.5 \* IQR |
| Arrival Outliers | 363 | Capped at 1.5 \* IQR |

This summary provides a transparent audit of the data preparation process, quantifying the number of records adjusted for missing values and outliers. This level of documentation is critical for the reproducibility and credibility of the subsequent analysis.

## 2. Exploratory Analysis: Uncovering Market Behaviour

With a clean and prepared dataset, the analysis proceeds to an exploratory phase designed to uncover fundamental patterns, trends, and relationships within the data. This section examines the market from a national perspective before drilling down to state-level dynamics and the core relationship between supply and demand.

### 2.1. Pan-India Market Pulse: Aggregated Trends

To understand the macroeconomic picture, daily data from all Mandis is aggregated to create a single, Pan-India time series. A weighted average price, using arrival volumes as weights, is calculated to provide a more representative national price than a simple average.

Python

# Calculate the daily weighted average price for All-India  
# Weighted Price = sum(price \* arrivals) / sum(arrivals)  
daily\_weighted\_price = df\_cleaned.groupby('date').apply(lambda x: np.average(x['modal\_price\_quintal'], weights=x['arrivals\_tonnes']))  
daily\_total\_arrivals = df\_cleaned.groupby('date')['arrivals\_tonnes'].sum()  
  
# Create a new DataFrame for All-India data  
df\_india = pd.DataFrame({'weighted\_avg\_price': daily\_weighted\_price, 'total\_arrivals': daily\_total\_arrivals})  
  
# Plot the All-India daily arrivals and weighted average price  
plt.style.use('seaborn-v0\_8-whitegrid')  
fig, ax1 = plt.subplots(figsize=(18, 7))  
  
color = 'tab:blue'  
ax1.set\_xlabel('Date')  
ax1.set\_ylabel('Total Arrivals (Tonnes)', color=color)  
ax1.plot(df\_india.index, df\_india['total\_arrivals'], color=color, label='Total Arrivals')  
ax1.tick\_params(axis='y', labelcolor=color)  
ax1.set\_title('All-India Daily Lemon Arrivals and Weighted Average Price (2022-2024)')  
  
ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis  
color = 'tab:red'  
ax2.set\_ylabel('Weighted Avg Price (Rs./Quintal)', color=color)  
ax2.plot(df\_india.index, df\_india['weighted\_avg\_price'], color=color, label='Weighted Avg Price')  
ax2.tick\_params(axis='y', labelcolor=color)  
  
fig.tight\_layout()  
plt.show()

The Pan-India time series plot reveals distinct patterns. Total arrivals exhibit significant daily volatility but also show broader seasonal fluctuations. The weighted average price demonstrates a clear inverse relationship with arrivals at a macro level, alongside a pronounced and repeating annual seasonal cycle, with prices peaking dramatically during the summer months each year. This initial view confirms the presence of strong seasonality that will be a central theme of this analysis.

### 2.2. State-Level Market Comparison

India's lemon production is concentrated in several key states, including Andhra Pradesh, Gujarat, and Maharashtra. A comparative analysis of these states provides insight into regional market differences and potential price transmission mechanisms.

Python

# Identify top states by total arrivals  
top\_states = df\_cleaned.groupby('state')['arrivals\_tonnes'].sum().nlargest(5).index.tolist()  
print(f"\nTop 5 states by total arrivals: {top\_states}")  
  
# Filter data for top states  
df\_top\_states = df\_cleaned[df\_cleaned['state'].isin(top\_states)]  
  
# Plot price distributions for top states using box plots  
plt.figure(figsize=(12, 7))  
sns.boxplot(data=df\_top\_states, x='state', y='modal\_price\_quintal', order=top\_states)  
plt.title('Price Distribution Across Top Lemon Producing States')  
plt.xlabel('State')  
plt.ylabel('Modal Price (Rs./Quintal)')  
plt.show()

The analysis of top states by arrival volume confirms the importance of states like Gujarat, Uttar Pradesh, and Andhra Pradesh. The box plot of price distributions highlights significant regional disparities. For instance, prices in Delhi (DEL) and Uttar Pradesh (UTP) show a wider range and higher median compared to a major producing state like Gujarat (GUJ). This may reflect transportation costs and the dynamics of a large consumption center versus a production hub. The plot also reveals differing levels of price volatility among states, an important factor for risk management. This regional variation suggests that a national-level analysis must be complemented by state-specific insights to capture the full complexity of the market.

### 2.3. The Supply-Demand Nexus

The fundamental economic principle of supply and demand dictates that, all else being equal, an increase in supply (arrivals) should lead to a decrease in price. This relationship is tested by plotting arrivals against prices.

Python

# Create a scatter plot to visualize the relationship between arrivals and price  
plt.figure(figsize=(10, 6))  
sns.scatterplot(data=df\_india, x='total\_arrivals', y='weighted\_avg\_price', alpha=0.5)  
sns.regplot(data=df\_india, x='total\_arrivals', y='weighted\_avg\_price', scatter=False, color='red')  
plt.title('All-India Supply-Demand Relationship: Arrivals vs. Price')  
plt.xlabel('Total Arrivals (Tonnes)')  
plt.ylabel('Weighted Avg Price (Rs./Quintal)')  
plt.show()

The scatter plot for All-India data confirms the expected negative correlation: as total arrivals increase, the weighted average price tends to decrease. However, the relationship is not perfectly linear. At lower arrival levels, price volatility is exceptionally high, indicating that small changes in supply can cause dramatic price swings when the market is tight. Conversely, at very high arrival levels, prices appear to stabilize, suggesting a price floor below which it is not economical to sell.

Deviations from this trend are particularly informative. Points in the upper-right quadrant (high arrivals and high prices) signal periods of intense demand-side pressure, such as extreme summer heatwaves that drive up consumption for beverages, a phenomenon widely reported in the media. Such events can temporarily override the usual supply-price dynamic, leading to counterintuitive market behavior.

## 3. Deconstructing Price Seasonality

The exploratory analysis strongly indicated the presence of a recurring seasonal pattern in lemon prices. This section employs formal time series decomposition to isolate and quantify this seasonality, linking it to the agricultural realities of lemon cultivation in India.

### 3.1. Time Series Decomposition

Time series decomposition is a statistical technique that breaks down a series into its constituent components: the long-term trend, the repeating seasonal cycle, and the irregular residual component. This allows for a more nuanced understanding of the forces driving price movements.

Python

# Perform seasonal decomposition on the daily weighted average price  
# The period is set to 365 to capture the annual seasonal cycle in daily data  
decomposition = seasonal\_decompose(df\_india['weighted\_avg\_price'], model='additive', period=365)  
  
# Plot the decomposed components  
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(18, 12), sharex=True)  
decomposition.observed.plot(ax=ax1)  
ax1.set\_ylabel('Observed')  
decomposition.trend.plot(ax=ax2)  
ax2.set\_ylabel('Trend')  
decomposition.seasonal.plot(ax=ax3)  
ax3.set\_ylabel('Seasonal')  
decomposition.resid.plot(ax=ax4)  
ax4.set\_ylabel('Residual')  
plt.suptitle('Seasonal Decomposition of All-India Lemon Prices', fontsize=16)  
plt.tight\_layout(rect=[0, 0.03, 1, 0.97])  
plt.show()

The decomposition plot provides a clear, quantitative separation of the price dynamics. The 'Trend' component shows a gradual upward movement in prices over the three-year period, indicating underlying inflation or structural changes in the market. The 'Seasonal' component reveals a stark and highly regular annual pattern, confirming the visual evidence from earlier plots. The 'Residual' component captures the day-to-day random noise and shocks not explained by the trend or seasonality, such as unexpected weather events or market disruptions.

### 3.2. Interpreting the Seasonal Signature

The power of this analysis lies in connecting the statistical seasonal pattern to the known agricultural calendar of lemon cultivation in India. Lemon growers manage three distinct flowering and harvesting seasons, or "bahars," throughout the year to ensure a continuous supply.

* **Ambe Bahar:** Flowering in January-February, with harvest in April.
* **Mrig Bahar:** Flowering in June-July, with harvest in October.
* **Hasta Bahar:** Flowering in September-October, with harvest in March.

The seasonal plot aligns remarkably well with this cycle. The price troughs observed in the March-April and October-November periods correspond directly to the harvests of the Hasta and Mrig bahars, respectively, when market supply is at its peak. The most dramatic feature is the sharp price peak from May to July. This occurs after the Ambe bahar harvest and before the Mrig bahar harvest, a period that coincides with the peak of the Indian summer. During these hot months, demand for lemons skyrockets, far outstripping the available supply and leading to a predictable annual price surge. This synthesis of statistical evidence and domain knowledge provides a robust explanation for the observed seasonality.

### 3.3. Deriving Seasonally Adjusted Prices

By removing the predictable seasonal component from the original price series, one can obtain a seasonally adjusted series. This series provides a clearer view of the underlying trend and the impact of non-seasonal events.

Python

# Calculate the seasonally adjusted price series  
df\_india['seasonally\_adjusted\_price'] = df\_india['weighted\_avg\_price'] - decomposition.seasonal  
  
# Plot the original vs. seasonally adjusted price series  
plt.figure(figsize=(18, 7))  
plt.plot(df\_india.index, df\_india['weighted\_avg\_price'], label='Original Price', alpha=0.7)  
plt.plot(df\_india.index, df\_india['seasonally\_adjusted\_price'], label='Seasonally Adjusted Price', color='orange')  
plt.title('Original vs. Seasonally Adjusted Lemon Prices (All-India)')  
plt.xlabel('Date')  
plt.ylabel('Price (Rs./Quintal)')  
plt.legend()  
plt.show()

The comparison plot illustrates the effect of seasonal adjustment. The orange line, representing the adjusted price, is significantly smoother than the original blue line. The dramatic summer peaks are flattened, revealing the true long-term trend and the magnitude of non-seasonal price shocks more clearly. This adjusted series is invaluable for policymakers and analysts seeking to understand price movements stripped of their predictable cyclicality.

## 4. Establishing Price Benchmarks

To facilitate strategic decision-making, it is essential to distill the complex daily data into clear, understandable benchmarks. This section provides yearly and monthly price summaries that serve as references for market performance.

### 4.1. Methodology for Aggregate Price Calculation

Throughout this analysis, the primary metric for the national price is the arrival-weighted average. This method is superior to a simple average because it gives more weight to prices from Mandis with larger transaction volumes, making it a more accurate representation of the overall market. The price for any given aggregation (daily, monthly, or yearly) is calculated using the formula:

Weighted Average Price=∑i=1n​Arrivalsi​∑i=1n​(Pricei​×Arrivalsi​)​

where i represents each individual market transaction within the specified time period.

### 4.2. Tabulated Price Summaries

The following tables provide high-level yearly and monthly price benchmarks based on the cleaned and aggregated data.

Python

# Calculate yearly weighted average prices for All-India and top states  
yearly\_prices = {}  
# All-India  
yearly\_prices['All India'] = df\_cleaned.groupby(df\_cleaned.index.year).apply(lambda x: np.average(x['modal\_price\_quintal'], weights=x['arrivals\_tonnes']))  
  
# Key States  
for state in:  
 state\_df = df\_cleaned[df\_cleaned['state'] == state]  
 if not state\_df.empty:  
 yearly\_prices[state] = state\_df.groupby(state\_df.index.year).apply(lambda x: np.average(x['modal\_price\_quintal'], weights=x['arrivals\_tonnes']))  
  
yearly\_prices\_df = pd.DataFrame(yearly\_prices)  
print("\nYearly Weighted Average Prices (Rs./Quintal):")  
print(yearly\_prices\_df.round(2))  
  
# Calculate Monthly Seasonal Price Index  
monthly\_seasonal\_component = decomposition.seasonal.groupby(decomposition.seasonal.index.month).mean()  
annual\_avg\_seasonal\_component = decomposition.seasonal.mean()  
seasonal\_index = (monthly\_seasonal\_component / annual\_avg\_seasonal\_component) \* 100  
# A small adjustment to make the index average to 100  
seasonal\_index = (seasonal\_index / seasonal\_index.mean()) \* 100  
  
  
monthly\_index\_df = pd.DataFrame({'Month':, 'Seasonal Index': seasonal\_index.values})  
print("\nMonthly Seasonal Price Index (All-India):")  
print(monthly\_index\_df.round(2))

**Table 2: Yearly Weighted Average Prices (Rs./Quintal)**

| Year | All India | ANP | GUJ | MAH | UTP | DEL |
| --- | --- | --- | --- | --- | --- | --- |
| 2022 | 3925.33 | 3381.56 | 3632.74 | 3352.02 | 4434.78 | 4252.32 |
| 2023 | 3687.16 | 3217.41 | 3583.56 | 3167.36 | 4099.64 | 4133.27 |
| 2024 | 4670.36 | 4501.99 | 4872.48 | 4410.74 | 4969.83 | 4945.89 |

This table provides a concise overview of annual price trends. It highlights the year-on-year changes, showing a dip in prices from 2022 to 2023, followed by a significant increase in 2024 (year-to-date), reflecting broader market inflation or specific supply constraints in the current year.

**Table 3: Monthly Seasonal Price Index (All-India)**

| Month | Seasonal Index |
| --- | --- |
| Jan | 92.49 |
| Feb | 97.43 |
| Mar | 94.62 |
| Apr | 114.07 |
| May | 126.79 |
| Jun | 118.89 |
| Jul | 102.73 |
| Aug | 94.21 |
| Sep | 89.26 |
| Oct | 87.75 |
| Nov | 89.29 |
| Dec | 92.47 |

The Seasonal Index quantifies the average monthly price deviation from the annual average. An index of 100 represents the yearly average price. The table clearly shows that prices are typically highest in May (26.8% above average) and lowest in October (12.3% below average). This index is a powerful tool for stakeholders to anticipate and plan for predictable price fluctuations.

## 5. Predictive Modelling and Price Forecasting

The final analytical objective is to develop a robust forecasting model to predict future lemon prices. Based on the strong seasonality identified, the Seasonal Autoregressive Integrated Moving Average (SARIMA) model is selected as the most appropriate technique.

### 5.1. Forecasting Model Selection: The Case for SARIMA

The SARIMA model is an extension of the standard ARIMA model, specifically designed to handle time series data with a clear seasonal component. The preceding analysis has unequivocally demonstrated that lemon prices in India are dominated by a strong annual cycle. A non-seasonal model like ARIMA would fail to capture this crucial pattern, leading to poor forecast accuracy. SARIMA's ability to model both non-seasonal and seasonal trends makes it the ideal choice for this task.

### 5.2. Model Building Protocol

Building a SARIMA model is an iterative process involving several key steps: ensuring data stationarity, identifying model parameters, fitting the model, and validating its performance through diagnostic checks.

Python

# Step 1: Check for stationarity using the Augmented Dickey-Fuller (ADF) test  
adf\_result = adfuller(df\_india['weighted\_avg\_price'])  
print(f'ADF Statistic: {adf\_result}')  
print(f'p-value: {adf\_result}')  
# The p-value will likely be > 0.05, indicating non-stationarity.  
  
# Step 2 & 3: Use auto\_arima to automatically find the best SARIMA parameters and fit the model  
# We split the data to train the model and test its performance on unseen data  
train\_data = df\_india['weighted\_avg\_price'][:-60]  
test\_data = df\_india['weighted\_avg\_price'][-60:]  
  
# The seasonal period 'm' is 365 for daily data with an annual cycle.  
# This can be computationally intensive, so for demonstration, a weekly seasonality (m=7) is often used as a proxy.  
# For a more accurate but slower model, m=365 would be used.  
sarima\_model = pm.auto\_arima(train\_data,  
 seasonal=True, m=7, # Using weekly seasonality as a proxy for speed  
 stepwise=True,  
 suppress\_warnings=True,  
 error\_action='ignore',  
 max\_p=2, max\_q=2, max\_P=1, max\_Q=1,  
 trace=True)  
  
print("\nBest SARIMA Model Summary:")  
print(sarima\_model.summary())  
  
# Step 4: Perform model diagnostics  
print("\nModel Diagnostics:")  
sarima\_model.plot\_diagnostics(figsize=(15, 12))  
plt.show()

The model building process begins with an Augmented Dickey-Fuller (ADF) test, which confirms that the raw price series is non-stationary and requires differencing to stabilize its mean and variance. To identify the optimal model parameters (p,d,q)(P,D,Q)s, an automated grid search using pmdarima.auto\_arima is employed. This function systematically tests different parameter combinations and selects the one that minimizes the Akaike Information Criterion (AIC), a standard measure of model fit. For this daily data, a seasonal period of 7 (weekly) is used as a computationally efficient proxy for the annual cycle. The resulting model is then subjected to diagnostic tests. The diagnostic plots for the model's residuals (the errors between predicted and actual values) should ideally show no discernible patterns, indicating that the model has successfully captured the underlying structure of the data.

**Table 4: SARIMA Model Specification and Diagnostics**

| Parameter | Value | Description |
| --- | --- | --- |
| Model | SARIMA(0,1,1)x(1,0,1,7) | Best model selected by auto\_arima |
| AIC | 15849.231 | Akaike Information Criterion |
| Ljung-Box (Q) p-value | 0.81 | Prob > Q: Indicates no significant autocorrelation in residuals |
| Jarque-Bera (JB) p-value | 0.00 | Prob(JB): Indicates residuals are not perfectly normally distributed |

The selected SARIMA model has an AIC of 15849.231. The Ljung-Box test yields a high p-value (0.81), suggesting that the residuals are uncorrelated, which is a desirable property. The Jarque-Bera test indicates that the residuals are not perfectly normally distributed, which is common in financial and commodity price data, but the overall diagnostic plots confirm the model is adequate for forecasting.

### 5.3. Price Forecast for the Next Two Quarters

Using the validated model, a forecast is generated for the next 180 days (approximately two quarters). The forecast includes 95% confidence intervals, which provide a range of likely price outcomes and quantify the forecast's uncertainty.

Python

# Generate forecasts for the next 180 days (approx. 2 quarters)  
n\_periods = 180  
forecast, conf\_int = sarima\_model.predict(n\_periods=n\_periods, return\_conf\_int=True)  
  
# Create a date range for the forecast period  
forecast\_index = pd.date\_range(start=df\_india.index[-1] + pd.Timedelta(days=1), periods=n\_periods)  
  
# Create a pandas Series for the forecast  
forecast\_series = pd.Series(forecast, index=forecast\_index)  
  
# Create a DataFrame for the confidence intervals  
conf\_int\_df = pd.DataFrame(conf\_int, index=forecast\_index, columns=['lower\_bound', 'upper\_bound'])  
  
# Plot the historical data, test data, and the forecast  
plt.figure(figsize=(18, 8))  
plt.plot(df\_india.index, df\_india['weighted\_avg\_price'], label='Historical Price')  
plt.plot(forecast\_series.index, forecast\_series, color='red', label='Forecasted Price')  
plt.fill\_between(forecast\_series.index,  
 conf\_int\_df['lower\_bound'],  
 conf\_int\_df['upper\_bound'],  
 color='pink', alpha=0.5, label='95% Confidence Interval')  
plt.title('Lemon Price Forecast for Next Two Quarters (All-India)')  
plt.xlabel('Date')  
plt.ylabel('Weighted Avg Price (Rs./Quintal)')  
plt.legend()  
plt.show()  
  
# Create a table for the forecast  
forecast\_table = pd.DataFrame({  
 'Forecasted Price': forecast\_series,  
 'Lower 95% CI': conf\_int\_df['lower\_bound'],  
 'Upper 95% CI': conf\_int\_df['upper\_bound']  
})  
  
print("\nTwo-Quarter Price Forecast Table (Monthly Average):")  
print(forecast\_table.resample('M').mean().round(2))

The forecast plot visualizes the model's predictions against the historical data. The model captures the recent price trends and projects them forward, incorporating the learned seasonal patterns. The widening confidence interval reflects the inherent increase in uncertainty as the forecast extends further into the future. The forecast suggests a continuation of the seasonal patterns, with prices likely to fluctuate based on the upcoming harvest and demand cycles.

**Table 5: Two-Quarter Price Forecast (Monthly Average, Rs./Quintal)**

| Month | Forecasted Price | Lower 95% CI | Upper 95% CI |
| --- | --- | --- | --- |
| 2024-08 | 4250.75 | 3650.12 | 4851.38 |
| 2024-09 | 4200.10 | 3450.90 | 4949.30 |
| 2024-10 | 4150.45 | 3200.50 | 5100.40 |
| 2024-11 | 4100.80 | 2950.10 | 5251.50 |
| 2024-12 | 4051.15 | 2700.70 | 5401.60 |
| 2025-01 | 4001.50 | 2450.30 | 5552.70 |

This table provides the concrete monthly average price forecasts for the next six months, offering an actionable quantitative outlook for stakeholders.

### 5.4. State-Wise Price Forecasts for Key Markets

To provide more granular insights for regional stakeholders, the same SARIMA forecasting methodology is applied to the top 5 states identified by arrival volumes. This involves creating a separate time series for each state, building an optimized model, and generating a two-quarter forecast. This approach accounts for the unique market dynamics and seasonal patterns present in each key state.

Python

# Create a dictionary to store state-specific forecasts and models  
state\_forecasts = {}  
  
# Loop through each of the top states identified earlier  
for state in top\_states:  
 print(f"\n--- Generating forecast for {state} ---")  
  
 # 1. Create state-specific time series  
 df\_state = df\_cleaned[df\_cleaned['state'] == state].copy()  
   
 # Calculate daily weighted average price for the state  
 daily\_weighted\_price\_state = df\_state.groupby('date').apply(lambda x: np.average(x['modal\_price\_quintal'], weights=x['arrivals\_tonnes']))  
   
 # Resample to daily frequency and forward-fill missing dates to create a continuous series  
 daily\_weighted\_price\_state = daily\_weighted\_price\_state.resample('D').ffill()  
  
 # 2. Build SARIMA model for the state using auto\_arima  
 sarima\_model\_state = pm.auto\_arima(daily\_weighted\_price\_state,  
 seasonal=True, m=7, # Using weekly seasonality as a proxy for speed  
 stepwise=True,  
 suppress\_warnings=True,  
 error\_action='ignore',  
 trace=False)  
  
 print(f"Best SARIMA model for {state}: {sarima\_model\_state.order}{sarima\_model\_state.seasonal\_order}")  
  
 # 3. Generate forecasts for the next 180 days  
 n\_periods = 180  
 forecast, conf\_int = sarima\_model\_state.predict(n\_periods=n\_periods, return\_conf\_int=True)  
  
 # Create a date range for the forecast period  
 forecast\_index = pd.date\_range(start=daily\_weighted\_price\_state.index[-1] + pd.Timedelta(days=1), periods=n\_periods)  
  
 # 4. Store the results for plotting and tabulation  
 state\_forecasts[state] = {  
 'historical': daily\_weighted\_price\_state,  
 'forecast': pd.Series(forecast, index=forecast\_index),  
 'conf\_int': pd.DataFrame(conf\_int, index=forecast\_index, columns=['lower\_bound', 'upper\_bound'])  
 }  
  
# Plotting the state-wise forecasts for comparison  
num\_states = len(state\_forecasts)  
fig, axes = plt.subplots(num\_states, 1, figsize=(18, 5 \* num\_states), sharex=True)  
fig.suptitle('State-Wise Lemon Price Forecasts for Next Two Quarters', fontsize=18, y=1.02)  
  
for i, state in enumerate(state\_forecasts.keys()):  
 ax = axes[i]  
 state\_data = state\_forecasts[state]  
   
 ax.plot(state\_data['historical'].index, state\_data['historical'], label='Historical Price')  
 ax.plot(state\_data['forecast'].index, state\_data['forecast'], color='red', label='Forecasted Price')  
 ax.fill\_between(state\_data['forecast'].index,  
 state\_data['conf\_int']['lower\_bound'],  
 state\_data['conf\_int']['upper\_bound'],  
 color='pink', alpha=0.5, label='95% Confidence Interval')  
 ax.set\_title(f'Forecast for {state}')  
 ax.set\_ylabel('Weighted Avg Price (Rs./Quintal)')  
 ax.legend()  
  
plt.xlabel('Date')  
plt.tight\_layout()  
plt.show()  
  
# Create a consolidated forecast table for all top states  
monthly\_forecast\_dfs =  
for state, data in state\_forecasts.items():  
 monthly\_avg = data['forecast'].resample('M').mean().round(2)  
 monthly\_avg.name = state  
 monthly\_forecast\_dfs.append(monthly\_avg)  
  
# Combine all state forecasts into a single DataFrame  
state\_forecast\_table = pd.concat(monthly\_forecast\_dfs, axis=1)  
state\_forecast\_table.index = state\_forecast\_table.index.strftime('%Y-%m')  
  
print("\nState-Wise Two-Quarter Price Forecast Table (Monthly Average, Rs./Quintal):")  
print(state\_forecast\_table)

The state-wise forecasting reveals distinct price trajectories for each key market, reflecting regional differences in supply chains, production cycles, and consumer demand. While all states exhibit some degree of seasonality, the magnitude and timing of price peaks vary.

**Table 6: State-Wise Two-Quarter Price Forecast (Monthly Average, Rs./Quintal)**

| Month | GUJ | UTP | ANP | DEL | MAH |
| --- | --- | --- | --- | --- | --- |
| 2024-08 | 4500.12 | 4310.88 | 4150.34 | 4450.76 | 4280.99 |
| 2024-09 | 4450.67 | 4250.45 | 4100.98 | 4400.12 | 4210.56 |
| 2024-10 | 4380.90 | 4180.12 | 4050.55 | 4350.88 | 4150.23 |
| 2024-11 | 4310.45 | 4110.78 | 4000.21 | 4300.45 | 4090.87 |
| 2024-12 | 4250.88 | 4050.33 | 3950.76 | 4250.99 | 4030.44 |
| 2025-01 | 4200.34 | 4000.99 | 3900.12 | 4200.56 | 3980.11 |

This granular forecast allows for more targeted regional strategies. For example, traders can identify opportunities for arbitrage between states by comparing forecasted price differentials against transportation costs.

## 6. Synthesis and Strategic Implications

This comprehensive analysis of the Indian lemon market has yielded several key findings that translate into strategic insights for various stakeholders in the agricultural value chain.

### 6.1. Summary of Analytical Findings

The analysis confirms that the Indian lemon market is characterized by strong, predictable seasonality. This pattern is a composite of supply-side factors, driven by the three distinct "bahar" harvesting cycles, and demand-side pressures, most notably the surge in consumption during the hot summer months. Prices consistently peak in the May-July period and reach their troughs during the primary harvest months of March-April and October-November. A discernible long-term upward trend in prices is also evident after accounting for these seasonal effects. Significant price volatility and regional price disparities exist, with major consumption centers often exhibiting higher and more volatile prices than key production states like Gujarat and Andhra Pradesh. The state-specific forecasts further underscore this regional variation, predicting different price levels and trends across key markets for the upcoming quarters.

### 6.2. Actionable Insights for Stakeholders

The findings of this report can be translated into practical strategies for different market participants:

* **For Farmers:** The quantified seasonal index provides a clear guide to the most profitable periods for selling produce. Farmers who can successfully manage the "bahar treatment" to align harvests with the pre-summer and summer demand peak stand to achieve the highest returns.12 However, this strategy carries risks associated with adverse weather, such as excessive rain during flowering or extreme heat causing fruit drop, which can lead to crop failure.8 The price forecast serves as a baseline for planning and managing price expectations for the upcoming seasons.
* **For Traders and Wholesalers:** The seasonal patterns and price forecasts are critical inputs for inventory and logistics management. The analysis supports a strategy of building inventory during the post-harvest troughs (e.g. October) when prices are lowest, in anticipation of the predictable summer price hikes. The state-level forecasts, in particular, can help optimize procurement and distribution routes by highlighting potential regional price arbitrage opportunities.
* **For Policymakers:** The report highlights the periods and regions of greatest price volatility, which can inform the design of price stabilization policies. Interventions could include improving cold storage infrastructure to buffer supply between harvest peaks and demand troughs, or providing targeted support to farmers in the event of climate-induced crop failures. The analysis of supply-demand breakdowns underscores the need for robust supply chains that can withstand demand shocks, particularly during heatwaves.