TIME SERIES FORECASTING USING MACHINE LEARNING

MARKET PRICE PREDICTION

Prashant Pal
Machine Learning Intern- Mentorness

Name- Prashant Pal

(Machine Learning Intern - Mentorness)

Batch – MIP-ML-10

Problem Statement: Market Price Prediction

Background: In the realm of market analysis and forecasting, understanding the intricate patterns within time series data is paramount for informed decision-making. With the advent of machine learning techniques, it's now possible to delve deeper into historical market data to predict future trends accurately. In this context, we have at our disposal a dataset containing monthly market data spanning multiple years, encompassing various regions, commodities, and pricing information.

Objective: The primary objective of this project is to develop a robust time series machine learning model capable of accurately forecasting market trends based on historical data. By leveraging advanced algorithms, we aim to predict the quantity and prices of commodities for future months, empowering stakeholders to make proactive decisions regarding production, procurement, pricing strategies, and resource allocation.

Data Description:

The dataset comprises the following columns:

- month: The month for which the data is recorded.
- year: The year corresponding to the recorded data.
- quantity: The quantity of the commodity traded or available.
- priceMin: The minimum price of the commodity during the month
- priceMax: The maximum price of the commodity during the month.
- priceMax: The maximum price of the commodity during the month.
- priceMod: The mode or most frequently occurring price of the commodity during the month.
- state: The state or region where the market is located.
- city: The city where the market is situated
- date: The specific date of the recorded data.

Task:

The task involves several key steps:

1. Data Preprocessing: Cleaning the dataset, handling missing values, and encoding categorical variables.
2. Exploratory Data Analysis (EDA): Analyzing the temporal patterns, identifying seasonality, trends, and anomalies within the data.
3. Feature Engineering: Creating relevant features such as lagged variables, rolling statistics, and seasonal indicators.
4. Model Selection and Training: Evaluating various time series forecasting models such as ARIMA, SARIMA, Prophet, and LSTM, selecting the most suitable one, and training it on the dataset.
5. Model Evaluation: Assessing the model's performance using appropriate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).
6. Fine-tuning and Validation: Fine-tuning the model parameters, validating its performance on unseen data, and iterating if necessary.

Outcome: The ultimate goal is to deploy a production-ready machine learning model capable of generating accurate forecasts for market quantity and prices for future months. The insights derived from this model will aid stakeholders in making data-driven decisions, optimizing inventory management, pricing strategies, and resource allocation, thereby enhancing overall efficiency and profitability in the market ecosystem.

```
#Importing necessary Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
import plotly.graph_objs as go
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.statespace.sarimax import SARIMAX


df=pd.read_csv('/content/drive/MyDrive/MarketPricePrediction.csv')   #Readin MarketPricePrediction.csv file
```

Data Preprocessing

```
df.head()
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city	date
0	ABOHAR(PB)	January	2005	2350	404	493	446	PB	ABOHAR	January-2005
1	ABOHAR(PB)	January	2006	900	487	638	563	PB	ABOHAR	January-2006
2	ABOHAR(PB)	January	2010	790	1283	1592	1460	PB	ABOHAR	January-2010
3	ABOHAR(PB)	January	2011	245	3067	3750	3433	PB	ABOHAR	January-2011
4	ABOHAR(PB)	January	2012	1035	523	686	605	PB	ABOHAR	January-2012

```
df.tail()
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city	date
10222	YEOLA(MS)	December	2011	131326	282	612	526	MS	YEOLA	December-2011
10223	YEOLA(MS)	December	2012	207066	485	1327	1136	MS	YEOLA	December-2012
10224	YEOLA(MS)	December	2013	215883	472	1427	1177	MS	YEOLA	December-2013
10225	YEOLA(MS)	December	2014	201077	446	1654	1456	MS	YEOLA	December-2014
10226	YEOLA(MS)	December	2015	223315	609	1446	1126	MS	YEOLA	December-2015

Finding missing values

```
print(df.isnull().sum())  # None missing values found
```

```
market      0
month       0
year        0
quantity    0
priceMin    0
priceMax    0
priceMod    0
state       0
city        0
date        0
dtype: int64
```

```
# # Encode categorical variables
# label_encoders = {}
# categorical_columns = ['market', 'state', 'city']
# for column in categorical_columns:
#     le = LabelEncoder()
#     df[column] = le.fit_transform(df[column])
#     label_encoders[column] = le


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10227 entries, 0 to 10226
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
```

```
---  ------    --------------  -----
 0   market    10227 non-null  object
 1   month     10227 non-null  object
 2   year      10227 non-null  int64
 3   quantity  10227 non-null  int64
 4   priceMin  10227 non-null  int64
 5   priceMax  10227 non-null  int64
 6   priceMod  10227 non-null  int64
 7   state     10227 non-null  object
 8   city      10227 non-null  object
 9   date      10227 non-null  object
dtypes: int64(5), object(5)
memory usage: 799.1+ KB
```

```
df.columns=df.columns.str.lower() # converting all columns name in lower string
df
```

	market	month	year	quantity	pricemin	pricemax	pricemod	state	city	date
0	ABOHAR(PB)	January	2005	2350	404	493	446	PB	ABOHAR	January-2005
1	ABOHAR(PB)	January	2006	900	487	638	563	PB	ABOHAR	January-2006
2	ABOHAR(PB)	January	2010	790	1283	1592	1460	PB	ABOHAR	January-2010
3	ABOHAR(PB)	January	2011	245	3067	3750	3433	PB	ABOHAR	January-2011
4	ABOHAR(PB)	January	2012	1035	523	686	605	PB	ABOHAR	January-2012
...	...	...	...	...	...	...	...	...	...	...
10222	YEOLA(MS)	December	2011	131326	282	612	526	MS	YEOLA	December-2011
10223	YEOLA(MS)	December	2012	207066	485	1327	1136	MS	YEOLA	December-2012
10224	YEOLA(MS)	December	2013	215883	472	1427	1177	MS	YEOLA	December-2013
10225	YEOLA(MS)	December	2014	201077	446	1654	1456	MS	YEOLA	December-2014
10226	YEOLA(MS)	December	2015	223315	609	1446	1126	MS	YEOLA	December-2015

10227 rows × 10 columns

```
# Convert date column to datetime

df['date'] = pd.to_datetime(df['date'])
df.tail()
```

	market	month	year	quantity	pricemin	pricemax	pricemod	state	city	date
10222	YEOLA(MS)	December	2011	131326	282	612	526	MS	YEOLA	2011-12-01
10223	YEOLA(MS)	December	2012	207066	485	1327	1136	MS	YEOLA	2012-12-01
10224	YEOLA(MS)	December	2013	215883	472	1427	1177	MS	YEOLA	2013-12-01
10225	YEOLA(MS)	December	2014	201077	446	1654	1456	MS	YEOLA	2014-12-01
10226	YEOLA(MS)	December	2015	223315	609	1446	1126	MS	YEOLA	2015-12-01

```
df.info() # checking whether date column's data type has changed to datetime64 or not
```

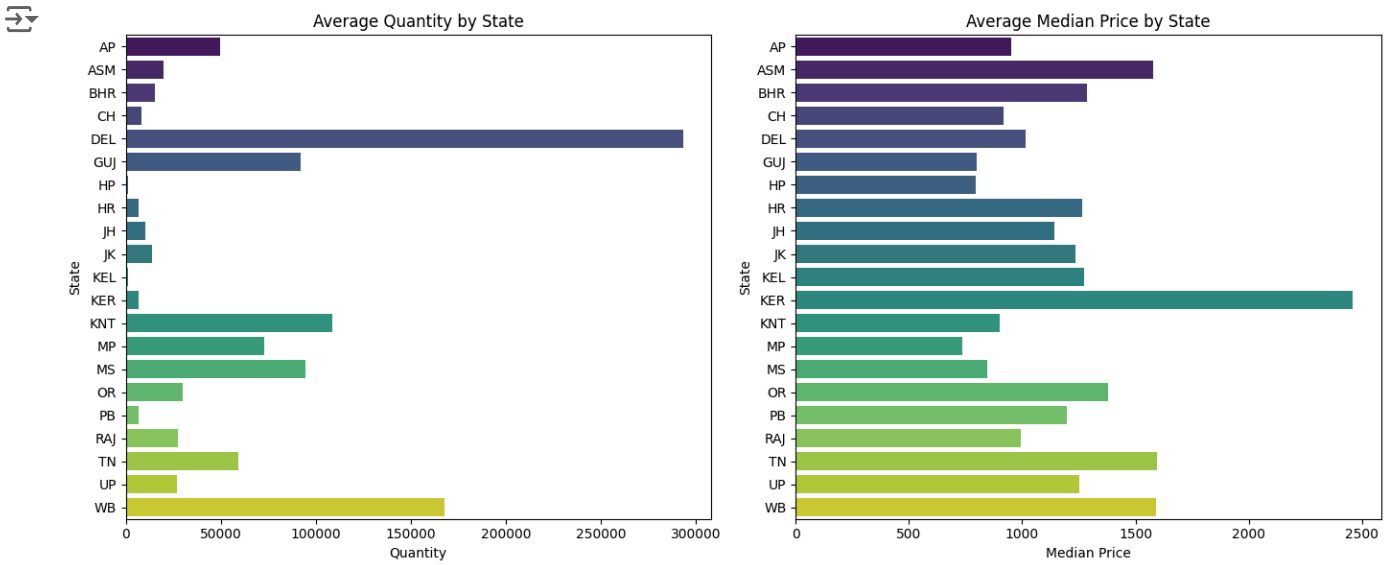
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10227 entries, 0 to 10226
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   market    10227 non-null  object
 1   month     10227 non-null  object
 2   year      10227 non-null  int64
 3   quantity  10227 non-null  int64
 4   pricemin  10227 non-null  int64
 5   pricemax  10227 non-null  int64
 6   pricemod  10227 non-null  int64
 7   state     10227 non-null  object
 8   city      10227 non-null  object
 9   date      10227 non-null  datetime64[ns]
dtypes: datetime64[ns](1), int64(5), object(4)
memory usage: 799.1+ KB
```

```
# Set date as index

df.set_index('date', inplace=True)
df.head()
```

Exploratory Data Analysis

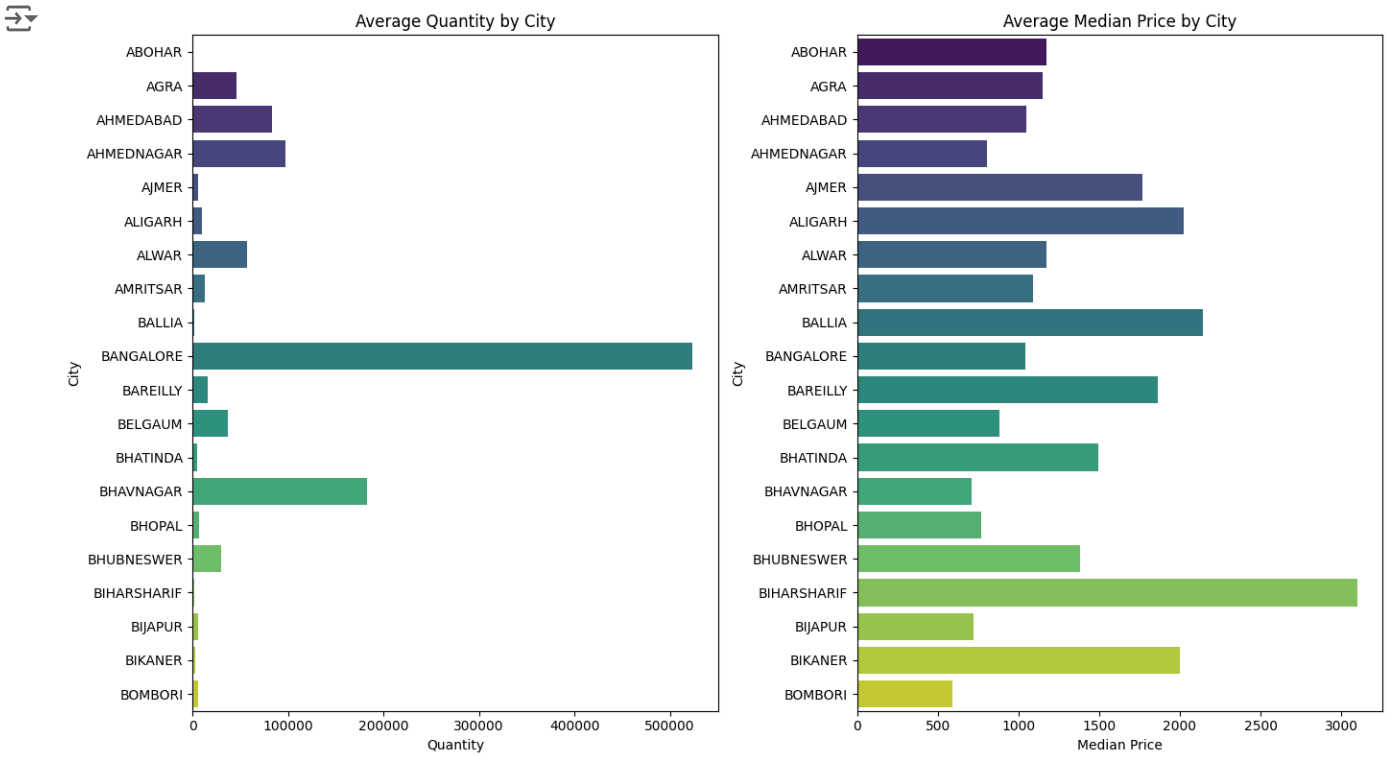
1. Average Quantity and Median Price by State

```
# Calculate mean quantity and median price by state
state_agg = df.groupby('state')[['quantity', 'median_price']].mean().reset_index()

# Plotting figure
plt.figure(figsize=(14, 6))

# Quantity by State
plt.subplot(1, 2, 1)
sns.barplot(x='quantity', y='state', data=state_agg, palette='viridis')
plt.title('Average Quantity by State')
plt.xlabel('Quantity')
plt.ylabel('State')

# Median Price by State
plt.subplot(1, 2, 2)
sns.barplot(x='median_price', y='state', data=state_agg, palette='viridis')
plt.title('Average Median Price by State')
plt.xlabel('Median Price')
plt.ylabel('State')

plt.tight_layout()
plt.show()
```



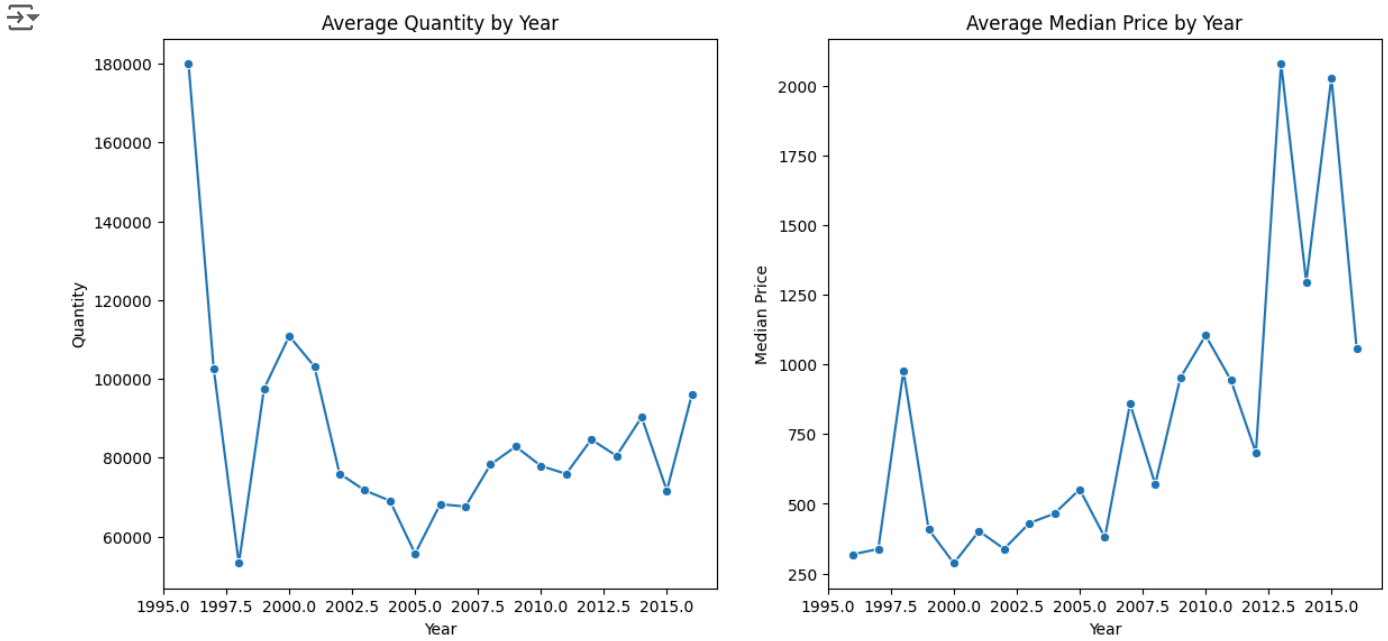
2. Average Quantity and Median Price by City

```
# Calculate mean quantity and median price by city
city_agg = df.groupby('city')[['quantity', 'median_price']].mean().reset_index()

#  Plotting figure
plt.figure(figsize=(14, 8))

# Quantity by City
plt.subplot(1, 2, 1)
sns.barplot(x='quantity', y='city', data=city_agg.head(20), palette='viridis')  # Limiting to top 20 cities for reada
plt.title('Average Quantity by City')
plt.xlabel('Quantity')
plt.ylabel('City')

# Median Price by City
plt.subplot(1, 2, 2)
sns.barplot(x='median_price', y='city', data=city_agg.head(20), palette='viridis')  # Limiting to top 20 cities for r
plt.title('Average Median Price by City')
plt.xlabel('Median Price')
plt.ylabel('City')

plt.tight_layout()
plt.show()
```



3. Average Quantity and Median Price by Year

```
# Calculate mean quantity and median price by year
year_agg = df.groupby('year')[['quantity', 'median_price']].mean().reset_index()

# Plotting

plt.figure(figsize=(14, 14))

# Quantity by Year
plt.subplot(2, 2, 3)
sns.lineplot(x='year', y='quantity', data=year_agg, marker='o', palette='viridis')
plt.title('Average Quantity by Year')
plt.xlabel('Year')
plt.ylabel('Quantity')

# Median Price by Year
plt.subplot(2, 2, 4)
sns.lineplot(x='year', y='median_price', data=year_agg, marker='o', palette='viridis')
plt.title('Average Median Price by Year')
plt.xlabel('Year')
plt.ylabel('Median Price')

plt.show()
```



.

.

.

.

.

4. Seasonal Monthly Trends

```
# Create a new column for month names
df['month_name'] = df.index.month_name()

# Ensure the month_name column has the correct categorical order
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'Oc
df['month_name'] = pd.Categorical(df['month_name'], categories=month_order, ordered=True)

# Group by month and calculate the median_price
monthly_trends = df.groupby('month_name')['median_price'].median().reindex(month_order)

# Create a bar plot with a trend line using Plotly
fig = px.bar(x=monthly_trends.index, y=monthly_trends.values)

# Adding a trend line
fig.add_trace(go.Scatter(
    x=monthly_trends.index, y=monthly_trends.values, mode='lines+markers', name='Trend Line', line=dict(c


fig.update_layout(title='Median price on Monthly basis',xaxis_title='Month',yaxis_title='Average Median P
fig.show()
```

.

.

.

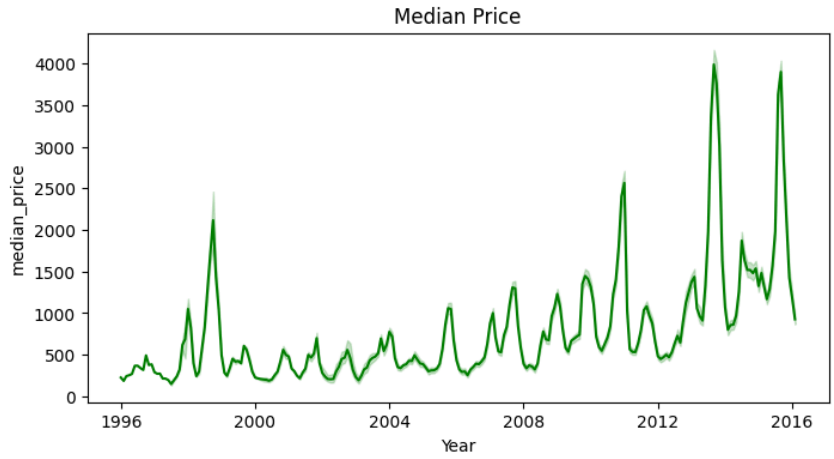
.

.

5. Median Price Lineplot by Year

```
plt.figure(figsize=(8, 4))
sns.lineplot(data=df, x=df.index, y='median_price',color='Green')
plt.title("Median Price")
plt.xlabel('Year')
```

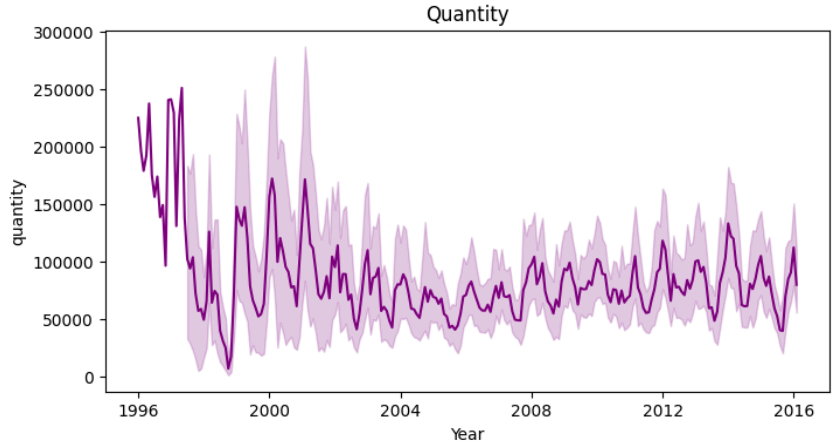
```
Text(0.5, 0, 'Year')
```



6. Quantity Lineplot by Year

```
plt.figure(figsize=(8, 4))
sns.lineplot(data=df, x=df.index, y='quantity',color='purple')
plt.title("Quantity")
plt.xlabel('Year')
```

```
Text(0.5, 0, 'Year')
```



7. Rolling Statistics

```
#Rolling Statistics
df['median_price:12_rolling']=df['median_price'].rolling(12).mean()
```

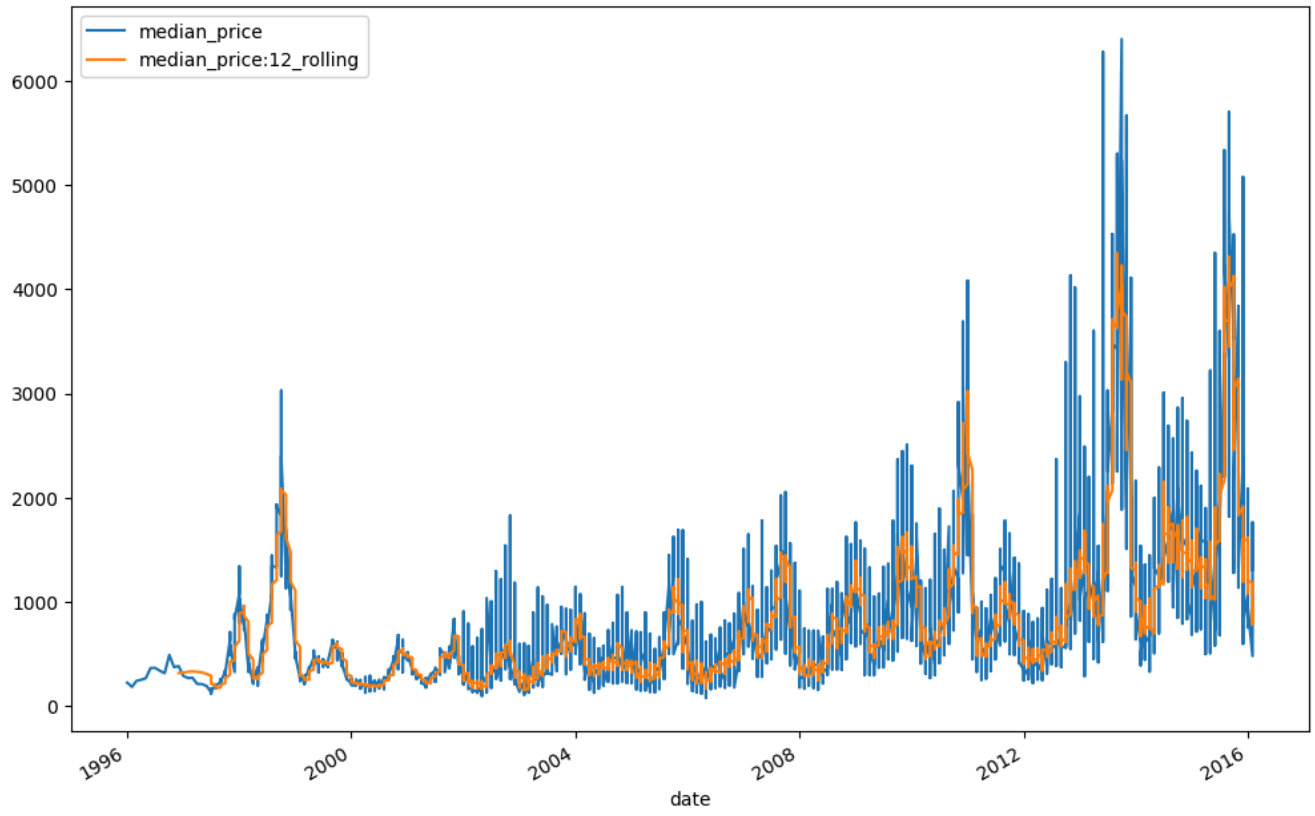
```
df.head()
```

date	market	month	year	quantity	pricemin	pricemax	pricemod	state	city	m
1996-01-01	LASALGAON(MS)	January	1996	225063	160	257	226	MS	LASALGAON	
1996-02-01	LASALGAON(MS)	February	1996	196164	133	229	186	MS	LASALGAON	
1996-03-01	LASALGAON(MS)	March	1996	178992	155	274	243	MS	LASALGAON	

Next steps: Generate code with df View recommended plots

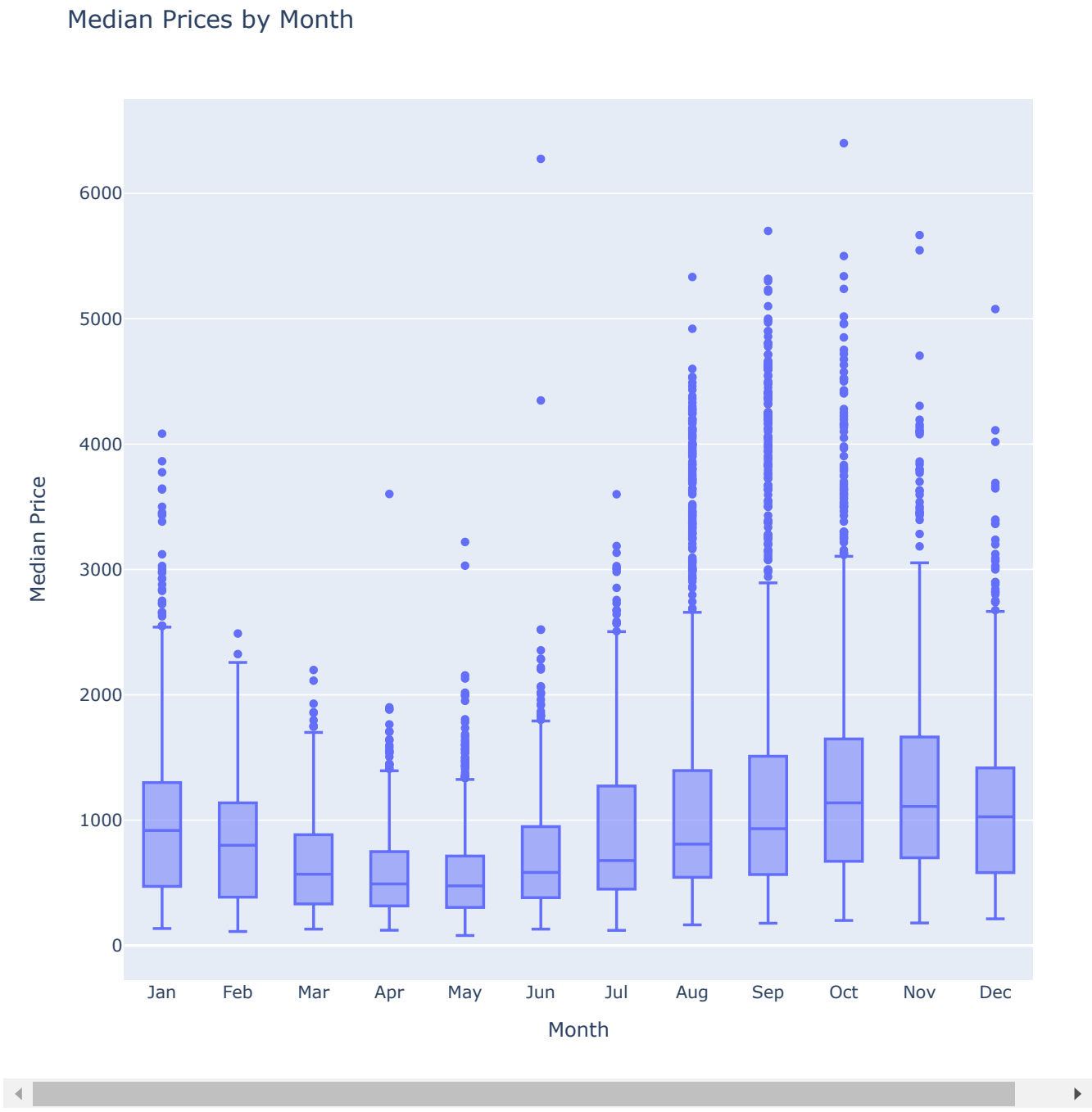
```
df[['median_price','median_price:12_rolling']].plot(figsize=(12,8))
```

```
<Axes: xlabel='date'>
```



8. Box Plot of Median Prices by Month

```
# Add a month column for plotting
df['month'] = df.index.month

# Box plot using Plotly
fig = px.box(df, x='month', y='median_price', title='Median Prices by Month', labels={'month': 'Month', '
fig.update_layout(xaxis=dict(tickmode='array', tickvals=list(range(1, 13)),
                             ticktext=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oc
fig.show()
```



.

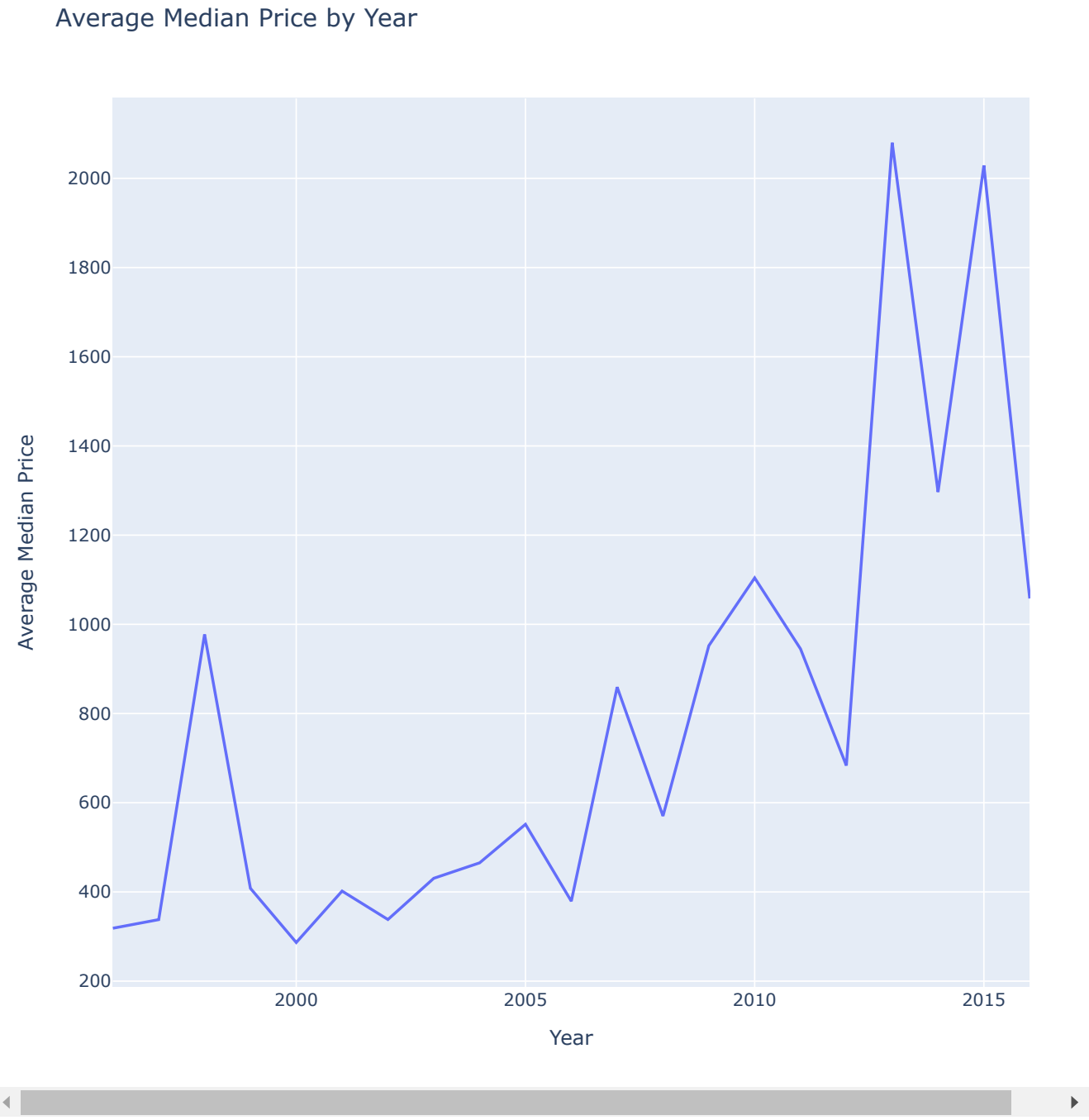
.

.

.

9. Plotting the Yearly Trend

```
# Extract year from the index
df['year'] = df.index.year

# Group by year and calculate the mean median_price
yearly_trends = df.groupby('year')['median_price'].mean()

# Plot using Plotly
fig = px.line(yearly_trends, x=yearly_trends.index, y='median_price', title='Average Median Price by Ye
fig.update_layout(xaxis_title='Year', yaxis_title='Average Median Price',height=800,width=800)
fig.show()
```

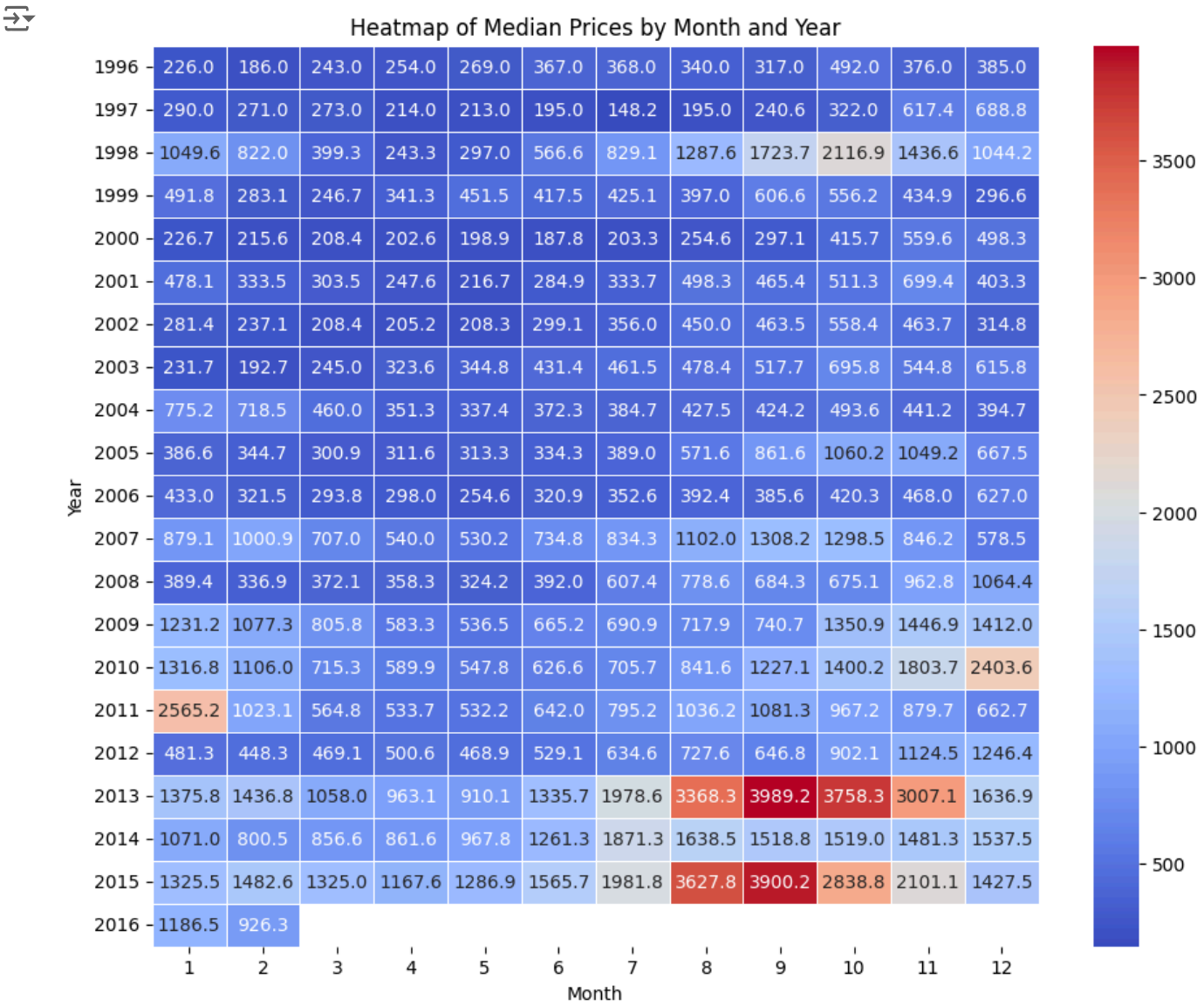


.

.

10. Heatmap of Median Prices by Month and Year

```
# Pivot table for heatmap
pivot = df.pivot_table(values='median_price', index=df.index.year, columns=df.index.month, aggfunc='mea

# Plot heatmap
plt.figure(figsize=(11, 9))
sns.heatmap(pivot, cmap='coolwarm', annot=True, fmt=".1f", linewidths=0.5)
plt.title('Heatmap of Median Prices by Month and Year')
plt.xlabel('Month')
plt.ylabel('Year')
plt.show()
```




```
df.columns
```

```
Index(['market', 'month', 'year', 'quantity', 'pricemin', 'pricemax',
       'pricemod', 'state', 'city', 'median_price', 'month_name',
       'median_price:12_rolling'],
      dtype='object')
```

```
# Dropping columns which are not neceesary for forecasting
df.drop(['market', 'month', 'year', 'pricemin', 'pricemax',
       'pricemod', 'state', 'city', 'median_price:12_rolling',
       'month_name'],axis=1,inplace=True)
```

```
df.head()
```

date	quantity	median_price
1996-01-01	225063	226.0
1996-02-01	196164	186.0
1996-03-01	178992	243.0
1996-04-01	192592	254.0
1996-05-01	237574	269.0

Next steps: Generate code with `df` View recommended plots

```
df.tail()
```

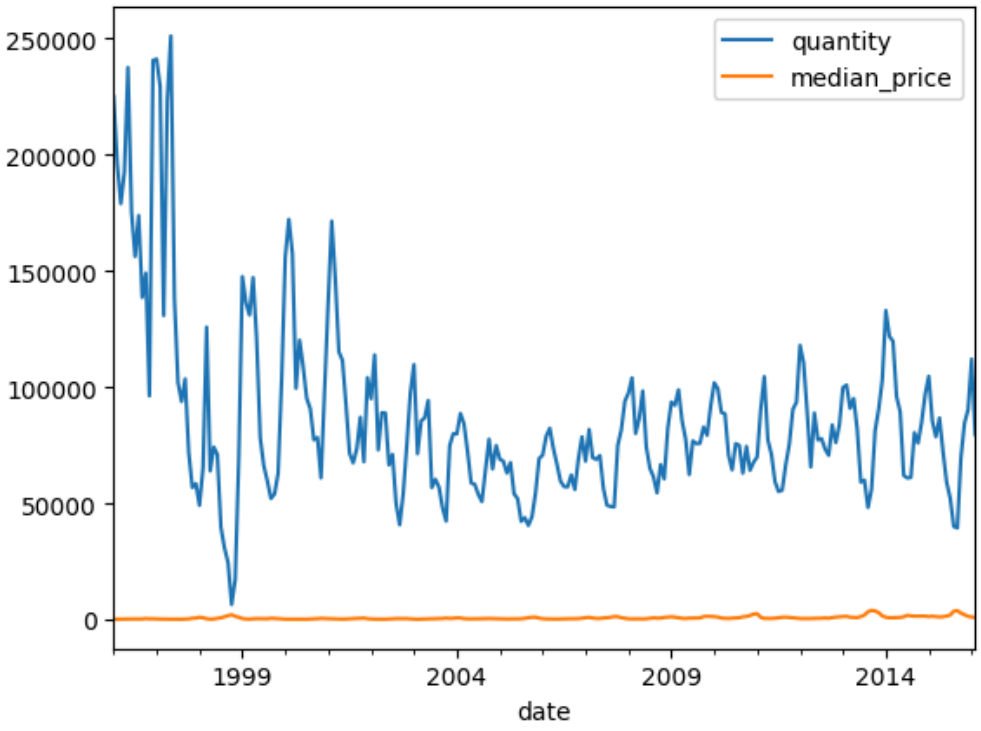
date	quantity	median_price
2016-02-01	4300	1077.0
2016-02-01	97178	575.0
2016-02-01	272527	730.0
2016-02-01	90142	806.0
2016-02-01	168020	1309.0

Removing duplicates (take mean for repeated dates) because we need to forecast on monthly basis

```
# Remove duplicate indices by aggregating data
df = df.groupby(df.index).mean()

# If the index is not in datetime format, convert it
if not pd.api.types.is_datetime64_any_dtype(df.index):
    df.index = pd.to_datetime(df.index)
```

```
df.plot()
```

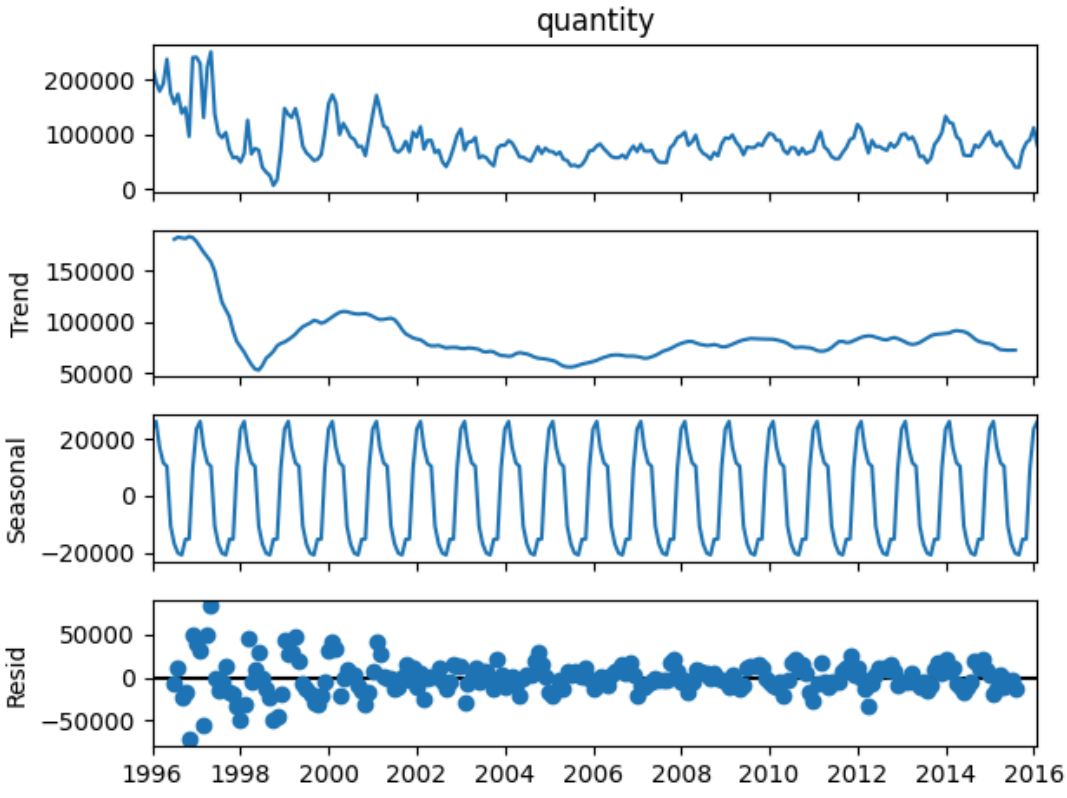
<Axes: xlabel='date'>



Start coding or generate with AI.

Seasonal decompose for Quantity

```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_data= seasonal_decompose(df['quantity'],model='additive')
decompose_data.plot();
```



.

.

.

```
# Testing For Stationarity
from statsmodels.tsa.stattools import adfuller


test_result=adfuller(df['quantity'])


# Ho: It is non stationary
# H1: It is stationary

def adfuller_test(quantity):
    result=adfuller(quantity)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non



adfuller_test(df['quantity'])
```

```
ADF Test Statistic : -6.082670116095265
p-value : 1.082777058260364e-07
#Lags Used : 15
Number of Observations Used : 226
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root
```

```
# Create rolling statistics
df['quantity_roll_mean'] = df['quantity'].rolling(window=3).mean()
df['quantity_roll_std'] = df['quantity'].rolling(window=3).std()


import statsmodels.api as sm
```

.

.

.

.

.

.

.

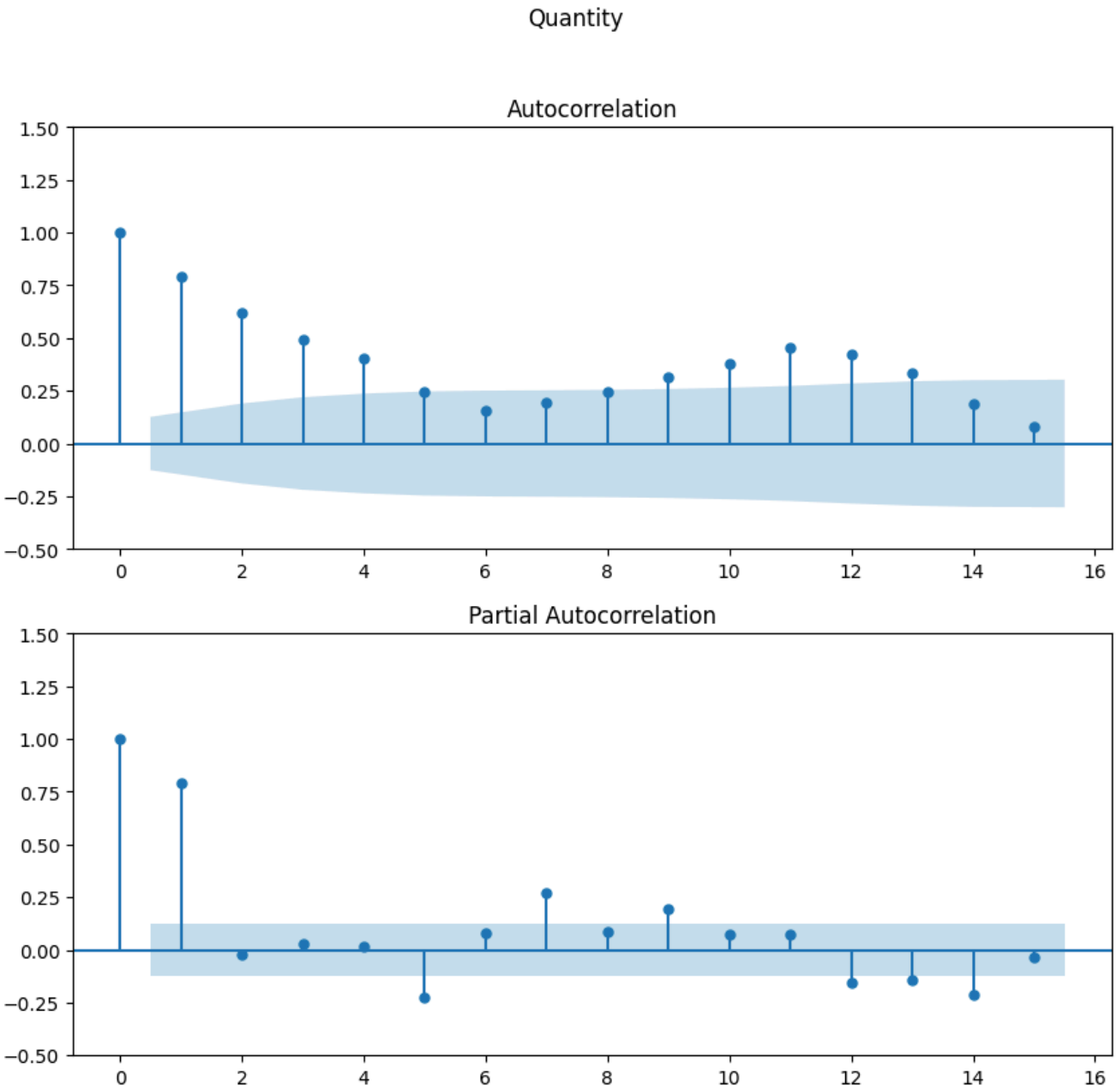
.

.

Plotting ACF and PACF graphs for Quantity

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf


fig = plt.figure(figsize=(10,9))
fig.suptitle('Quantity')
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['quantity'].dropna(),lags=15,ax=ax1)
ax1.set_ylim(-0.50, 1.5)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['quantity'].dropna(),lags=15,ax=ax2)
ax2.set_ylim(-0.50, 1.5)
```

```
(-0.5, 1.5)
```




```
# upper and lower bound increasing shows increase in errors
```

ARIMA model for Quantity

```
 # For non-seasonal data
#p=1, d=0, q=5
from statsmodels.tsa.arima.model import ARIMA

model=ARIMA(df['quantity'],order=(1,0,5))
model_fit=model.fit()


model_fit.summary()
```

SARIMAX Results

Dep. Variable:	quantity	**No. Observations:**	242
Model:	ARIMA(1, 0, 5)	**Log Likelihood**	-2777.167
Date:	Fri, 31 May 2024	**AIC**	5570.335
Time:	20:01:33	**BIC**	5598.246
Sample:	01-01-1996 - 02-01-2016	**HQIC**	5581.578

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
const	8.685e+04	5472.085	15.872	0.000	7.61e+04	9.76e+04
ar.L1	0.4089	0.058	7.099	0.000	0.296	0.522
ma.L1	0.4546	0.059	7.768	0.000	0.340	0.569
ma.L2	0.3644	0.051	7.085	0.000	0.264	0.465
ma.L3	0.2367	0.039	6.060	0.000	0.160	0.313
ma.L4	0.4684	0.039	12.149	0.000	0.393	0.544
ma.L5	0.2988	0.037	7.995	0.000	0.226	0.372
sigma2	2.983e+08	0.813	3.67e+08	0.000	2.98e+08	2.98e+08

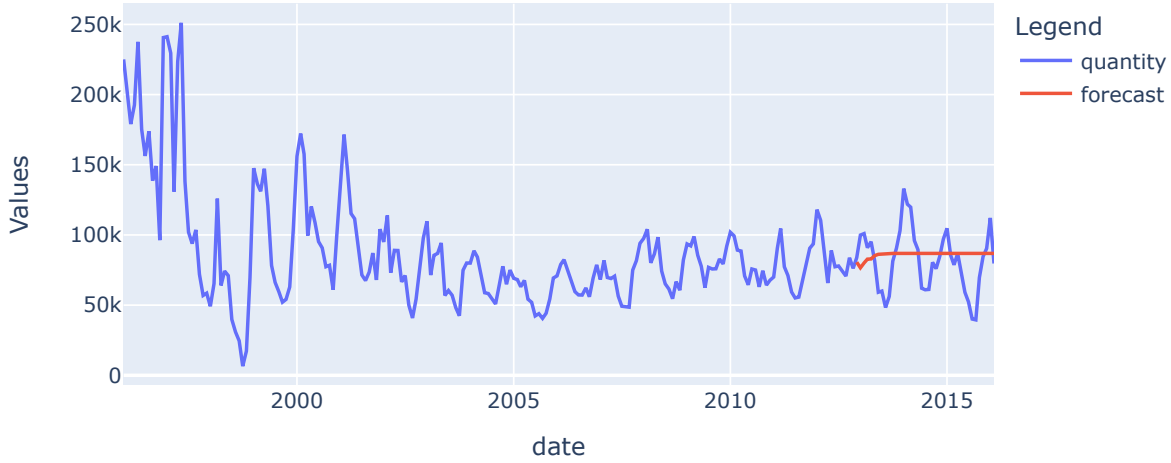
Ljung-Box (L1) (Q):	0.36	**Jarque-Bera (JB):**	627.30
Prob(Q):	0.55	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.17	**Skew:**	1.34
Prob(H) (two-sided):	0.00	**Kurtosis:**	10.42

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.7e+23. Standard errors may be unstable.

```
df.drop(['quantity_roll_mean','quantity_roll_std'],axis=1,inplace=True)


df.tail()
```

date	quantity	median_price
2015-10-01	69684.900000	2838.788889
2015-11-01	84584.707865	2101.056180
2015-12-01	90368.806452	1427.516129
2016-01-01	112143.650602	1186.457831
2016-02-01	79671.049383	926.345679

```
import datetime


# Generate forecast
start_date = datetime.datetime(2012, 12, 1)
end_date = datetime.datetime(2016, 2, 1)
forecasted_date= datetime.datetime(2017, 2, 1)


df['forecast']=model_fit.predict(start=start_date,end=end_date,dynamic=True)

fig = px.line(df, y=['quantity', 'forecast'], labels={'value': 'Values', 'variable': 'Legend'}, title='
fig.update_layout(width=700, height=400)
fig.show()
```



Calculating MAE, MSE, RMSE

```
# Calculate error metrics
actual = df['quantity'][start_date:end_date]
forecast = df['forecast'][start_date:end_date]

ARIMA_Qty_mae = mean_absolute_error(actual, forecast)
ARIMA_Qty_mse = mean_squared_error(actual, forecast)
ARIMA_Qty_rmse = np.sqrt(ARIMA_Qty_mse)

print(f'ARIMA_Qty_MAE: {ARIMA_Qty_mae}')
print(f'ARIMA_Qty_MSE: {ARIMA_Qty_mse}')
print(f'ARIMA_Qty_RMSE: {ARIMA_Qty_rmse}')
```

```
ARIMA_Qty_MAE: 18077.99213946133
ARIMA_Qty_MSE: 515925661.36036867
ARIMA_Qty_RMSE: 22713.997036197055
```

```
df
```

date	quantity	median_price	forecast
1996-01-01	225063.000000	226.000000	NaN
1996-02-01	196164.000000	186.000000	NaN
1996-03-01	178992.000000	243.000000	NaN
1996-04-01	192592.000000	254.000000	NaN
1996-05-01	237574.000000	269.000000	NaN
...	...	...	...
2015-10-01	69684.900000	2838.788889	86854.922141
2015-11-01	84584.707865	2101.056180	86854.922141
2015-12-01	90368.806452	1427.516129	86854.922141
2016-01-01	112143.650602	1186.457831	86854.922141
2016-02-01	79671.049383	926.345679	86854.922141

242 rows × 3 columns

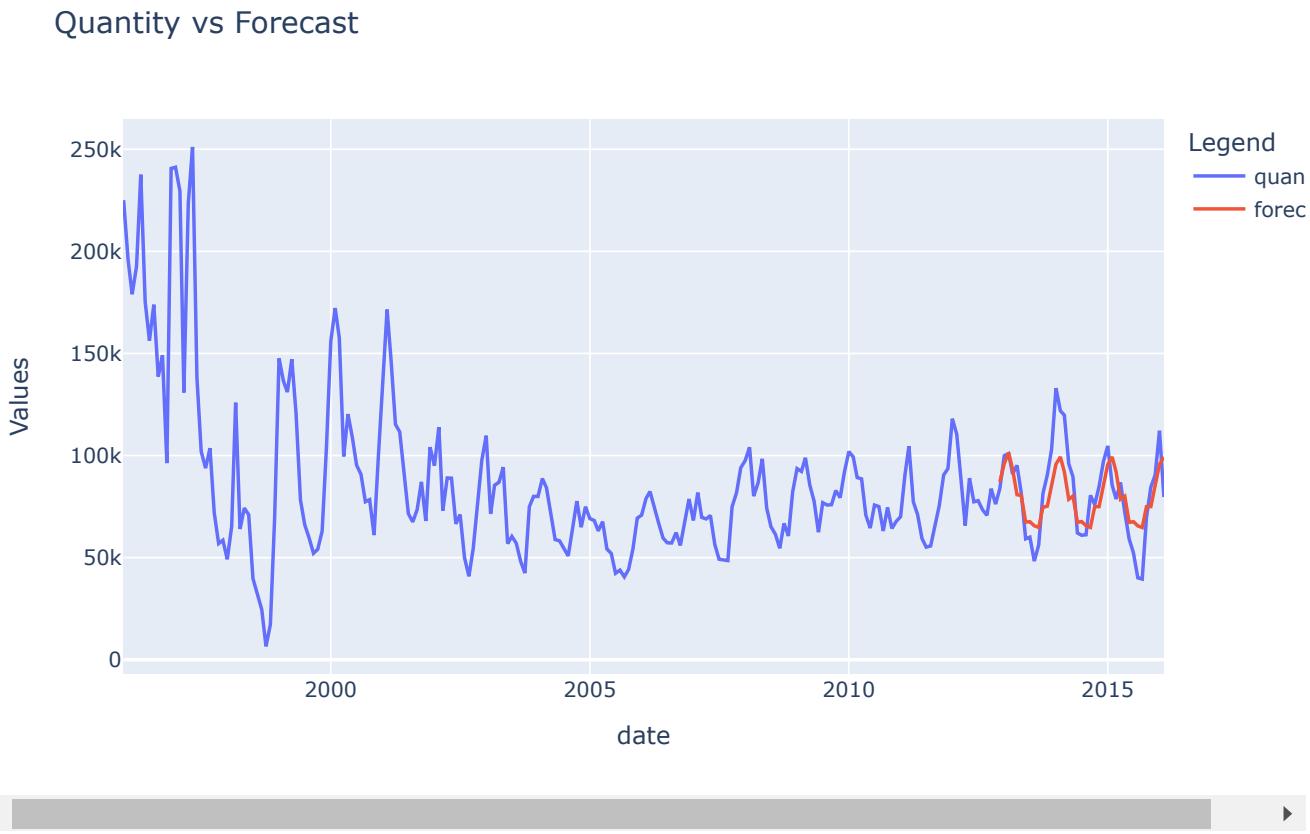
Next steps: Generate code with `df` View recommended plots

SARIMAX model for Quantity

```
model=sm.tsa.statespace.SARIMAX(df['quantity'],order=(1, 0, 5),seasonal_order=(1,1,1,12))
results=model.fit()
```

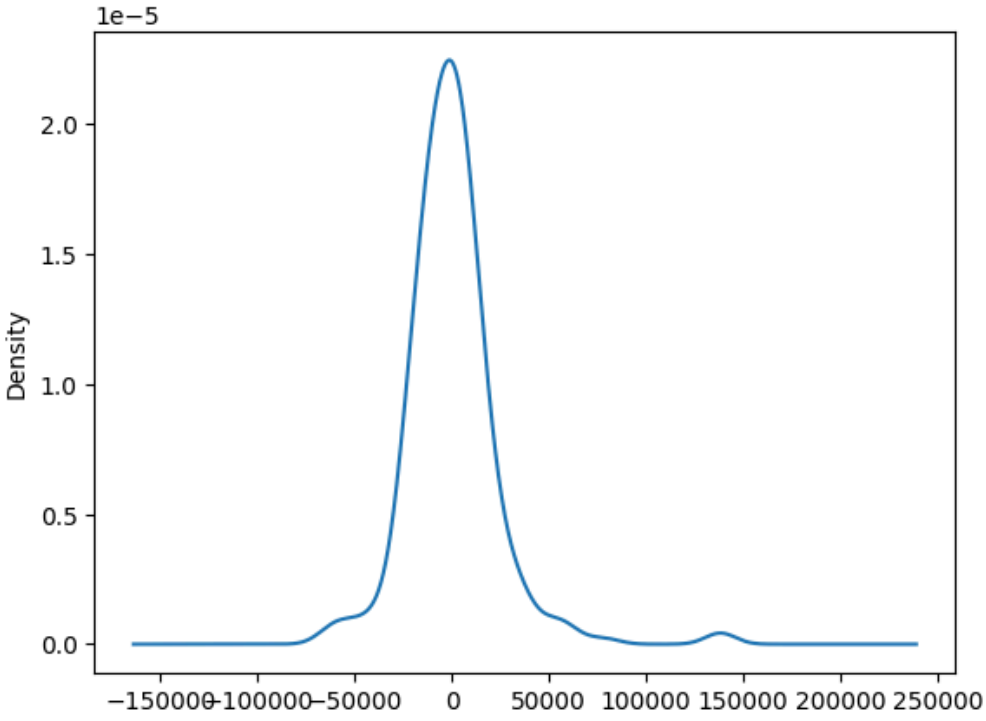
```
import plotly.express as px
```

```
df['forecast']=results.predict(start=start_date,end=end_date,dynamic=True)

fig = px.line(df, y=['quantity', 'forecast'], labels={'value': 'Values', 'variable': 'Legend'}, title='
fig.update_layout(width=800, height=500)
fig.show()
```




```
model_fit.resid.plot(kind='kde')
```

```
<Axes: ylabel='Density'>
```



.

.

.

Calculating MAE, MSE, RMSE

```
# Calculate error metrics
actual = df['quantity'][start_date:end_date]
forecast = df['forecast'][start_date:end_date]

SARIMAX_Qty_mae = mean_absolute_error(actual, forecast)
SARIMAX_Qty_mse = mean_squared_error(actual, forecast)
SARIMAX_Qty_rmse = np.sqrt(SARIMAX_Qty_mse)

print(f'SARIMAX_Qty_MAE: {SARIMAX_Qty_mae}')
print(f'SARIMAX_Qty_MSE: {SARIMAX_Qty_mse}')
print(f'SARIMAX_Qty_RMSE: {SARIMAX_Qty_rmse}')
```

```
SARIMAX_Qty_MAE: 11753.369779947783
SARIMAX_Qty_MSE: 203379011.72280535
SARIMAX_Qty_RMSE: 14261.101350274646
```

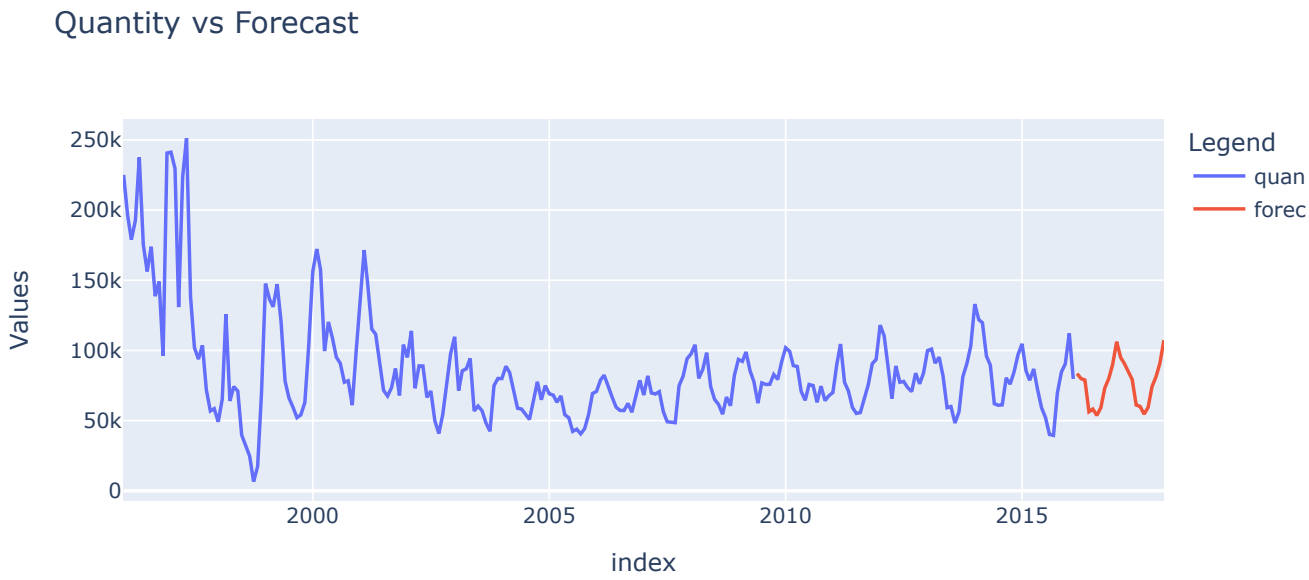
```
from pandas.tseries.offsets import DateOffset
future_dates=[df.index[-1]+ DateOffset(months=x)for x in range(0,24)]


future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df.columns)


print(future_datest_df.index[0])
future_datest_df.index[-1]
```

```
2016-03-01 00:00:00
Timestamp('2018-01-01 00:00:00')
```

```
future_datest_df.tail()
```

	quantity	median_price	forecast
2017-09-01	NaN	NaN	NaN
2017-10-01	NaN	NaN	NaN
2017-11-01	NaN	NaN	NaN
2017-12-01	NaN	NaN	NaN
2018-01-01	NaN	NaN	NaN

```
future_df=pd.concat([df,future_datest_df])


future_df['forecast'] = results.predict(start = datetime.datetime(2016,3,1), end = datetime.datetime(20
fig1 = px.line(future_df, y=['quantity', 'forecast'], labels={'value': 'Values', 'variable': 'Legend'},
               title='Quantity vs Forecast')
fig1.update_layout(width=800, height=400)
fig1.show()
```



PROPHET model for Quantity

```
#pip install prophet

#conda update -n base -c defaults conda

from prophet import Prophet

#Initializing model
model=Prophet()
model
```

```
<prophet.forecaster.Prophet at 0x7f69e5d0cdf0>
```

```
df.drop('forecast',axis=1,inplace=True)

df= df.reset_index()

df.head()
```

	date	quantity	median_price
0	1996-01-01	225063.0	226.0
1	1996-02-01	196164.0	186.0
2	1996-03-01	178992.0	243.0
3	1996-04-01	192592.0	254.0
4	1996-05-01	237574.0	269.0

Next steps: Generate code with df | View recommended plots

```
df=pd.DataFrame(df[['date','quantity']])
```

```
df.head()
```

	date	quantity
0	1996-01-01	225063.0
1	1996-02-01	196164.0
2	1996-03-01	178992.0
3	1996-04-01	192592.0
4	1996-05-01	237574.0

Next steps: Generate code with df | View recommended plots

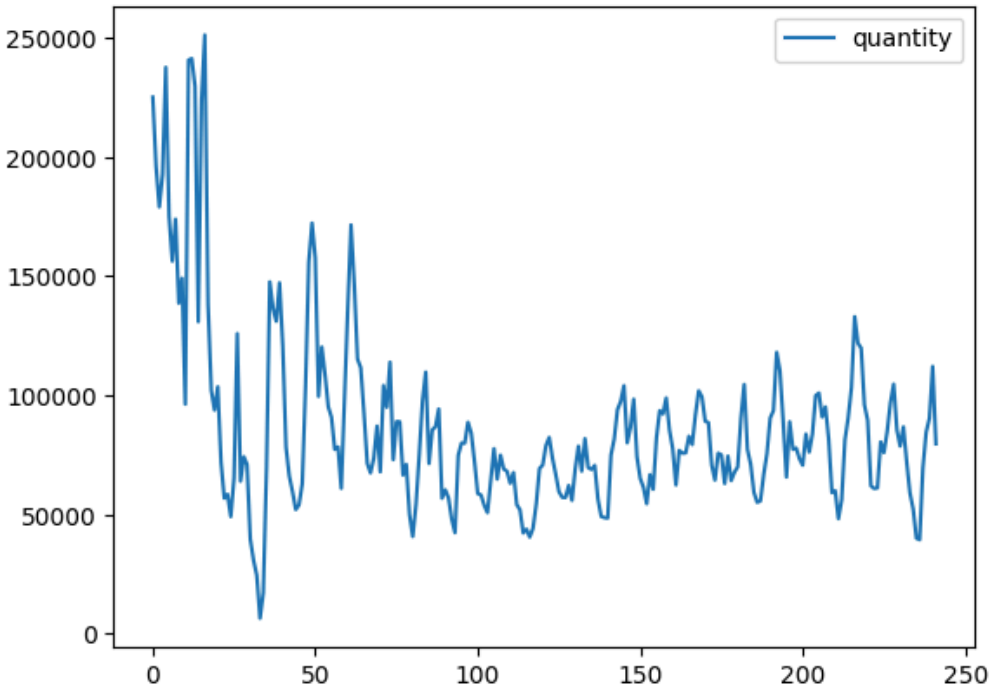
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242 entries, 0 to 241
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      242 non-null    datetime64[ns]
 1   quantity  242 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 3.9 KB
```

```
df['date']=df['date'].astype(str)
```

```
df[['date','quantity']].plot()
```

```
<Axes: >
```




```
df.columns = ['ds','y']
df.head()
```

	ds	y
0	1996-01-01	225063.0
1	1996-02-01	196164.0
2	1996-03-01	178992.0
3	1996-04-01	192592.0
4	1996-05-01	237574.0

Next steps: Generate code with `df` View recommended plots

```
df.tail()
```

	ds	y
237	2015-10-01	69684.900000
238	2015-11-01	84584.707865
239	2015-12-01	90368.806452
240	2016-01-01	112143.650602
241	2016-02-01	79671.049383

```
df['ds'] = pd.to_datetime(df['ds'])
df.head()
```

	ds	y
0	1996-01-01	225063.0
1	1996-02-01	196164.0
2	1996-03-01	178992.0
3	1996-04-01	192592.0
4	1996-05-01	237574.0

Next steps: Generate code with `df` View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242 entries, 0 to 241
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   ds      242 non-null    datetime64[ns]
 1   y       242 non-null    float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 3.9 KB
```

```
df.columns
```

```
Index(['ds', 'y'], dtype='object')
```

```
model.fit(df)
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override thi
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5r655x8k/91fk2pgy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5r655x8k/oj23dols.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_
19:30:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
19:30:20 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7f69e5d0cdf0>
```

```
model.component_modes
```

```
{'additive': ['yearly',
  'additive_terms',
  'extra_regressors_additive',
  'holidays'],
 'multiplicative': ['multiplicative_terms', 'extra_regressors_multiplicative']}
```

```
df.tail()
```

	ds	y
237	2015-10-01	69684.900000
238	2015-11-01	84584.707865
239	2015-12-01	90368.806452
240	2016-01-01	112143.650602
241	2016-02-01	79671.049383

```
### Create future dates of 365 days
future_dates=model.make_future_dataframe(periods=365)
```

```
future_dates.tail()
```

	ds
602	2017-01-27
603	2017-01-28
604	2017-01-29
605	2017-01-30
606	2017-01-31

```
prediction=model.predict(future_dates)
```

```
prediction.head()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	a
0	1996-01-01	139990.519816	129664.702955	199358.562591	139990.519816	139990.519816	26007.121425	
1	1996-02-01	139199.524330	129629.894690	200724.799045	139199.524330	139199.524330	24290.096581	
2	1996-03-01	138459.560811	120076.074139	189699.667339	138459.560811	138459.560811	16653.142091	
3	1996-04-01	137668.565325	113117.126572	181630.047180	137668.565325	137668.565325	10300.868574	
4	1996-05-01	136903.085823	114678.090656	185524.189529	136903.085823	136903.085823	13531.555439	
5	1996-06-01	136112.090337	91938.848931	160034.609434	136112.090337	136112.090337	-10729.731592	
6	1996-07-01	135346.610835	81574.993132	151492.275462	135346.610835	135346.610835	-18065.266310	
7	1996-08-01	134555.615349	77830.884665	147935.026880	134555.615349	134555.615349	-20523.768090	
8	1996-09-01	133764.619863	77748.109623	146177.167143	133764.619863	133764.619863	-22056.470191	
9	1996-10-01	132999.140362	82782.665031	153101.016896	132999.140362	132999.140362	-14447.109569	

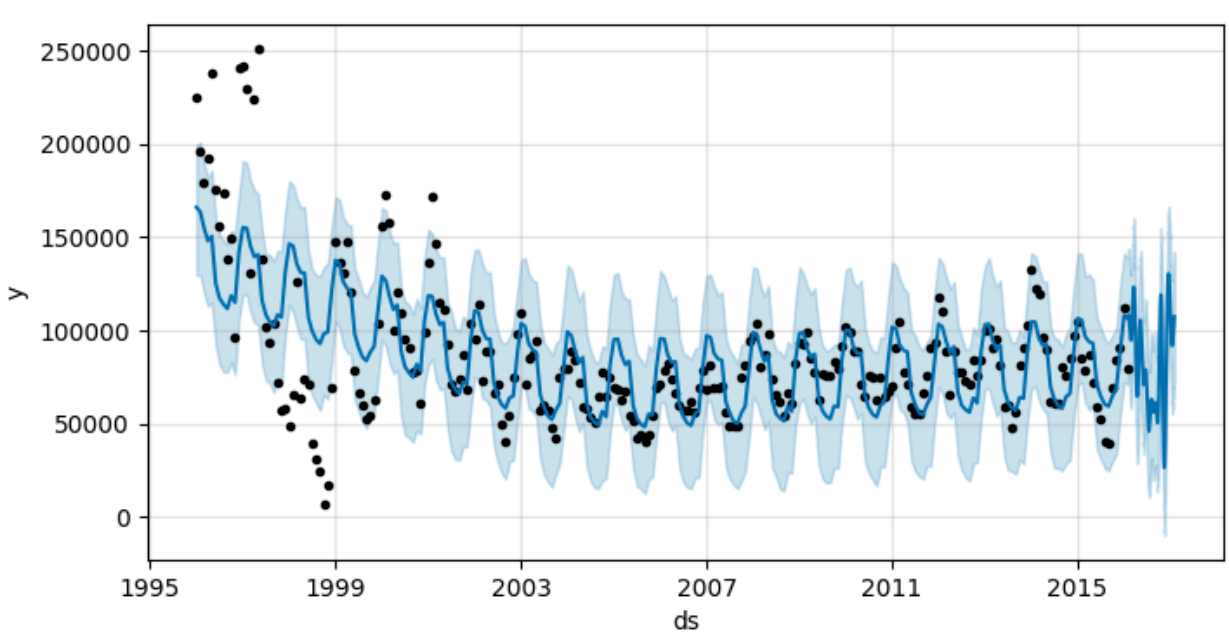
Next steps: Generate code with `prediction` View recommended plots

Calculating MAE, MSE, RMSE

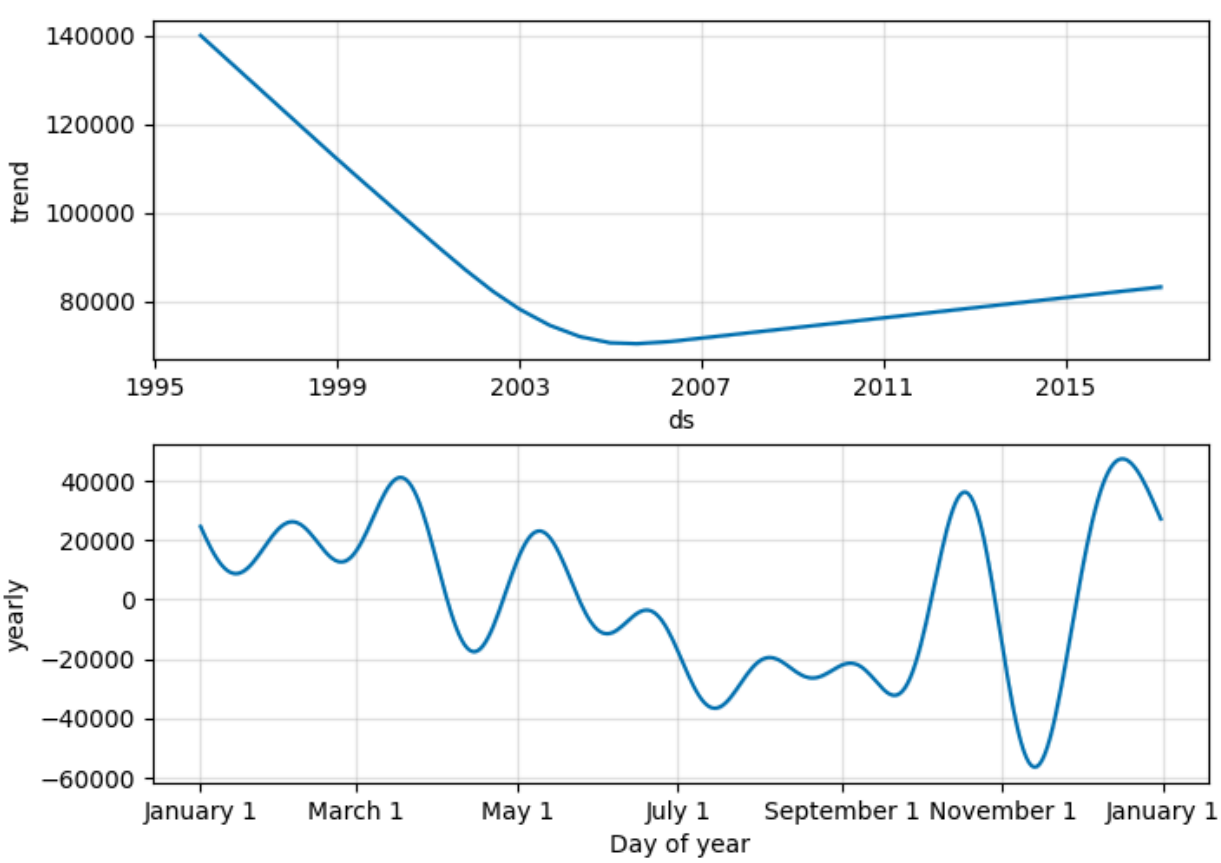
```
# Add forecast to the original dataframe
df['prediction'] = prediction['yhat'].iloc[:len(df)]

# Define start and end dates for error metrics calculation
start_date = '2012-12-01'
end_date = '2016-02-01'

# Filter actual and forecasted values
actual = df.loc[(df['ds'] >= start_date) & (df['ds'] <= end_date), 'y']
forecasted = df.loc[(df['ds'] >= start_date) & (df['ds'] <= end_date), 'prediction']

# Calculate error metrics
Prophet_Qty_mae = mean_absolute_error(actual, forecasted)
Prophet_Qty_mse = mean_squared_error(actual, forecasted)
Prophet_Qty_rmse = np.sqrt(Prophet_Qty_mse)

print(f'Prophet_Qty_MAE: {Prophet_Qty_mae}')
print(f'Prophet_Qty_MSE: {Prophet_Qty_mse}')
print(f'Prophet_Qty_RMSE: {Prophet_Qty_rmse}')
```

```
Prophet_Qty_MAE: 11432.407342700437
Prophet_Qty_MSE: 199122076.2316043
Prophet_Qty_RMSE: 14111.062193598478
```

```
### plot the predicted projection
fig, ax = plt.subplots(figsize=(8, 4))
model.plot(prediction,ax=ax);
```




```
#### Visualize Each Components[Trends,yearly]
components_fig = model.plot_components(prediction)
components_fig.set_size_inches(7, 5)
```



.

.

.

Cross validation through prophet model

```
from prophet.diagnostics import cross_validation

df_cv = cross_validation(model, initial='730 days', period='180 days', horizon = '365 days')
```

Show hidden output

```
df_cv.head(20)
```

	ds	yhat	yhat_lower	yhat_upper	
0	1998-06-01	19206.131541	-18200.283651	54661.919974	71028.8
1	1998-07-01	-29803.764530	-67925.352084	6167.108574	39771.8
2	1998-08-01	-41124.210618	-77136.727875	-6949.796323	30891.2
3	1998-09-01	38672.282910	4047.693703	72374.958862	24575.0
4	1998-10-01	-788.197140	-34931.814110	36609.922314	6526.8
5	1998-11-01	48928.786079	14266.717442	81734.406178	17272.2
6	1998-12-01	-69321.307629	-103978.376663	-31940.073529	69516.5
7	1999-01-01	-14045.908989	-46926.757087	21991.112937	147595.4
8	1999-02-01	-28833.948251	-67060.559211	6596.354162	136469.4
9	1999-03-01	95749.816681	60573.358889	131668.168190	131090.0
10	1999-04-01	19272.185450	-14725.779660	53772.847198	147196.2
11	1999-05-01	1671.615871	-31669.428301	37309.219979	120705.3
12	1998-11-01	62195.850275	25803.178492	97250.521239	17272.2

Next steps: Generate code with `df_cv` View recommended plots

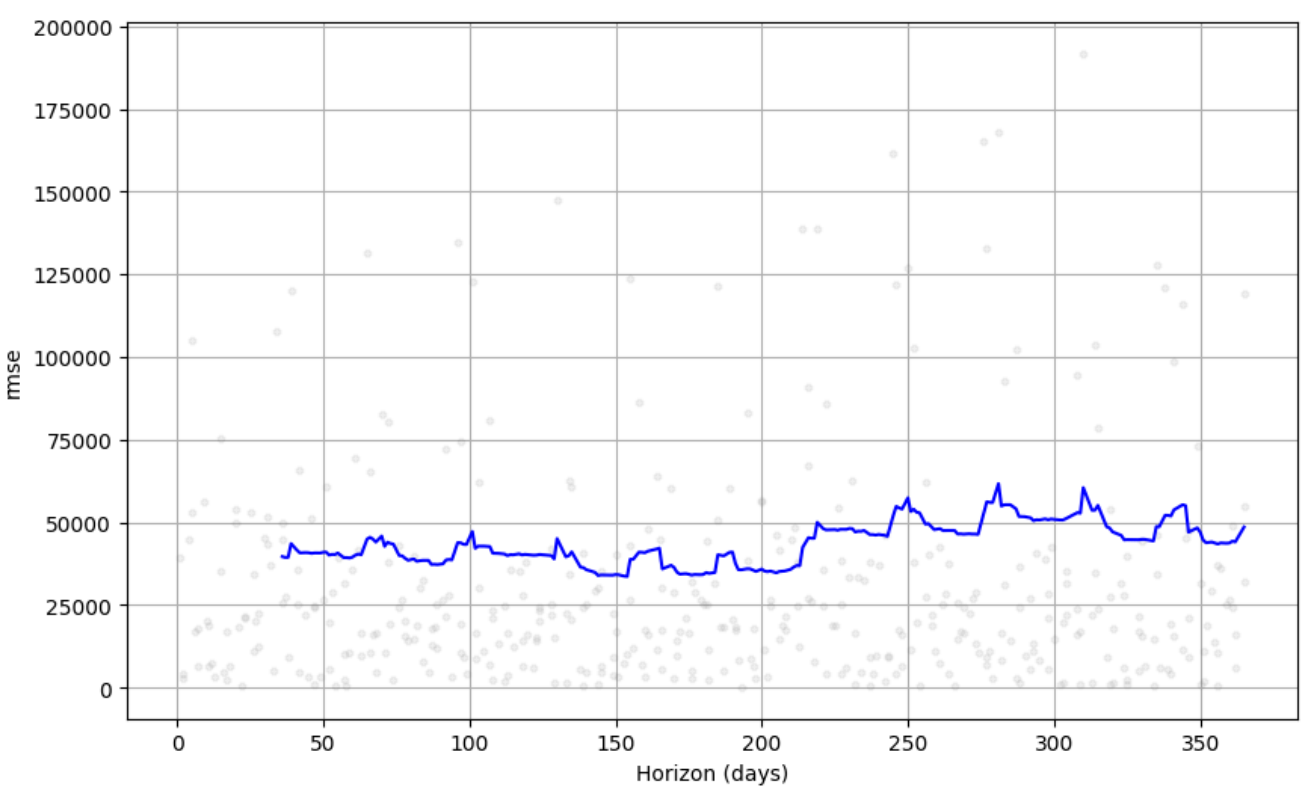
```
from prophet.diagnostics import performance_metrics
df_p = performance_metrics(df_cv)
```

```
df_p.head(13)
```

	horizon	mse	rmse	mae	mape
0	36 days	1.572612e+09	39656.178481	30333.392501	0.458452
1	37 days	1.553550e+09	39415.095854	30048.370229	0.458139
2	38 days	1.555383e+09	39438.337963	30193.118703	0.460066
3	39 days	1.899193e+09	43579.735304	32976.080962	0.486695
4	41 days	1.731796e+09	41614.855995	31476.363129	0.429898
5	42 days	1.663217e+09	40782.558700	30864.951512	0.421024
6	44 days	1.670301e+09	40869.315388	31096.457096	0.425791
7	45 days	1.666224e+09	40819.407579	30887.695777	0.423444
8	46 days	1.652551e+09	40651.578578	30760.527684	0.419685
9	47 days	1.661875e+09	40766.106804	30864.598285	0.414350
10	49 days	1.660845e+09	40753.461053	30767.035798	0.412789
11	50 days	1.677476e+09	40956.997417	31318.141570	0.417946
12	51 days	1.683053e+09	41025.026392	31449.436302	0.413473

Next steps: Generate code with df_p | View recommended plots

```
from prophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='rmse')
fig.set_size_inches(10, 6)
```



LSTM Model on Quantity

```
df=pd.read_csv('/content/drive/MyDrive/MarketPricePrediction.csv')
```

```
#Convert date column to datetime
df['date'] = pd.to_datetime(df['date'])
df.tail()
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city	date
10222	YEOLA(MS)	December	2011	131326	282	612	526	MS	YEOLA	2011-12-01
10223	YEOLA(MS)	December	2012	207066	485	1327	1136	MS	YEOLA	2012-12-01
10224	YEOLA(MS)	December	2013	215883	472	1427	1177	MS	YEOLA	2013-12-01

```
df.set_index('date', inplace=True)
df.head()
```

date	market	month	year	quantity	priceMin	priceMax	priceMod	state	city
2005-01-01	ABOHAR(PB)	January	2005	2350	404	493	446	PB	ABOHAR
2006-01-01	ABOHAR(PB)	January	2006	900	487	638	563	PB	ABOHAR
2010-01-01	ABOHAR(PB)	January	2010	790	1283	1592	1460	PB	ABOHAR
2011-01-									

Next steps: Generate code with `df` View recommended plots

```
df1=df.reset_index()['quantity']
```

```
df1
```

```
0          2350
1           900
2           790
3           245
4          1035
          ...
10222    131326
10223    207066
10224    215883
10225    201077
10226    223315
Name: quantity, Length: 10227, dtype: int64
```

```
# Remove duplicate indices by aggregating data (Taking mean for repeated dates)
df1 = df1.groupby(df.index).mean()

# If the index is not in datetime format, convert it
if not pd.api.types.is_datetime64_any_dtype(df.index):
    df1.index = pd.to_datetime(df1.index)


df1
```

```
date
1996-01-01    225063.000000
1996-02-01    196164.000000
1996-03-01    178992.000000
1996-04-01    192592.000000
1996-05-01    237574.000000
                  ...
2015-10-01     69684.900000
2015-11-01     84584.707865
2015-12-01     90368.806452
2016-01-01    112143.650602
2016-02-01     79671.049383
Name: quantity, Length: 242, dtype: float64
```

```
df1.tail()
```

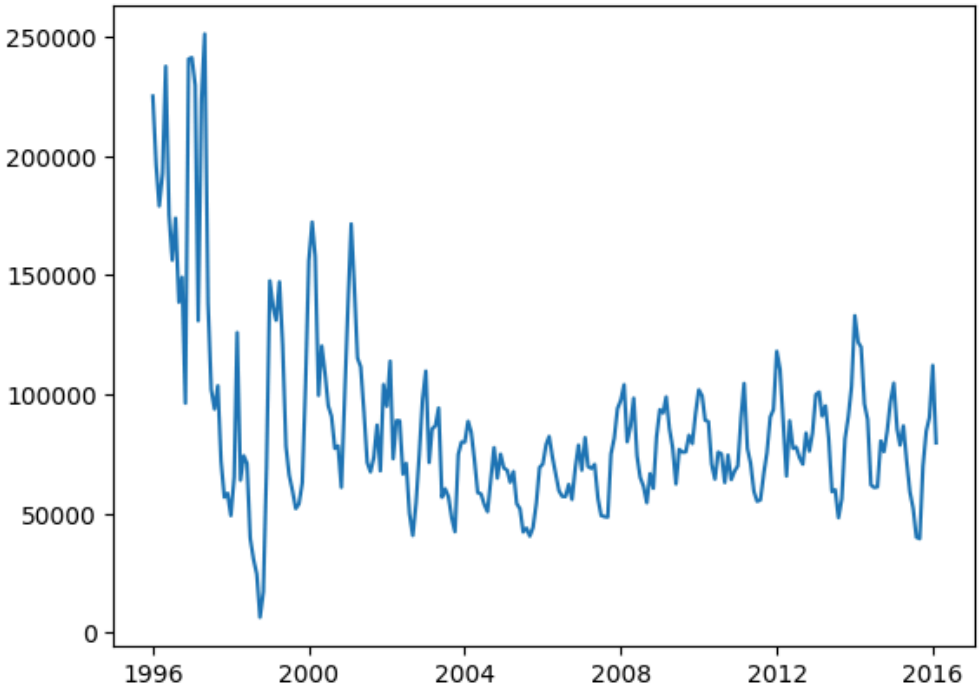
```
date
2015-10-01     69684.900000
2015-11-01     84584.707865
2015-12-01     90368.806452
2016-01-01    112143.650602
2016-02-01     79671.049383
Name: quantity, dtype: float64
```

```
df1.shape
```

```
(242,)
```

```
import matplotlib.pyplot as plt
plt.plot(df1)
```

```
[<matplotlib.lines.Line2D at 0x7f697a83abc0>]
```




```
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler


import numpy as np


from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))


print(df1)
```

Show hidden output

```
##splitting dataset into train and test split
training_size=int(len(df1)*0.80)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]


training_size,test_size
```

```
(193, 49)
```

```
train_data
```

Show hidden output

```
import numpy
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return numpy.array(dataX), numpy.array(dataY)


# reshape into X=t,t+1,t+2,t+3,t+4 and Y=t+5
time_step = 5
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)


print(X_train.shape), print(y_train.shape)
```

```
(187, 5)
(187,)
(None, None)
```

```
print(X_test.shape), print(ytest.shape)
```

```
(43, 5)
(43,)
(None, None)
```

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)


### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
from keras.callbacks import EarlyStopping
from tensorflow.keras.layers import LSTM
from tensorflow.keras.optimizers import Adam


# Define the learning rate
learning_rate = 0.001  # You can adjust this value as needed

# Create the optimizer with the custom learning rate
optimizer = Adam(learning_rate=learning_rate)


model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(5, 1)))
model.add(Dropout(0.2))  # Add dropout layer with 20% dropout rate
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=optimizer)


model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 lstm_3 (LSTM)               (None, 5, 50)             10400

 dropout_2 (Dropout)         (None, 5, 50)             0

 lstm_4 (LSTM)               (None, 5, 50)             20200

 dropout_3 (Dropout)         (None, 5, 50)             0

 lstm_5 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=500,batch_size=64,verbose=1)
```

Show hidden output

```
## Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
6/6 [==============================] - 1s 5ms/step
2/2 [==============================] - 0s 7ms/step
```

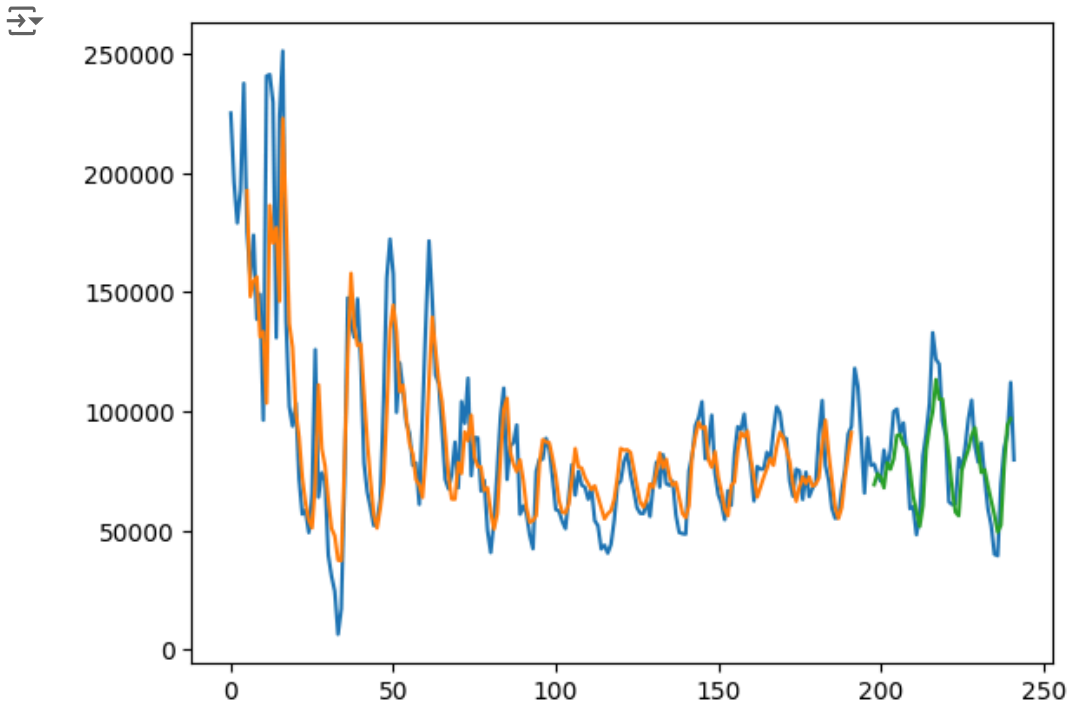
```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

Calculate MAE,MSE,RMSE

```
# Calculate error metrics for the test set
LSTM_Qty_mae = mean_absolute_error(ytest, test_predict)
LSTM_Qty_mse = mean_squared_error(ytest, test_predict)
LSTM_Qty_rmse = np.sqrt(LSTM_Qty_mse)

print(f'LSTM_Qty_MAE: {LSTM_Qty_mae}')
print(f'LSTM_Qty_MSE: {LSTM_Qty_mse}')
print(f'LSTM_Qty_RMSE: {LSTM_Qty_rmse}')
```

```
LSTM_Qty_MAE: 77854.61477517188
LSTM_Qty_MSE: 6299328010.93953
LSTM_Qty_RMSE: 79368.30608586484
```

```
### Plotting
# shift train predictions for plotting
look_back=5
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```




```
len(test_data)
```

```
49
```

```
x_input=test_data[44:].reshape(1,-1)
x_input.shape
```

```
(1, 5)
```

```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

```
temp_input
```

```
[0.25820424095644734,
 0.31911794134044796,
 0.34276461191999563,
 0.43178497678058214,
 0.2990298218628061]
```

```
# demonstrate prediction for next 30 Months
from numpy import array
```

```
lst_output=[]
n_steps=5
i=0
while(i<30):

    if(len(temp_input)>5):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

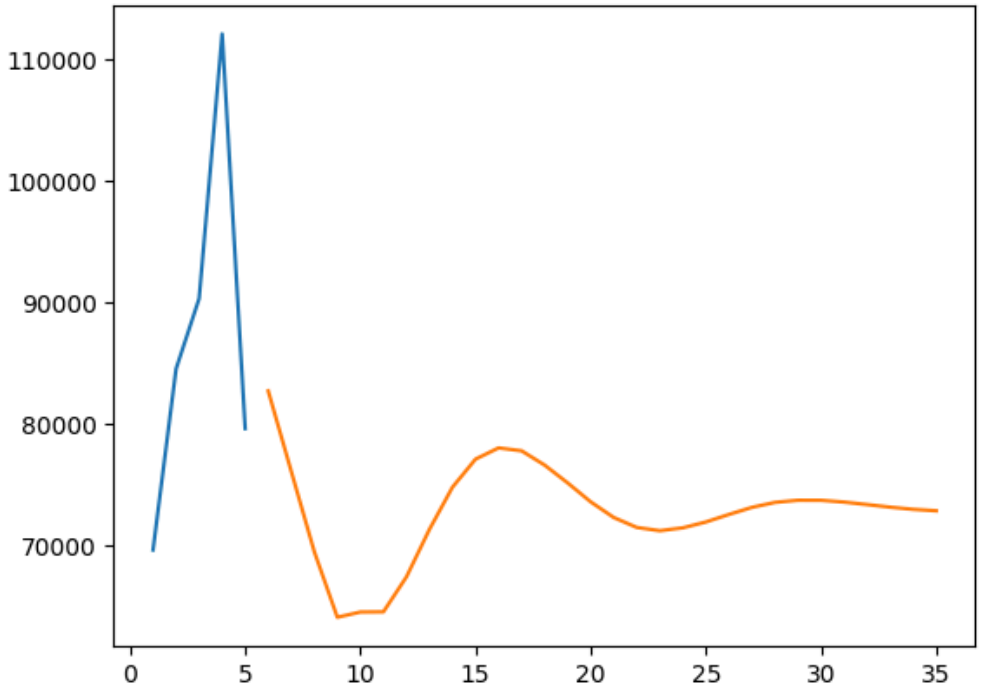
Show hidden output

```
day_new=np.arange(1,6)
day_pred=np.arange(6,36)


len(df1)
```

242

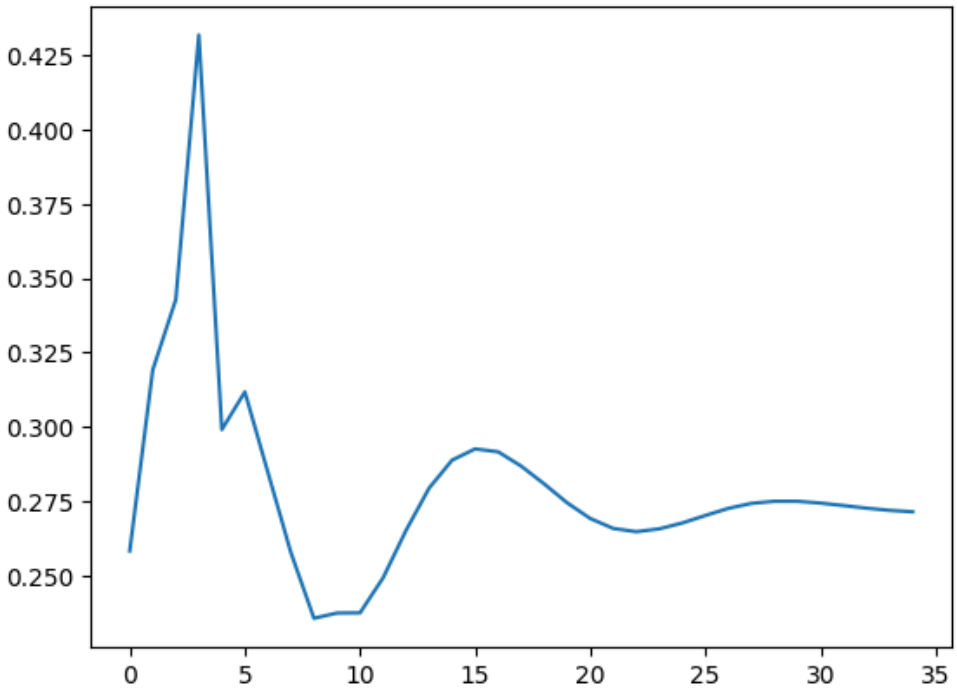
```
#Forecasting for next 30 months
plt.plot(day_new,scaler.inverse_transform(df1[237:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x7f697cbc8b50>]
```



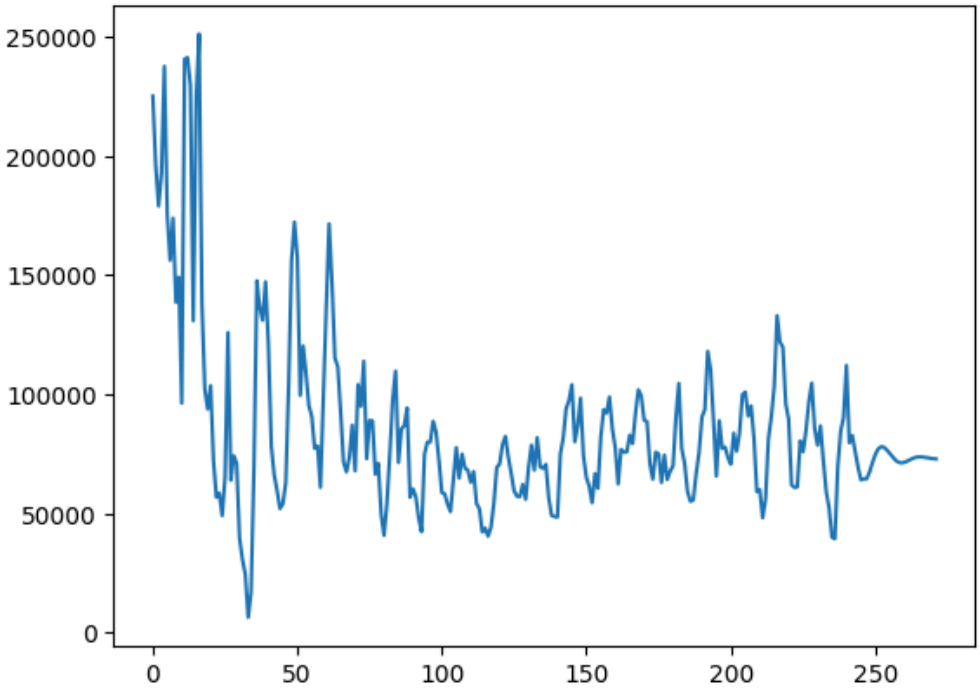

```
#extending the line from the previous months
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[237:])
```

```
[<matplotlib.lines.Line2D at 0x7f69754809a0>]
```




```
# Complete plot including forecasting of 30 months
df3=scaler.inverse_transform(df3).tolist()


plt.plot(df3)
```

```
[<matplotlib.lines.Line2D at 0x7f697cc1fd60>]
```




```
# prompt: convert df3 to dataframe

import pandas as pd
df3 = pd.DataFrame(df3, columns=['quantity'])
```

```
df3.tail(14)
```

	quantity
258	71533.100927
259	71266.775263
260	71502.483754
261	71984.208518
262	72602.974289
263	73187.849765
264	73596.807715
265	73772.426360
266	73768.759589
267	73621.724262
268	73416.137237
269	73202.159649
270	73024.703974
271	72905.326159

Forecasting For Price

Data Preprocessing

```
df=pd.read_csv('/content/drive/MyDrive/MarketPricePrediction.csv')
```

```
df.columns=df.columns.str.lower()
```

```
df.columns
```

```
Index(['market', 'month', 'year', 'quantity', 'pricemin', 'pricemax',
       'pricemod', 'state', 'city', 'date'],
      dtype='object')
```

```
df['median_price']=df[['pricemin', 'pricemax','pricemod']].median(  axis=1)
```

```
df['date']=pd.to_datetime(df['date'])
```

```
df.set_index('date', inplace=True)
```

```
# Sort the data by date
df.sort_index(inplace=True)
df.head()
```

date	market	month	year	quantity	pricemin	pricemax	pricemod	state	city	m
1996-01-01	LASALGAON(MS)	January	1996	225063	160	257	226	MS	LASALGAON	
1996-02-01	LASALGAON(MS)	February	1996	196164	133	229	186	MS	LASALGAON	
1996-03-01	LASALGAON(MS)	March	1996	178992	155	274	243	MS	LASALGAON	

Next steps: Generate code with df View recommended plots

```
df.drop(['market', 'month', 'year', 'quantity', 'pricemin', 'pricemax','pricemod', 'state', 'city'],axi
```

```
df
```

date	median_price
1996-01-01	226.0
1996-02-01	186.0
1996-03-01	243.0
1996-04-01	254.0
1996-05-01	269.0
...	...
2016-02-01	1077.0
2016-02-01	575.0
2016-02-01	730.0
2016-02-01	806.0
2016-02-01	1309.0

10227 rows × 1 columns

Next steps: Generate code with df View recommended plots

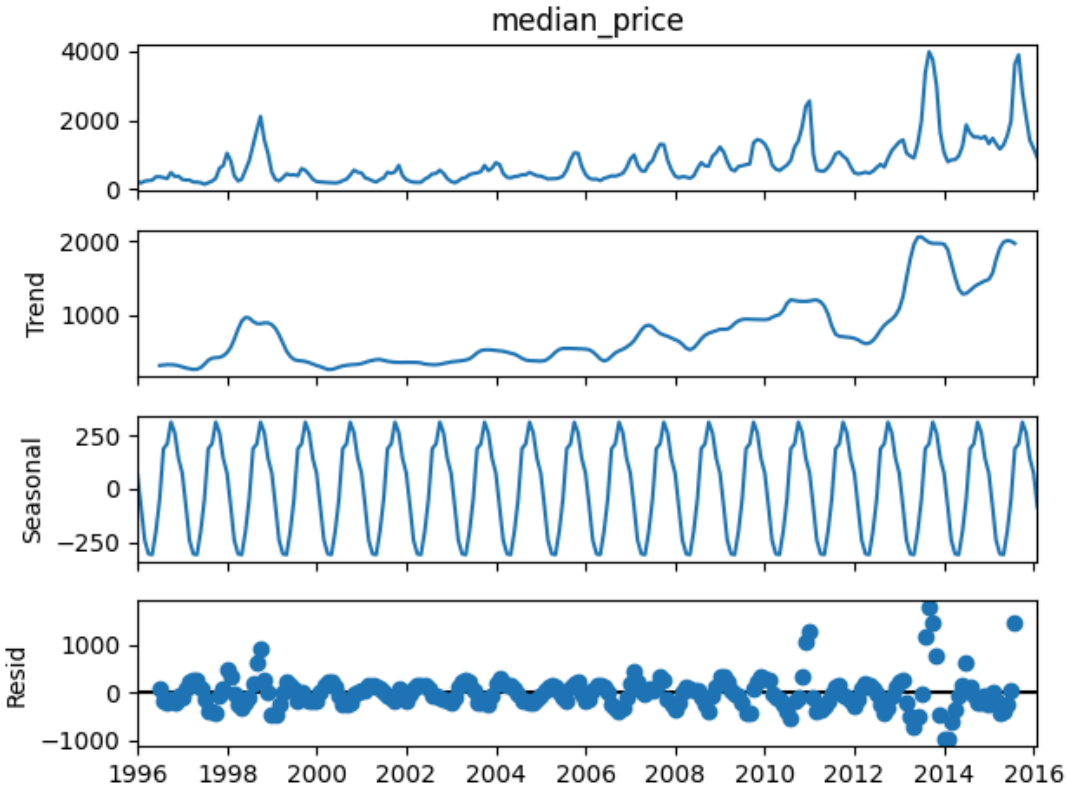
```
# Remove duplicate indices by aggregating data (e.g., take mean for repeated dates)
df = df.groupby(df.index).mean()

# If the index is not in datetime format, convert it
if not pd.api.types.is_datetime64_any_dtype(df.index):
    df.index = pd.to_datetime(df.index)


df
```

date	median_price
1996-01-01	226.000000
1996-02-01	186.000000
1996-03-01	243.000000
1996-04-01	254.000000
1996-05-01	269.000000
...	...
2015-10-01	2838.788889
2015-11-01	2101.056180
2015-12-01	1427.516129
2016-01-01	1186.457831
2016-02-01	926.345679

242 rows × 1 columns

Next steps: Generate code with df View recommended plots

```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_data= seasonal_decompose(df['median_price'],model='additive')
decompose_data.plot();
```



Testing For Stationarity

```
test_result=adfuller(df['median_price'])


#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(median_price):
    result=adfuller(median_price)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non



adfuller_test(df['median_price'])
```

```
ADF Test Statistic : -1.5529565759212154
p-value : 0.5071710159736902
#Lags Used : 13
Number of Observations Used : 228
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

```
## Use Techniques Differencing
df['median_price First Difference']=df['median_price']-df['median_price'].shift(1)
```

```
df.head()
```

date	median_price	median_price First Difference
1996-01-01	226.0	NaN
1996-02-01	186.0	-40.0
1996-03-01	243.0	57.0
1996-04-01	254.0	11.0
1996-05-01	269.0	15.0

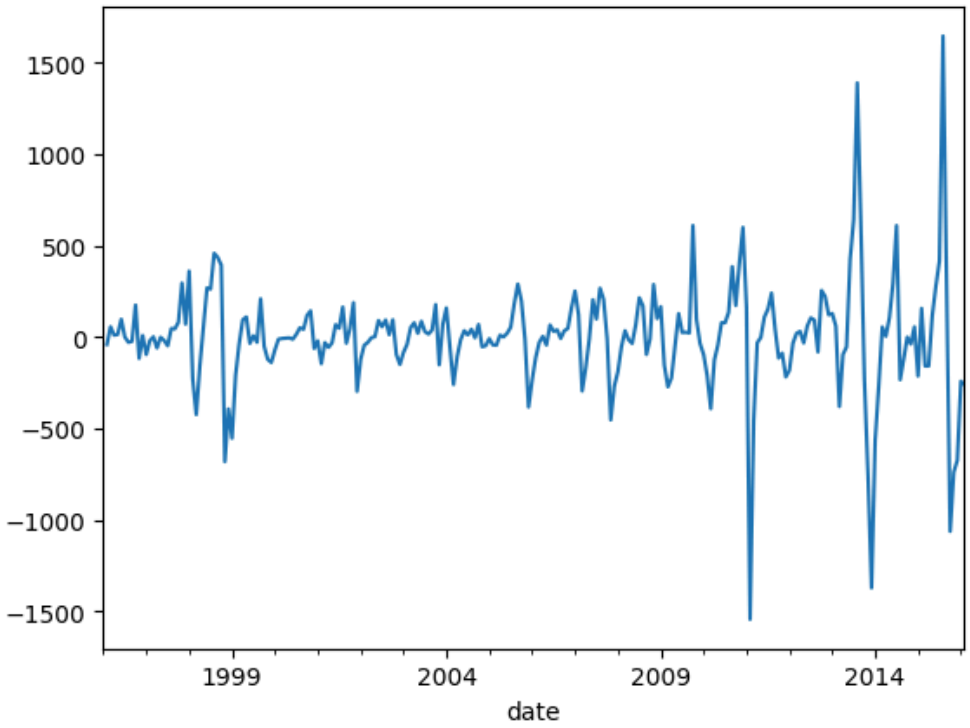
Next steps: Generate code with `df` View recommended plots

```
adfuller_test(df['median_price First Difference'].dropna())
```

```
ADF Test Statistic : -5.28728699416396
p-value : 5.817627863234602e-06
#Lags Used : 12
Number of Observations Used : 228
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root
```

```
df['median_price First Difference'].plot()
```

```
<Axes: xlabel='date'>
```




```
## Use Techniques Differencing for period=12
df['median_price 12 Difference']=df['median_price']-df['median_price'].shift(12)
```

```
df.head(13)
```

date	median_price	median_price First Difference	median_price 12 Difference
1996-01-01	226.0	NaN	NaN
1996-02-01	186.0	-40.0	NaN
1996-03-01	243.0	57.0	NaN
1996-04-01	254.0	11.0	NaN
1996-05-01	269.0	15.0	NaN
1996-06-01	367.0	98.0	NaN
1996-07-01	368.0	1.0	NaN
1996-08-01	340.0	-28.0	NaN
1996-09-01	317.0	-23.0	NaN
1996-10-01	492.0	175.0	NaN
1996-11-01	376.0	-116.0	NaN
1996-12-01	385.0	9.0	NaN
1997-01-01	290.0	-95.0	64.0

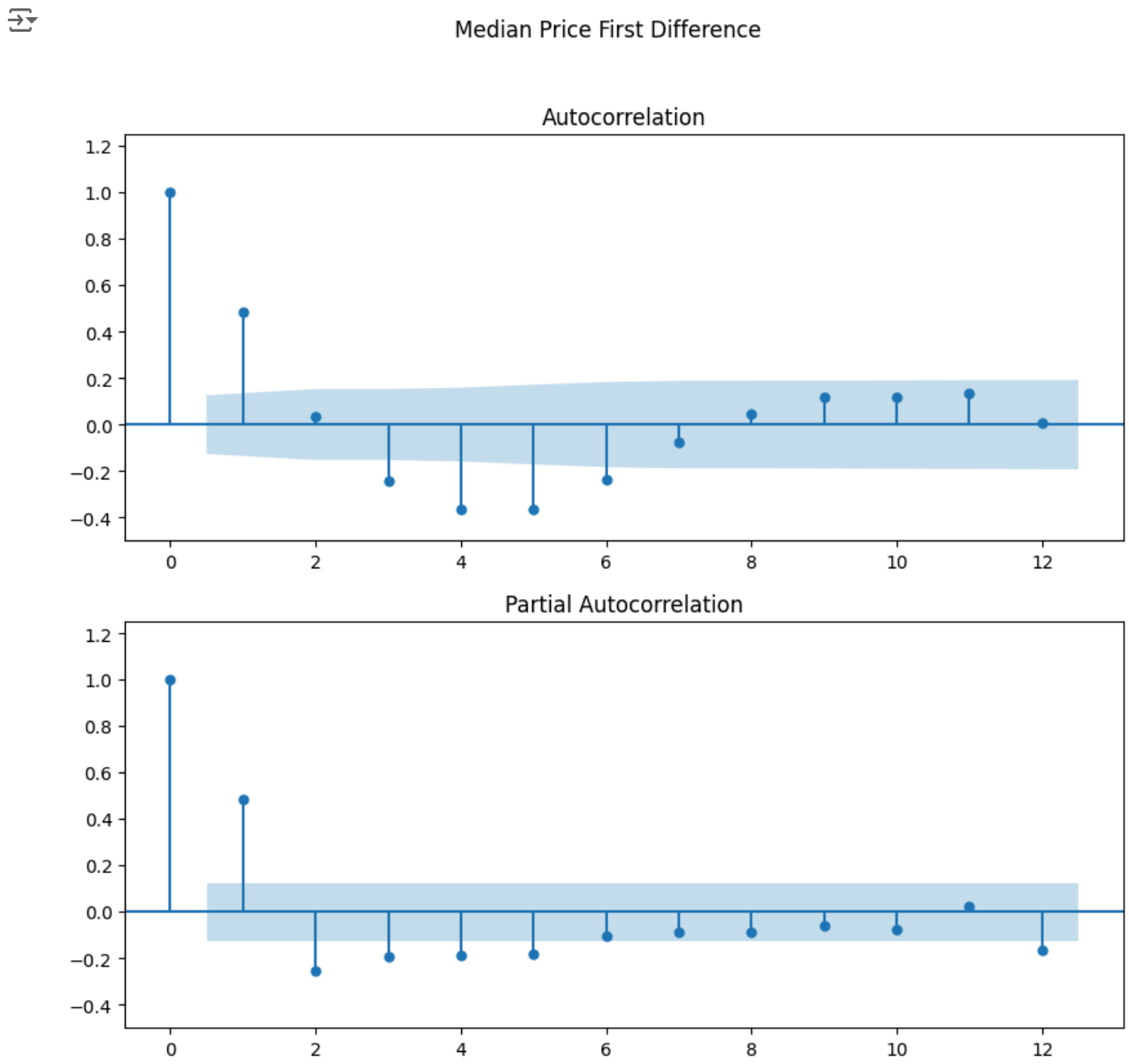
Next steps: Generate code with df View recommended plots

```
adfuller_test(df['median_price 12 Difference'].dropna())
```

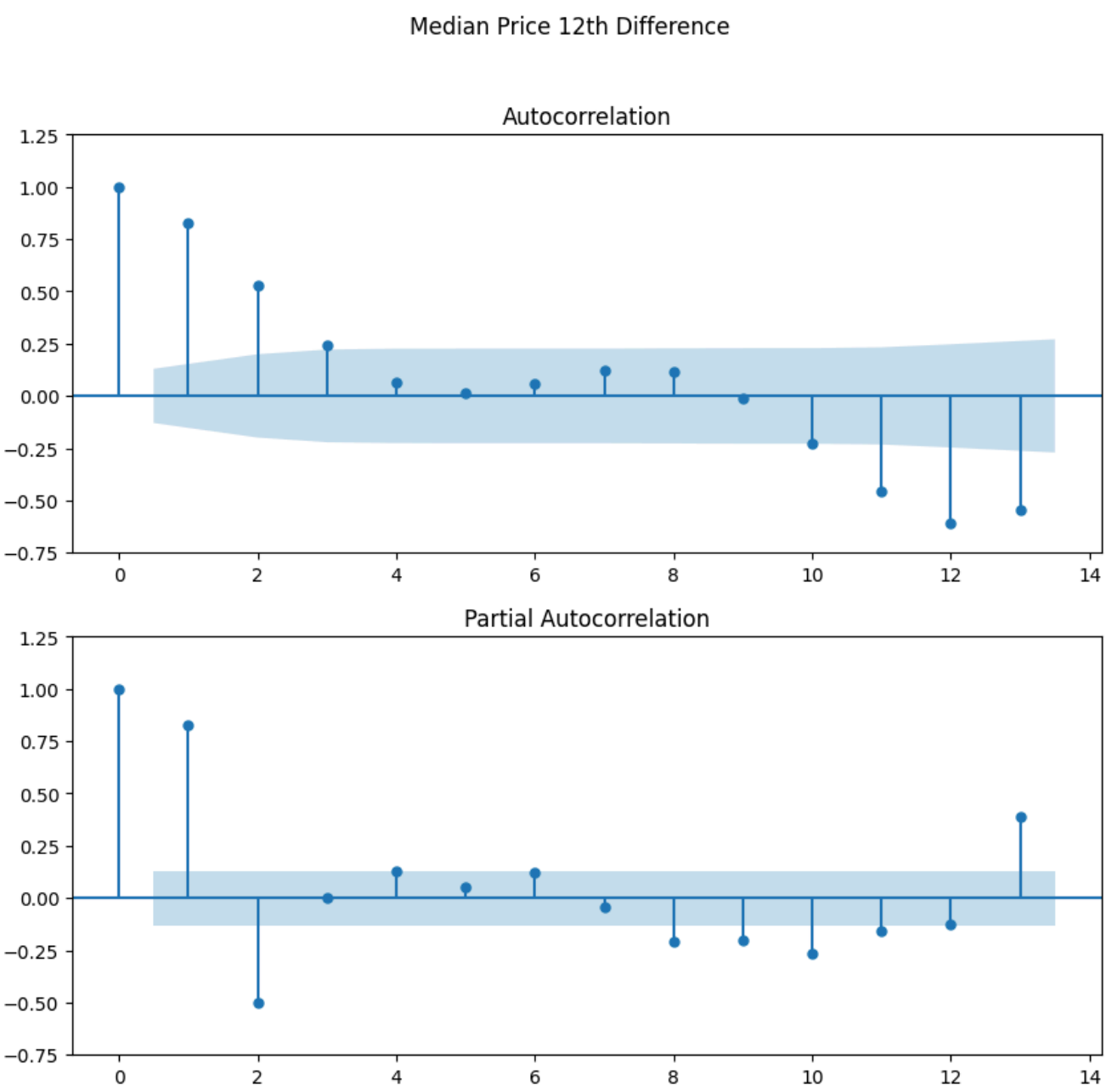
```
ADF Test Statistic : -5.184876242816724
p-value : 9.43915517882745e-06
#Lags Used : 13
Number of Observations Used : 216
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root
```

Plotting ACF and PACF graphs for Price

```
fig = plt.figure(figsize=(10,9))
fig.suptitle('Median Price First Difference')
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['median_price First Difference'].dropna(),lags=12,ax=ax1)
ax1.set_ylim(-0.50, 1.25);
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['median_price First Difference'].dropna(),lags=12,ax=ax2)
ax2.set_ylim(-0.50, 1.25);
```




```
fig = plt.figure(figsize=(10,9))
fig.suptitle('Median Price 12th Difference')
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['median_price 12 Difference'].dropna(),lags=13,ax=ax1)
ax1.set_ylim(-0.75, 1.25);
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['median_price 12 Difference'].dropna(),lags=13,ax=ax2)
ax2.set_ylim(-0.75, 1.25);
```



ARIMA model for Price

```
# For non-seasonal data
#p=9, d=1, q=0

model=ARIMA(df['median_price'],order=(9,1,0))
model_fit=model.fit()


model_fit.summary()
```

SARIMAX Results

Dep. Variable:	median_price	**No. Observations:**	242
Model:	ARIMA(9, 1, 0)	**Log Likelihood**	-1651.289
Date:	Fri, 31 May 2024	**AIC**	3322.579
Time:	20:57:38	**BIC**	3357.427
Sample:	01-01-1996 - 02-01-2016	**HQIC**	3336.618

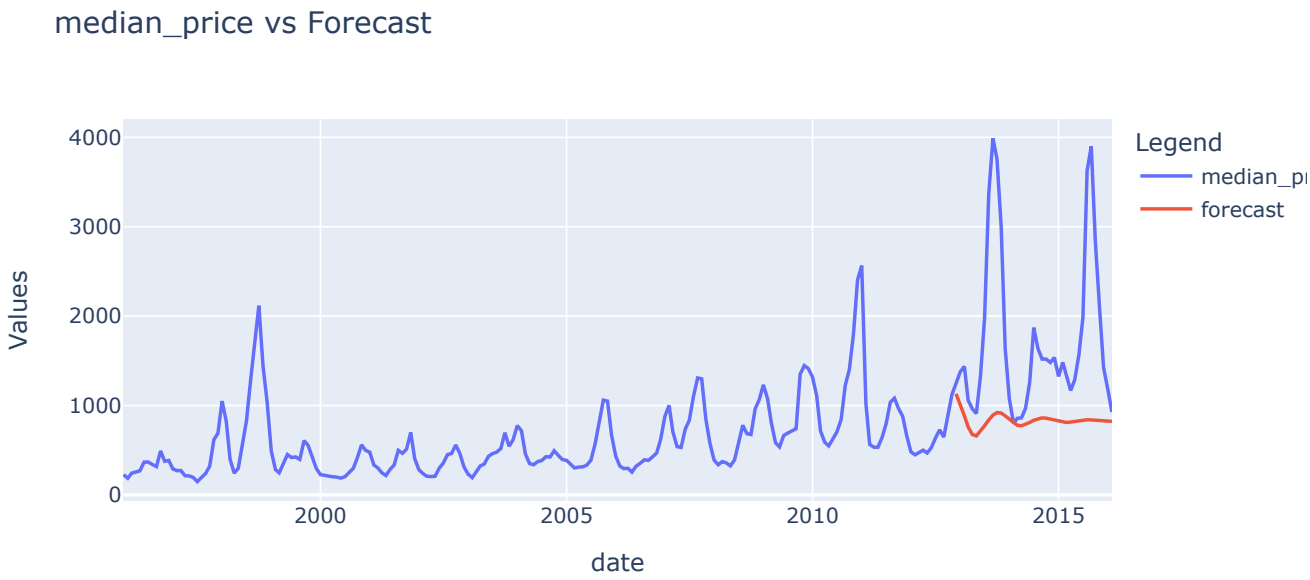
Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4170	0.048	8.693	0.000	0.323	0.511
ar.L2	-0.2212	0.051	-4.377	0.000	-0.320	-0.122
ar.L3	-0.1720	0.085	-2.027	0.043	-0.338	-0.006
ar.L4	-0.1623	0.070	-2.333	0.020	-0.299	-0.026
ar.L5	-0.1896	0.090	-2.102	0.036	-0.366	-0.013
ar.L6	-0.1236	0.086	-1.435	0.151	-0.293	0.045
ar.L7	-0.0723	0.090	-0.803	0.422	-0.249	0.104
ar.L8	-0.0621	0.073	-0.846	0.398	-0.206	0.082
ar.L9	-0.0748	0.094	-0.798	0.425	-0.259	0.109
sigma2	5.209e+04	2664.215	19.553	0.000	4.69e+04	5.73e+04

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	1905.72
Prob(Q):	0.92	**Prob(JB):**	0.00
Heteroskedasticity (H):	5.23	**Skew:**	0.77
Prob(H) (two-sided):	0.00	**Kurtosis:**	16.69

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
# Generate forecast
start_date = datetime.datetime(2012, 12, 1)
end_date = datetime.datetime(2016, 2, 1)
forecasted_date= datetime.datetime(2017, 2, 1)


df['forecast']=model_fit.predict(start=start_date,end=end_date,dynamic=True)

fig = px.line(df, y=['median_price', 'forecast'], labels={'value': 'Values', 'variable': 'Legend'}, tit
fig.update_layout(width=800, height=400)
fig.show()
```

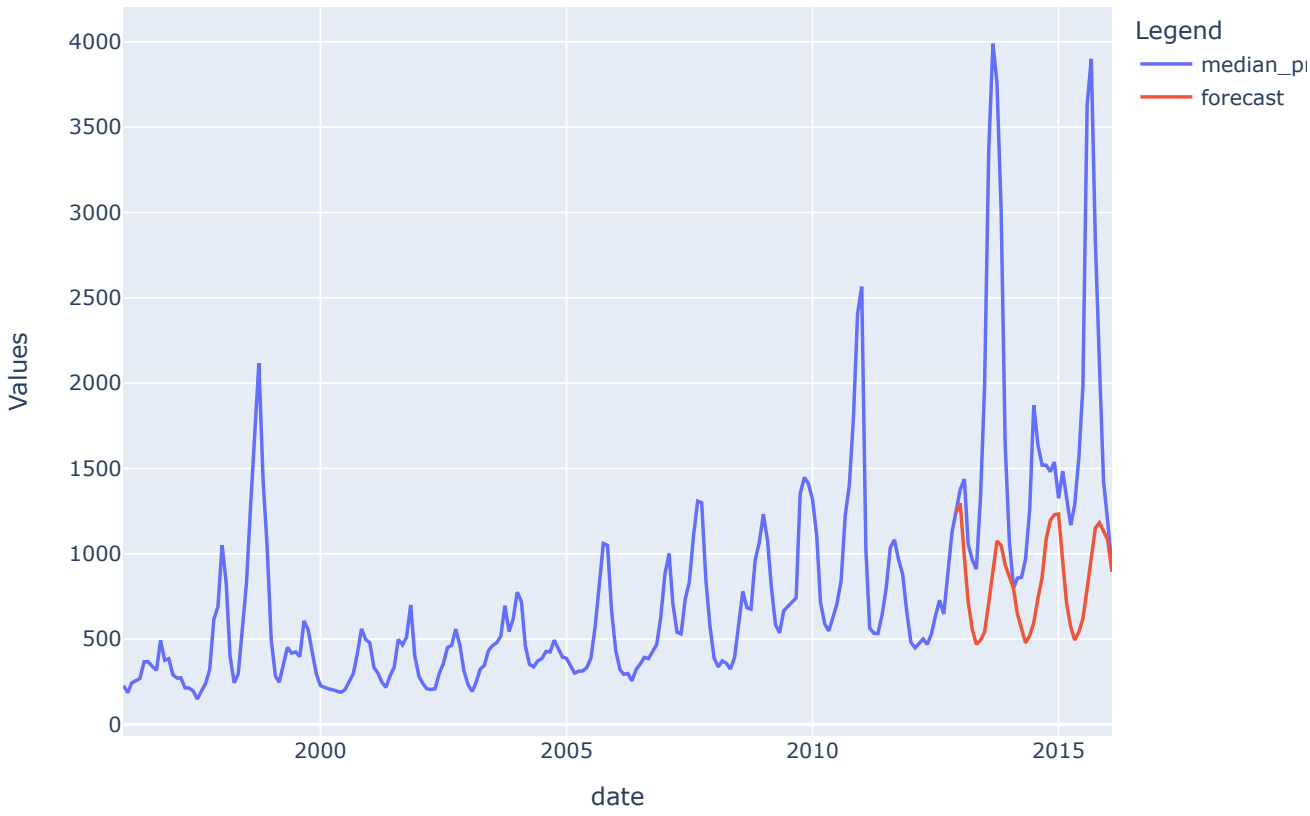



```
# Calculate error metrics
actual = df['median_price'][start_date:end_date]
forecast = df['forecast'][start_date:end_date]

ARIMA_price_mae = mean_absolute_error(actual, forecast)
ARIMA_price_mse = mean_squared_error(actual, forecast)
ARIMA_price_rmse = np.sqrt(ARIMA_price_mse)

print(f'ARIMA_price_MAE: {ARIMA_price_mae}')
print(f'ARIMA_price_MSE: {ARIMA_price_mse}')
print(f'ARIMA_price_RMSE: {ARIMA_price_rmse}')
```

```
ARIMA_price_MAE: 899.8626804617461
ARIMA_price_MSE: 1582561.8574288574
ARIMA_price_RMSE: 1257.9991484213563
```

SARIMAX Model for Price

```
# for seasonal data
# p=2 ,d=1 ,q=3
model=sm.tsa.statespace.SARIMAX(df['median_price'],order=(2,1,3),seasonal_order=(2,1,3,12))
results=model.fit()


df['forecast']=results.predict(start=start_date,end=end_date,dynamic=True)

fig = px.line(df, y=['median_price', 'forecast'], labels={'value': 'Values', 'variable': 'Legend'}, tit
fig.update_layout(width=800, height=400)
fig.show()
```




```
# Calculate error metrics
actual = df['median_price'][start_date:end_date]
forecast = df['forecast'][start_date:end_date]


SARIMAX_price_mae = mean_absolute_error(actual, forecast)
SARIMAX_price_mse = mean_squared_error(actual, forecast)
SARIMAX_price_rmse = np.sqrt(SARIMAX_price_mse)


print(f'SARIMAX_price_MAE: {SARIMAX_price_mae}')
print(f'SARIMAX_price_MSE: {SARIMAX_price_mse}')
print(f'SARIMAX_price_RMSE: {SARIMAX_price_rmse}')
```

```
SARIMAX_price_MAE: 890.3167866259508
SARIMAX_price_MSE: 1555605.8207442826
SARIMAX_price_RMSE: 1247.2392796670101
```

.

.

.

.

.

.

PROPHET Model for Price

```
#Initializing model

model=Prophet()
model
```

```
<prophet.forecaster.Prophet at 0x7f697a7d6650>
```

```
df.columns
```

```
Index(['median_price', 'median_price First Difference',
       'median_price 12 Difference', 'forecast'],
      dtype='object')
```

```
df.drop(['median_price First Difference','median_price 12 Difference', 'forecast'],axis=1,inplace=True)
```

```
df= df.reset_index()
```

```
df.head()
```

	date	median_price
0	1996-01-01	226.0
1	1996-02-01	186.0
2	1996-03-01	243.0
3	1996-04-01	254.0
4	1996-05-01	269.0

Next steps: Generate code with df | View recommended plots

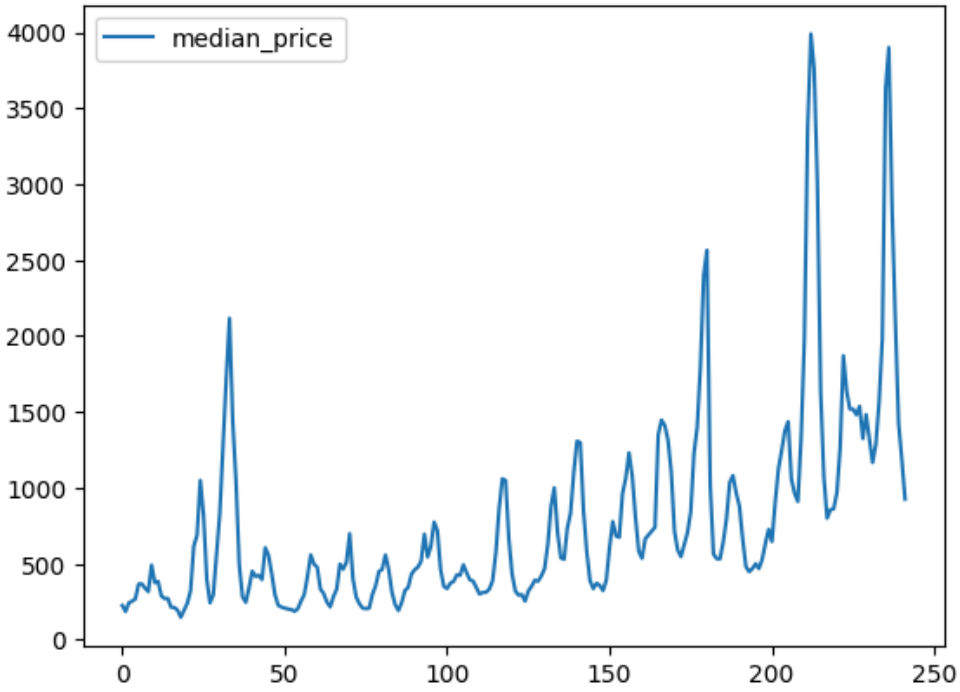
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242 entries, 0 to 241
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          242 non-null    datetime64[ns]
 1   median_price  242 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 3.9 KB
```

```
df['date']=df['date'].astype(str)
```

```
df[['date','median_price']].plot()
```

<Axes: >




```
df.columns = ['ds','y']
df.head()
```

	ds	y
0	1996-01-01	226.0
1	1996-02-01	186.0
2	1996-03-01	243.0
3	1996-04-01	254.0
4	1996-05-01	269.0

Next steps: Generate code with `df` View recommended plots

```
df.tail()
```

	ds	y
237	2015-10-01	2838.788889
238	2015-11-01	2101.056180
239	2015-12-01	1427.516129
240	2016-01-01	1186.457831
241	2016-02-01	926.345679

```
df['ds'] = pd.to_datetime(df['ds'])
df.head()
```

	ds	y
0	1996-01-01	226.0
1	1996-02-01	186.0
2	1996-03-01	243.0
3	1996-04-01	254.0
4	1996-05-01	269.0

Next steps: Generate code with df View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242 entries, 0 to 241
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   ds      242 non-null    datetime64[ns]
 1   y       242 non-null    float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 3.9 KB
```

```
model.fit(df)
```

Show hidden output

```
model.component_modes
```

```
{'additive': ['yearly',
  'additive_terms',
  'extra_regressors_additive',
  'holidays'],
 'multiplicative': ['multiplicative_terms', 'extra_regressors_multiplicative']}
```

```
df.tail()
```

	ds	y
237	2015-10-01	2838.788889
238	2015-11-01	2101.056180
239	2015-12-01	1427.516129
240	2016-01-01	1186.457831
241	2016-02-01	926.345679

```
### Create future dates of 365 days
future_dates=model.make_future_dataframe(periods=365)
```

```
future_dates.tail()
```

	ds
602	2017-01-27
603	2017-01-28
604	2017-01-29
605	2017-01-30
606	2017-01-31

```
prediction=model.predict(future_dates)
```

```
prediction.head()
```

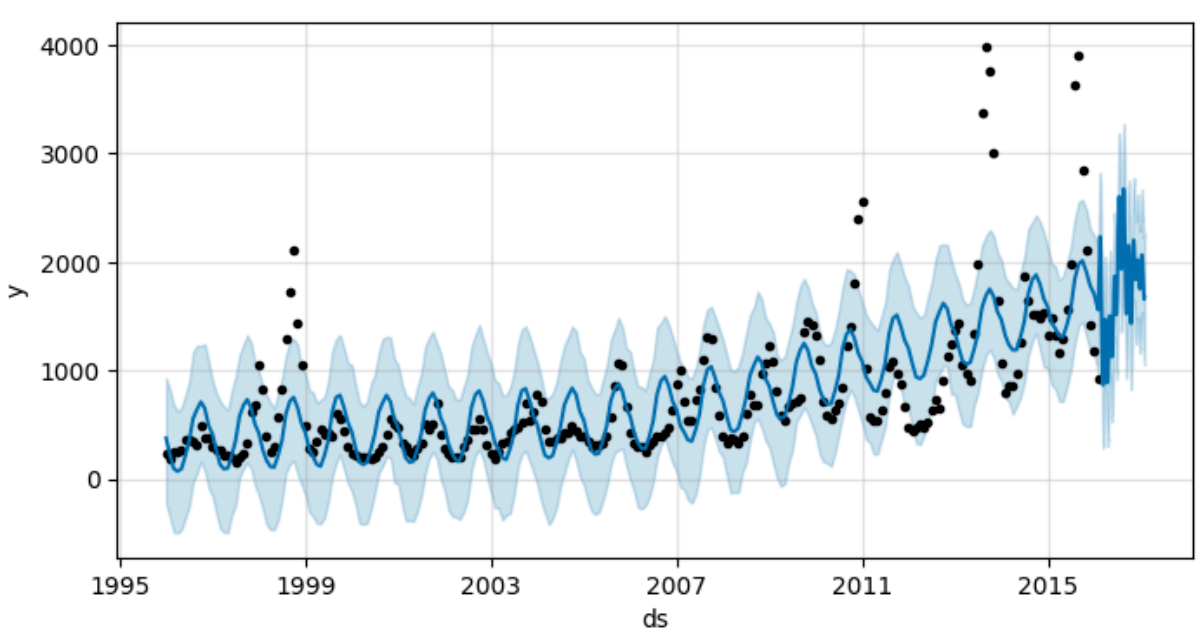
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_ter
0	1996-01-01	363.453150	-218.747541	928.125890	363.453150	363.453150	12.495211	1
1	1996-02-01	364.750163	-363.545541	818.904069	364.750163	364.750163	-142.954126	-14
2	1996-03-01	365.963498	-488.701077	682.006641	365.963498	365.963498	-271.496992	-27
3	1996-04-01	367.260511	-499.889688	622.331204	367.260511	367.260511	-298.463934	-29
4	1996-05-01	368.515685	-467.282127	655.944611	368.515685	368.515685	-276.826175	-27

Next steps: Generate code with prediction | View recommended plots

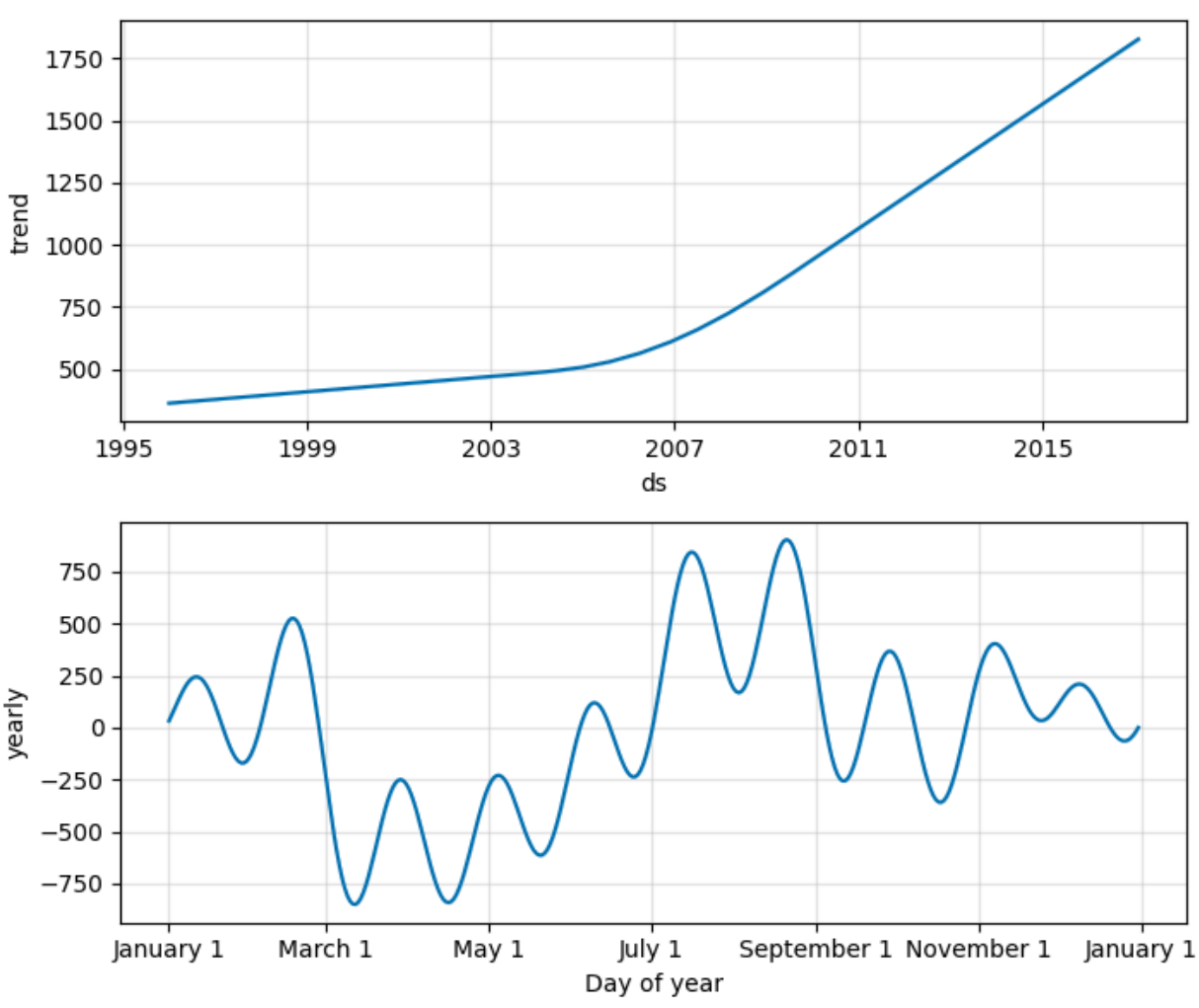
```
# Add forecast to the original dataframe
df['prediction'] = prediction['yhat'].iloc[:len(df)]

# Define start and end dates for error metrics calculation
start_date = '2012-12-01'
end_date = '2016-02-01'

# Filter actual and forecasted values
actual = df.loc[(df['ds'] >= start_date) & (df['ds'] <= end_date), 'y']
forecasted = df.loc[(df['ds'] >= start_date) & (df['ds'] <= end_date), 'prediction']

# Calculate error metrics
Prophet_price_mae = mean_absolute_error(actual, forecasted)
Prophet_price_mse = mean_squared_error(actual, forecasted)
Prophet_price_rmse = np.sqrt(Prophet_price_mse)

print(f'Prophet_price_MAE: {Prophet_price_mae}')
print(f'Prophet_price_MSE: {Prophet_price_mse}')
print(f'Prophet_price_RMSE: {Prophet_price_rmse}')
```

```
Prophet_price_MAE: 508.81715995993176
Prophet_price_MSE: 635490.9253549961
Prophet_price_RMSE: 797.1768469762504
```

```
### plot the predicted projection
fig, ax = plt.subplots(figsize=(8, 4))
model.plot(prediction,ax=ax);
```




```
#### Visualize Each Components[Trends,yearly]
components_fig = model.plot_components(prediction)
components_fig.set_size_inches(7,6)
```



Cross Validation using PROPHET

```
from prophet.diagnostics import cross_validation

df_cv = cross_validation(model, initial='730 days', period='180 days', horizon = '365 days')
```

Show hidden output

```
df_cv.head()
```

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	1998-06-01	569.859729	388.195938	752.036199	566.555556	1998-05-01
1	1998-07-01	448.848941	258.618118	639.576349	829.111111	1998-05-01
2	1998-08-01	450.227755	282.970474	639.421152	1287.555556	1998-05-01
3	1998-09-01	427.149774	242.350274	605.872593	1723.666667	1998-05-01
4	1998-10-01	270.374505	96.226261	454.816849	2116.900000	1998-05-01

Next steps: Generate code with df_cv | View recommended plots

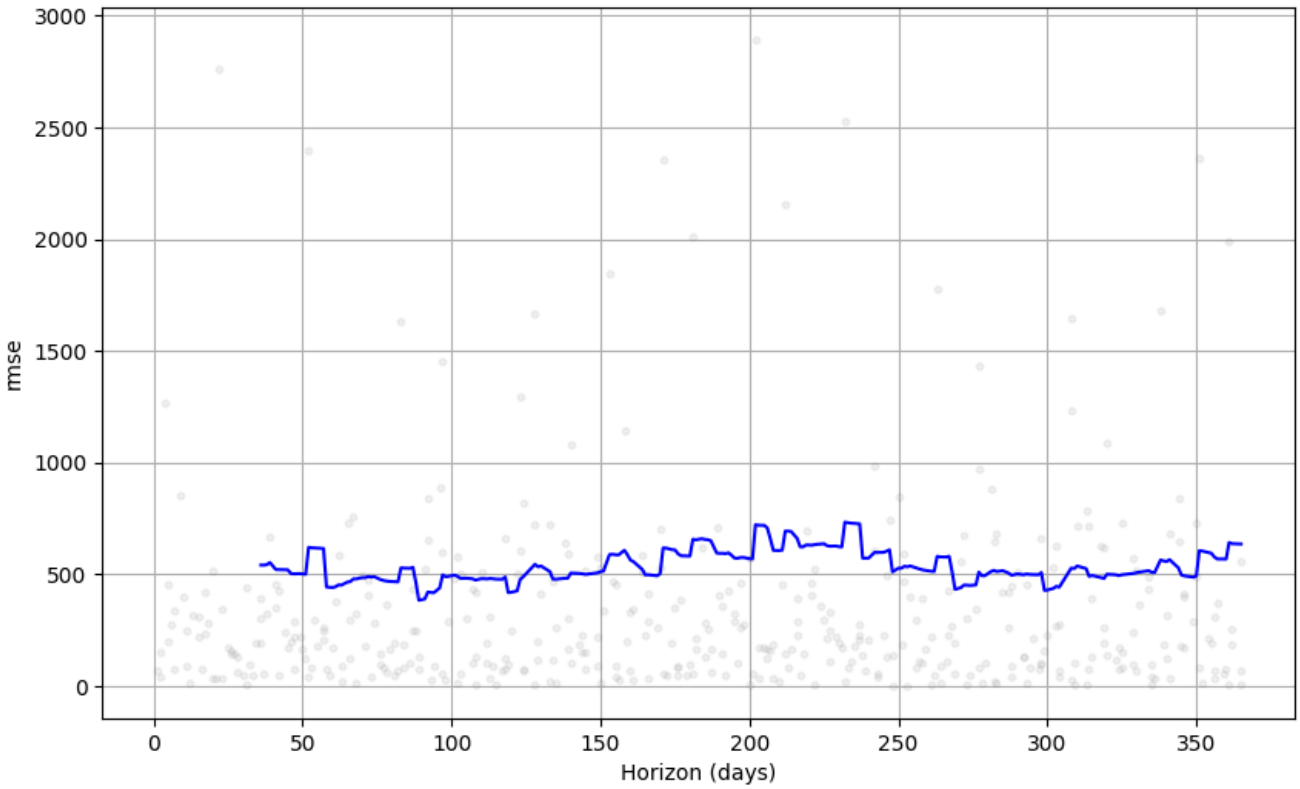
```
from prophet.diagnostics import performance_metrics
df_p = performance_metrics(df_cv)
```

```
df_p.head()
```

	horizon	mse	rmse	mae	mape	mdape	smape	coverage
0	36 days	291827.236583	540.210363	298.249264	0.399587	0.332837	0.368917	0.809524
1	37 days	291791.790438	540.177555	297.965899	0.401686	0.332837	0.371542	0.809524
2	38 days	293983.473144	542.202428	303.400475	0.410368	0.351450	0.375514	0.809524
3	39 days	304224.465425	551.565468	316.989166	0.458030	0.351450	0.393605	0.785714
4	41 days	270615.043034	520.206731	297.957247	0.438924	0.349330	0.364202	0.797619

Next steps: Generate code with df_p | View recommended plots

```
from prophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='rmse')
fig.set_size_inches(10, 6)
```



LSTM Model for Price

```
df=pd.read_csv('/content/drive/MyDrive/MarketPricePrediction.csv')
```

```
#Convert date column to datetime
df['date'] = pd.to_datetime(df['date'])
df.tail()
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city	date
10222	YEOLA(MS)	December	2011	131326	282	612	526	MS	YEOLA	2011-12-01
10223	YEOLA(MS)	December	2012	207066	485	1327	1136	MS	YEOLA	2012-12-01
10224	YEOLA(MS)	December	2013	215883	472	1427	1177	MS	YEOLA	2013-12-01

```
df.set_index('date', inplace=True)
df.head()
```

date	market	month	year	quantity	priceMin	priceMax	priceMod	state	city
2005-01-01	ABOHAR(PB)	January	2005	2350	404	493	446	PB	ABOHAR
2006-01-01	ABOHAR(PB)	January	2006	900	487	638	563	PB	ABOHAR
2010-01-01	ABOHAR(PB)	January	2010	790	1283	1592	1460	PB	ABOHAR
2011-01-01	ABOHAR(PB)	January	2011	245	3067	3750	3433	PB	ABOHAR
2012-01-01	ABOHAR(PB)	January	2012	1035	523	686	605	PB	ABOHAR

Next steps: Generate code with df View recommended plots

```
# Sort the data by date
df.sort_index(inplace=True)
df.head()
```

date	market	month	year	quantity	priceMin	priceMax	priceMod	state	city
1996-01-01	LASALGAON(MS)	January	1996	225063	160	257	226	MS	LASALGAON
1996-02-01	LASALGAON(MS)	February	1996	196164	133	229	186	MS	LASALGAON
1996-03-01	LASALGAON(MS)	March	1996	178992	155	274	243	MS	LASALGAON

Next steps: Generate code with df View recommended plots

```
df.columns=df.columns.str.lower()


# Calculate Median Price
df['median_price'] = df[['pricemin', 'pricemax', 'pricemod']].median(axis=1)


df1=df.reset_index()['median_price']


df1
```

```
0         226.0
1         186.0
2         243.0
3         254.0
4         269.0
          ...
10222    1077.0
10223     575.0
10224     730.0
10225     806.0
10226    1309.0
Name: median_price, Length: 10227, dtype: float64
```

```
# Remove duplicate indices by aggregating data (e.g., take mean for repeated dates)
df1 = df1.groupby(df.index).mean()

# If the index is not in datetime format, convert it
if not pd.api.types.is_datetime64_any_dtype(df.index):
    df1.index = pd.to_datetime(df1.index)


df1.tail()
```

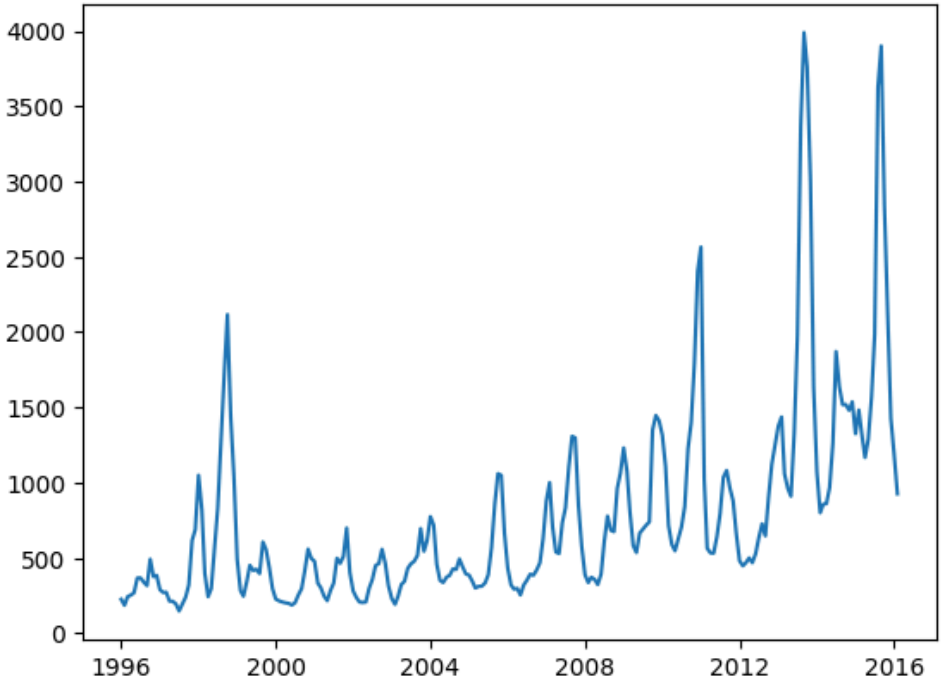
```
date
2015-10-01    2838.788889
2015-11-01    2101.056180
2015-12-01    1427.516129
2016-01-01    1186.457831
2016-02-01     926.345679
Name: median_price, dtype: float64
```

```
df1.shape
```

```
(242,)
```

```
# Plotting df1
plt.plot(df1)
```

```
[<matplotlib.lines.Line2D at 0x7f697a6d47c0>]
```




```
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler


scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))


print(df1)
```

Show hidden output

```
##splitting dataset into train and test split
training_size=int(len(df1)*0.80)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

```
training_size,test_size
```

```
(193, 49)
```

```
train_data
```

Show hidden output

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 5
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

```
print(X_train.shape), print(y_train.shape)
```

```
(187, 5)
(187,)
(None, None)
```

```
print(X_test.shape), print(ytest.shape)
```

```
(43, 5)
(43,)
(None, None)
```

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

```
# Define the learning rate
learning_rate = 0.001  # You can adjust this value as needed

# Create the optimizer with the custom learning rate
optimizer = Adam(learning_rate=learning_rate)
```

```
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(5, 1)))
model.add(Dropout(0.2))  # Add dropout layer with 20% dropout rate
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=optimizer)
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 5, 50)             10400

 dropout_4 (Dropout)         (None, 5, 50)             0

 lstm_7 (LSTM)               (None, 5, 50)             20200

 dropout_5 (Dropout)         (None, 5, 50)             0

 lstm_8 (LSTM)               (None, 50)                20200

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=500,batch_size=64,verbose=1)
```

Show hidden output

```
## Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
6/6 [==============================] - 1s 4ms/step
2/2 [==============================] - 0s 7ms/step
```

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

Calculating MAE, MSE, RMSE

```
# Calculate error metrics for the test set
LSTM_price_mae = mean_absolute_error(ytest, test_predict)
LSTM_price_mse = mean_squared_error(ytest, test_predict)
LSTM_price_rmse = np.sqrt(LSTM_price_mse)

print(f'LSTM_price_MAE: {LSTM_price_mae}')
print(f'LSTM_price_MSE: {LSTM_price_mse}')
print(f'LSTM_price_RMSE: {LSTM_price_rmse}')
```

```
LSTM_price_MAE: 1200.4718132514781
LSTM_price_MSE: 2086461.8160530075
LSTM_price_RMSE: 1444.459004628725
```

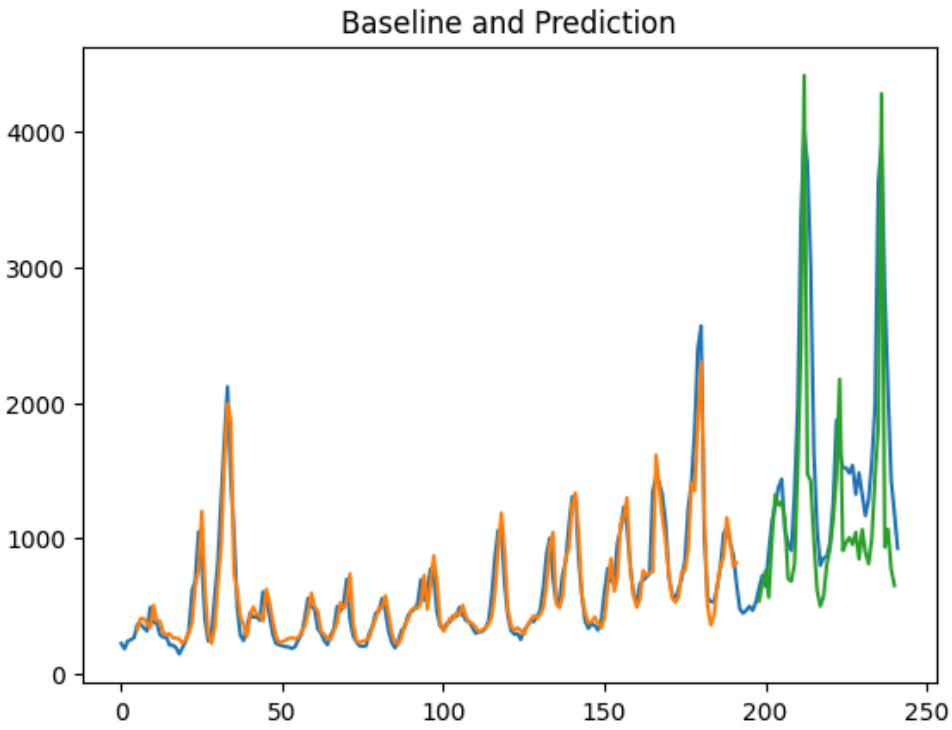
```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
684.4847932952464
```

```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

```
1444.459004628725
```

```
### Plotting
# shift train predictions for plotting
look_back=5
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.title('Baseline and Prediction')
plt.show()
```




```
len(test_data)
```

```
49
```

```
x_input=test_data[44:].reshape(1,-1)
x_input.shape
```

```
(1, 5)
```

```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

```
temp_input
```

```
# demonstrate prediction for next 30 days
from numpy import array

lst_output=[]
n_steps=5
i=0
while(i<30):

    if(len(temp_input)>5):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

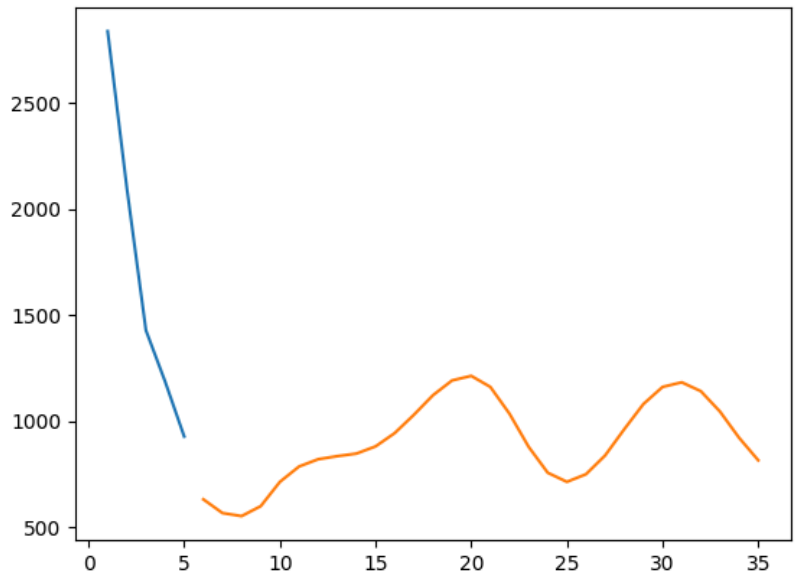
Show hidden output

```
day_new=np.arange(1,6)
day_pred=np.arange(6,36)


len(df1)
```

242

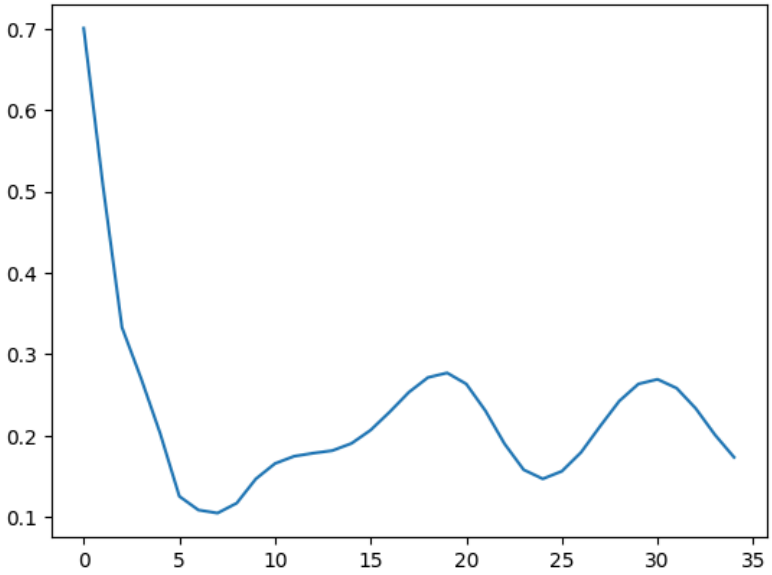
```
plt.plot(day_new,scaler.inverse_transform(df1[237:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

[<matplotlib.lines.Line2D at 0x7dc536d18490>]




```
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[237:])
```

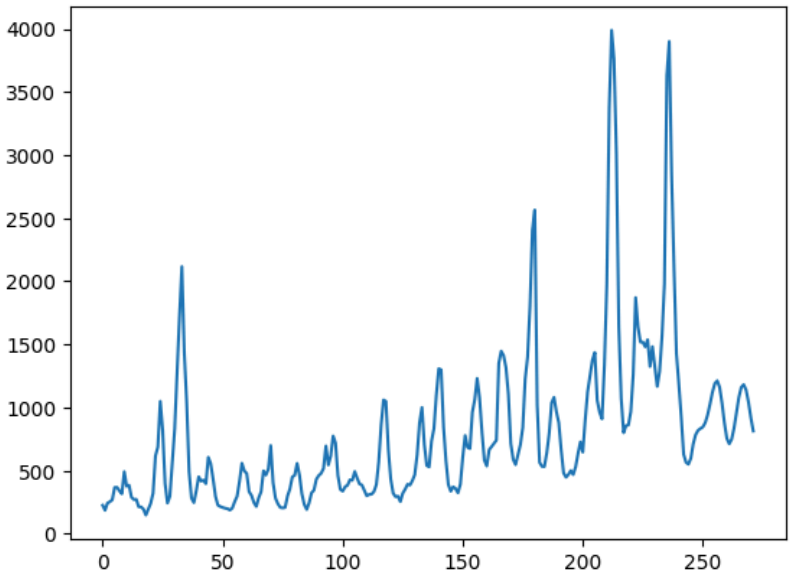
```
[<matplotlib.lines.Line2D at 0x7dc536e50490>]
```




```
df3=scaler.inverse_transform(df3).tolist()
```

```
plt.plot(df3)
```

```
[<matplotlib.lines.Line2D at 0x7dc536deb0a0>]
```




```
# prompt: convert df3 to dataframe

import pandas as pd
df3 = pd.DataFrame(df3, columns=['median_price'])
```

```
df3.tail()
```

	median_price
267	1182.164749
268	1140.297644
269	1044.195535
270	920.252155
271	814.275024

MODEL Evaluation for both Quantity and Price Forecasting

```
# Create a dictionary to store the error metrics for quantity
quantity_metrics = {
    'Model': ['ARIMA', 'SARIMAX', 'PROPHET', 'LSTM'],
    'MAE': [ARIMA_Qty_mae, SARIMAX_Qty_mae, Prophet_Qty_mae, LSTM_Qty_mae],
    'MSE': [ARIMA_Qty_mse, SARIMAX_Qty_mse, Prophet_Qty_mse, LSTM_Qty_mse],
    'RMSE': [ARIMA_Qty_rmse, SARIMAX_Qty_rmse, Prophet_Qty_rmse, LSTM_Qty_rmse]
}

# Create a DataFrame for quantity metrics
quantity_df = pd.DataFrame(quantity_metrics)

quantity_df=quantity_df.set_index('Model')

display(Markdown("**Evaluation of Quantity Metrics:**"))
quantity_df.head()
```

Evaluation of Quantity Metrics:

Model	MAE	MSE	RMSE
ARIMA	18077.992139	5.159257e+08	22713.997036
SARIMAX	11753.369780	2.033790e+08	14261.101350
PROPHET	11432.407343	1.991221e+08	14111.062194
LSTM	76176.445171	6.053489e+09	77804.172737

Next steps: Generate code with `quantity_df` View recommended plots

```
# Create a dictionary to store the error metrics for price
price_metrics = {
    'Model': ['ARIMA', 'SARIMAX', 'PROPHET', 'LSTM'],
    'MAE': [ARIMA_price_mae, SARIMAX_price_mae, Prophet_price_mae, LSTM_price_mae],
    'MSE': [ARIMA_price_mse, SARIMAX_price_mse, Prophet_price_mse, LSTM_price_mse],
    'RMSE': [ARIMA_price_rmse, SARIMAX_price_rmse, Prophet_price_rmse, LSTM_price_rmse]
}

# Create a DataFrame for price metrics
price_df = pd.DataFrame(price_metrics)
price_df=price_df.set_index('Model')
display(Markdown("**Evaluation of Price Metrics:**"))
price_df.head()
```

Evaluation of Price Metrics:

Model	MAE	MSE	RMSE
ARIMA	899.862680	1.582562e+06	1257.999148
SARIMAX	890.316787	1.555606e+06	1247.239280
PROPHET	508.817160	6.354909e+05	797.176847
LSTM	1320.470731	2.442432e+06	1562.828349

Next steps: Generate code with `price_df` View recommended plots

CONCLUSION

Quantity Forecasting

1. The SARIMAX and Prophet models outperform ARIMA and LSTM in terms of MAE, MSE, and RMSE metrics for quantity forecasting.
2. Among the models, Prophet demonstrates the lowest error metrics, indicating its superior performance in accurately predicting the quantity of commodities for future months.
3. LSTM, although visually accurate, exhibits significantly higher error metrics compared to the other models, suggesting potential overfitting or limitations in capturing the underlying patterns in the data.

Price Forecasting

1. Prophet emerges as the top-performing model for price forecasting, with the lowest MAE, MSE, and RMSE values.
2. ARIMA and SARIMAX also provide competitive results, with slightly higher error metrics compared to Prophet.
3. LSTM, similar to its performance in quantity forecasting, exhibits higher error metrics, indicating its limitations in accurately predicting commodity prices.

Based on the evaluation, Prophet appears to be the most suitable model for both quantity and price forecasting, offering the most accurate predictions among the evaluated models.

==========

.

.

.

.

.

.

.