

```
In [1]: import pandas as pd
from openpyxl.workbook import Workbook

#reading excel file
df = pd.read_excel('C:/Users/svasudev/OneDrive - IGMFinancial/Documents/Predictive Analyti
engine='openpyxl', sheet_name=0, header=0, index_col=False, keep_default_na=
```

```
In [2]: #Top 5 rows of the Excel file
df.head()
```

Out[2]:

	Date	Ottawa	Toronto West/Ouest	Toronto East/Est	Windsor	London	Peterborough	St. Catharine's	Sudbury	Sault Saint Marie	Thunder Bay	N
0	1990-01-03	55.9	49.1	48.7	45.2	50.1	0.0	0.0	56.4	54.8	56.6	
1	1990-01-10	55.9	47.7	46.8	49.7	47.6	0.0	0.0	56.4	54.9	56.8	
2	1990-01-17	55.9	53.2	53.2	49.6	53.7	0.0	0.0	55.8	54.9	56.8	
3	1990-01-24	55.9	53.2	53.5	49.0	52.1	0.0	0.0	55.7	54.9	56.8	
4	1990-01-31	55.9	51.9	52.6	48.6	49.1	0.0	0.0	55.6	54.8	56.8	

```
In [3]: #Datatype and Null Information about the columns in the Excel file
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9962 entries, 0 to 9961
Data columns (total 20 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Date                                       9962 non-null   datetime64[ns]
1   Ottawa                                    9962 non-null   float64
2   Toronto West/Ouest                       9962 non-null   float64
3   Toronto East/Est                         9962 non-null   float64
4   Windsor                                  9962 non-null   float64
5   London                                    9962 non-null   float64
6   Peterborough                             9962 non-null   float64
7   St. Catharine's                          9962 non-null   float64
8   Sudbury                                  9962 non-null   float64
9   Sault Saint Marie                       9962 non-null   float64
10  Thunder Bay                              9962 non-null   float64
11  North Bay                                9962 non-null   float64
12  Timmins                                  9962 non-null   float64
13  Kenora                                   9962 non-null   float64
14  Parry Sound                              9962 non-null   float64
15  Ontario Average/Moyenne provinciale      9962 non-null   float64
16  Southern Average/Moyenne du sud de l'Ontario 9962 non-null   float64
17  Northern Average/Moyenne du nord de l'Ontario 9962 non-null   float64
```

```
18 Fuel Type          9962 non-null    object
19 Type de carburant    9962 non-null    object
dtypes: datetime64[ns](1), float64(17), object(2)
memory usage: 1.5+ MB
```

```
In [4]: #Creating a new dataframe
df2 = pd.DataFrame()
```

```
In [5]: #Choosing rows from the original dataframe with Fuel Type "Regular Unleaded gasoline" and
df2 = df.loc[df['Fuel Type']=='Regular Unleaded Gasoline']
```

```
In [6]: #Datatype and Null Information about the columns in the new dataframe
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1675 entries, 0 to 1674
Data columns (total 20 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Date                                     1675 non-null   datetime64[ns]
1   Ottawa                                 1675 non-null   float64
2   Toronto West/Ouest                     1675 non-null   float64
3   Toronto East/Est                       1675 non-null   float64
4   Windsor                                1675 non-null   float64
5   London                                 1675 non-null   float64
6   Peterborough                           1675 non-null   float64
7   St. Catharine's                        1675 non-null   float64
8   Sudbury                                1675 non-null   float64
9   Sault Saint Marie                      1675 non-null   float64
10  Thunder Bay                             1675 non-null   float64
11  North Bay                               1675 non-null   float64
12  Timmins                                 1675 non-null   float64
13  Kenora                                  1675 non-null   float64
14  Parry Sound                             1675 non-null   float64
15  Ontario Average/Moyenne provinciale    1675 non-null   float64
16  Southern Average/Moyenne du sud de l'Ontario 1675 non-null   float64
17  Northern Average/Moyenne du nord de l'Ontario 1675 non-null   float64
18  Fuel Type                               1675 non-null   object
19  Type de carburant                       1675 non-null   object
dtypes: datetime64[ns](1), float64(17), object(2)
memory usage: 274.8+ KB
```

```
In [7]: df2.head()
```

Out[7]:

	Date	Ottawa	Toronto West/Ouest	Toronto East/Est	Windsor	London	Peterborough	St. Catharine's	Sudbury	Sault Saint Marie	Thunder Bay	N
0	1990-01-03	55.9	49.1	48.7	45.2	50.1	0.0	0.0	56.4	54.8	56.6	
1	1990-01-10	55.9	47.7	46.8	49.7	47.6	0.0	0.0	56.4	54.9	56.8	
2	1990-01-17	55.9	53.2	53.2	49.6	53.7	0.0	0.0	55.8	54.9	56.8	

	Date	Ottawa	Toronto West/Ouest	Toronto East/Est	Windsor	London	Peterborough	St. Catharine's	Sudbury	Sault Saint Marie	Thunder Bay	N
3	1990-01-24	55.9	53.2	53.5	49.0	52.1	0.0	0.0	55.7	54.9	56.8	
4	1990-01-31	55.9	51.9	52.6	48.6	49.1	0.0	0.0	55.6	54.8	56.8	

```
In [8]: #Creating a new dataframe
df3 = pd.DataFrame()
```

```
In [9]: #Analyzing 'Regular Unleaded Gasoline' price for Windsor Region only and pasting it into df3
df3.insert(0, "Date", df2['Date'], True)
df3.insert(1, "Windsor", df2['Windsor'], True)
```

```
In [10]: df3.head()
```

```
Out[10]:
```

	Date	Windsor
0	1990-01-03	45.2
1	1990-01-10	49.7
2	1990-01-17	49.6
3	1990-01-24	49.0
4	1990-01-31	48.6

```
In [11]: #Date column only had weekly values. Sometimes the week started on Wednesday and sometimes
#this inconsistency, converting the date column into daily values and assigning the previous
#daily values.
df3.set_index('Date', inplace=True)
df3 = df3.resample('D').ffill().reset_index()
```

```
In [12]: #Last 10 values from the dataset
df3.tail(10)
```

```
Out[12]:
```

	Date	Windsor
11707	2022-01-22	145.9
11708	2022-01-23	145.9
11709	2022-01-24	147.3
11710	2022-01-25	147.3
11711	2022-01-26	147.3
11712	2022-01-27	147.3
11713	2022-01-28	147.3
11714	2022-01-29	147.3

	Date	Windsor
11715	2022-01-30	147.3
11716	2022-01-31	152.1

```
In [13]: #Checking for null values
df3.isnull().values.any()
```

```
Out[13]: False
```

```
In [14]: #Splitting the "Date" column into year, month and week to explore trends
df3['Year']=df3['Date'].dt.year
df3['Month']=df3['Date'].dt.month
df3['Week']=df3['Date'].dt.isocalendar().week
```

```
In [15]: df3.head(10)
```

```
Out[15]:
```

	Date	Windsor	Year	Month	Week
0	1990-01-03	45.2	1990	1	1
1	1990-01-04	45.2	1990	1	1
2	1990-01-05	45.2	1990	1	1
3	1990-01-06	45.2	1990	1	1
4	1990-01-07	45.2	1990	1	1
5	1990-01-08	45.2	1990	1	2
6	1990-01-09	45.2	1990	1	2
7	1990-01-10	49.7	1990	1	2
8	1990-01-11	49.7	1990	1	2
9	1990-01-12	49.7	1990	1	2

```
In [16]: #Splitting the dataset in Train and Test
#Train from Year 1990 to Year 2019
#Test from Year 2020

train = df3[(df3['Date'] > '1990-01-01') & (df3['Date'] <= '2019-12-31')]
test = df3[df3['Date'] >= '2020-01-01']
```

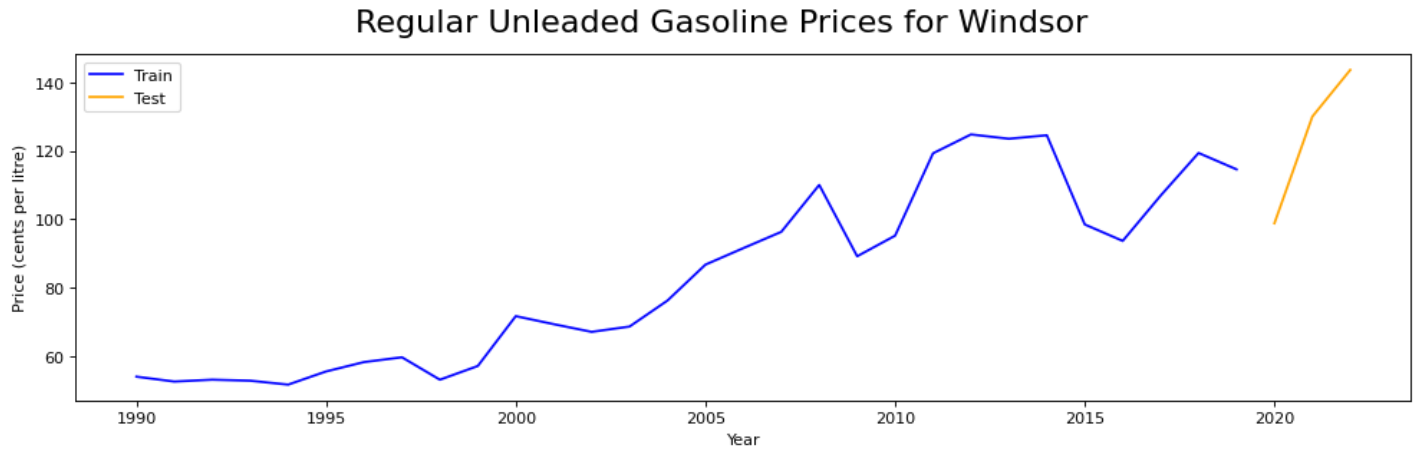
## Yearly Price Visualization on Train and Test Dataset

```
In [17]: import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

yearly_train_Price = train.groupby(['Year'])['Windsor'].mean()
yearly_test_Price = test.groupby(['Year'])['Windsor'].mean()

figure(figsize=(15, 4), dpi=80)
plt.plot(yearly_train_Price, label='Train',c='blue')
plt.plot(yearly_test_Price, label='Test',c='orange')
plt.legend(loc='best')
```

```
plt.suptitle('Regular Unleaded Gasoline Prices for Windsor', fontsize=20)
plt.xlabel('Year')
plt.ylabel('Price (cents per litre)')
plt.show()
```



## DataPrep for Time Series

```
In [18]: train.index = pd.DatetimeIndex(train['Date'])
          #Changing the frequency of the index to Daily
          train.index = train.asfreq('d').index

          test.index = pd.DatetimeIndex(test['Date'])
          #Changing the frequency of the index to Daily
          test.index = test.asfreq('d').index
```

## Train and Time Series Dataset

```
In [19]: train_time_series = pd.DataFrame()
          train_time_series.index = train.index
          train_time_series.insert(0,"Windsor Gas Price Train",train['Windsor'],True)

          test_time_series = pd.DataFrame()
          test_time_series.index = test.index
          test_time_series.insert(0,"Windsor Gas Price Test",test['Windsor'],True)

          print(train_time_series.tail())
          print(test_time_series.tail())
```

### Windsor Gas Price Train

Date	Price
2019-12-27	117.7
2019-12-28	117.7
2019-12-29	117.7
2019-12-30	118.4
2019-12-31	118.4

### Windsor Gas Price Test

Date	Price
2022-01-27	147.3
2022-01-28	147.3
2022-01-29	147.3
2022-01-30	147.3
2022-01-31	152.1

## ARIMA Model

To predict time series with ARIMA, we need to set the values of three parameters (p,d,q):

p: The order of the auto-regressive (AR) model (i.e., the number of lag observations)

d: The degree of differencing.

q: The order of the moving average (MA) model.

## Checking if data is stationary - We can see that it is not based on the P-value - Augmented Dickey Fuller Test

In [20]:

```
from statsmodels.tsa.stattools import adfuller
results = adfuller(train_time_series['Windsor Gas Price Train'])
print('ADF Statistic: ', results[0])
print('p-value: ', results[1])
print('Critical Values', results[4])
```

ADF Statistic: -2.2159638395298806

p-value: 0.20050260538288545

Critical Values {'1%': -3.430948594020318, '5%': -2.861804545421058, '10%': -2.566910812397698}

## Taking First difference - P value is < 0.05. We can stop at the First Difference; d = 1

In [21]:

```
train_time_series_stationary1 = train_time_series.diff().dropna()
results1 = adfuller(train_time_series_stationary1['Windsor Gas Price Train'])
print('ADF Statistic: ', results1[0])
print('p-value: ', results1[1])
print('Critical Values', results1[4])
```

ADF Statistic: -20.581634517019687

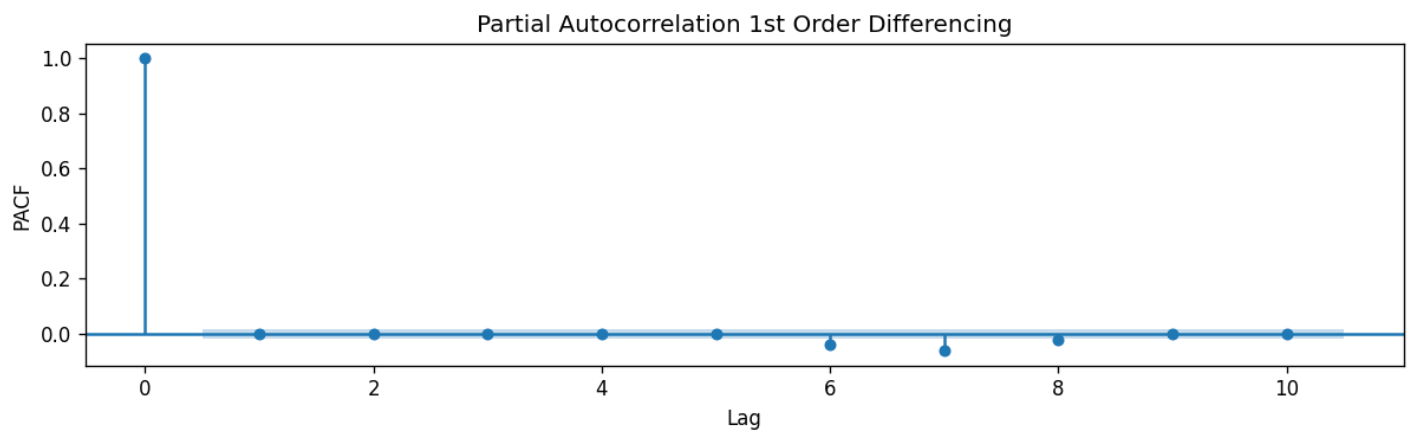
p-value: 0.0

Critical Values {'1%': -3.430948594020318, '5%': -2.861804545421058, '10%': -2.566910812397698}

## The Order of Autoregressive Term p; p = 0

In [22]:

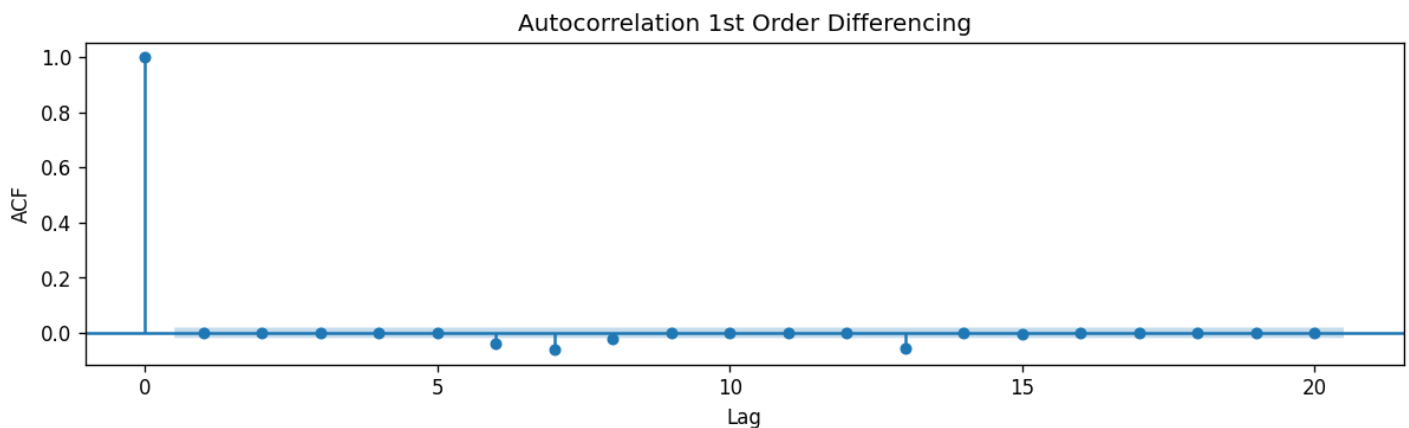
```
plt.rcParams.update({'figure.figsize':(12,3), 'figure.dpi':120})
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(train_time_series_stationary1, lags=10, title="Partial Autocorrelation 1st Order")
plt.xlabel('Lag')
plt.ylabel('PACF')
plt.show()
```



The order of the Moving Average term  $q$ ;  $q = 0$

In [23]:

```
plt.rcParams.update({'figure.figsize':(12,3), 'figure.dpi':120})
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(train_time_series_stationary1, lags=20, title="Autocorrelation 1st Order Differencing")
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.show()
```



Check the above  $p, d, q$  parameters with Auto Arima - Best Model ARIMA (0,1,0)

In [24]:

```
import pmdarima as pm
from pmdarima.model_selection import train_test_split
import numpy as np

modell = pm.auto_arima(train_time_series, trace=True, error_action='ignore', suppress_warnings=True)
modell.fit(train_time_series)

forecast1 = modell.predict(n_periods=len(test_time_series))
forecast1 = pd.DataFrame(forecast1, index = test_time_series.index, columns=['Prediction'])
```

Performing stepwise search to minimize aic

```
ARIMA(2,1,2) (0,0,0) [0] intercept : AIC=37173.435, Time=1.42 sec
ARIMA(0,1,0) (0,0,0) [0] intercept : AIC=37165.435, Time=0.22 sec
ARIMA(1,1,0) (0,0,0) [0] intercept : AIC=37167.435, Time=0.61 sec
ARIMA(0,1,1) (0,0,0) [0] intercept : AIC=37167.435, Time=0.88 sec
ARIMA(0,1,0) (0,0,0) [0]           : AIC=37163.716, Time=0.15 sec
ARIMA(1,1,1) (0,0,0) [0] intercept : AIC=37169.435, Time=1.03 sec
```

Best model: ARIMA(0,1,0) (0,0,0) [0]

Total fit time: 4.360 seconds

## Model Summary

In [25]:

```
import statsmodels.api as sm
model = sm.tsa.arima.ARIMA(train_time_series, order=(0,1,0))
model_result = model.fit()
print(model_result.summary())
```

### SARIMAX Results

```
=====
Dep. Variable:      Windsor Gas Price Train      No. Observations:      10955
Model:              ARIMA(0, 1, 0)              Log Likelihood        -18580.858
Date:              Sat, 05 Mar 2022              AIC                   37163.716
Time:              13:34:56                      BIC                   37171.018
Sample:            01-03-1990                    HQIC                  37166.176
                  - 12-31-2019
Covariance Type:    opg
=====
              coef      std err          z      P>|z|      [0.025      0.975]
-----
sigma2          1.7414      0.004     493.692      0.000        1.734        1.748
=====
Ljung-Box (L1) (Q):              0.00      Jarque-Bera (JB):              3463843.52
Prob(Q):                        1.00      Prob(JB):              0.00
Heteroskedasticity (H):          5.42      Skew:              2.61
Prob(H) (two-sided):            0.00      Kurtosis:              89.96
=====
```

Warnings:

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

## Model Prediction

In [26]:

```
import warnings
warnings.filterwarnings('ignore')
ARIMA_Predict = model_result.predict(start='1/1/2020', end='1/31/2022')
ARIMA_Predict_df = pd.DataFrame(ARIMA_Predict)
```

In [27]:

```
ARIMA_Predict_df.tail()
```

Out[27]:

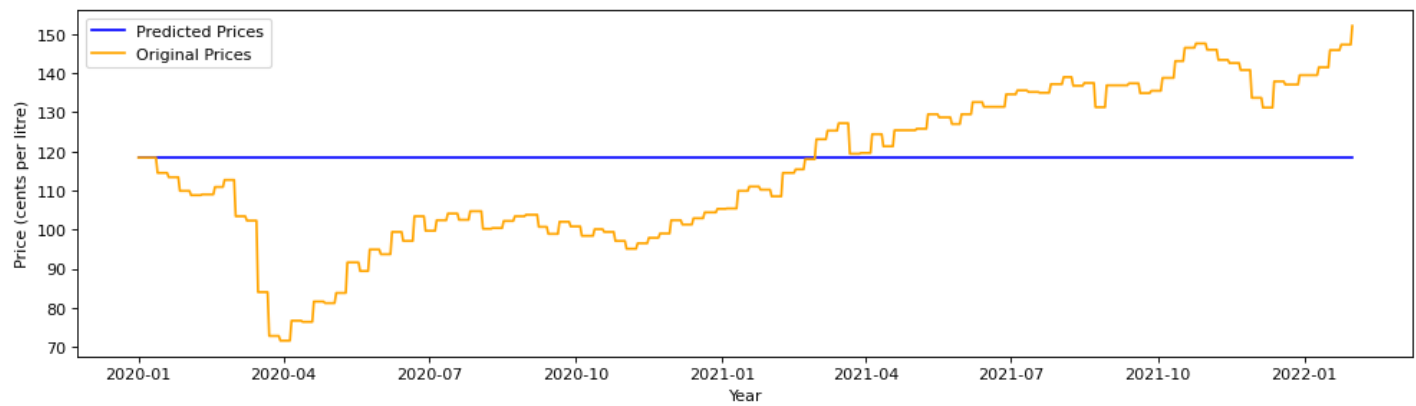
	predicted_mean
<b>2022-01-27</b>	118.4
<b>2022-01-28</b>	118.4
<b>2022-01-29</b>	118.4
<b>2022-01-30</b>	118.4
<b>2022-01-31</b>	118.4

In [28]:

```
figure(figsize=(15, 4), dpi=80)
plt.plot(ARIMA_Predict_df, label='Predicted Prices',c='blue')
plt.plot(test_time_series, label='Original Prices',c='orange')
plt.legend(loc='best')
plt.suptitle('ARIMA Model Regular Unleaded Gasoline Prices Forecast Windsor', fontsize=20)
plt.xlabel('Year')
plt.ylabel('Price (cents per litre)')
plt.show()
```



# ARIMA Model Regular Unleaded Gasoline Prices Forecast Windsor



## Evaluation of the Model

### Mean Absolute Error (MAE) ARIMA

```
In [29]: from sklearn.metrics import mean_absolute_error
maeARIMA=mean_absolute_error(test_time_series['Windsor Gas Price Test'],ARIMA_Predict)
print('Mean Absolute Error ARIMA = {}'.format(round(maeARIMA, 2)))
```

Mean Absolute Error ARIMA = 17.06

### Mean squared error (MSE) ARIMA

```
In [30]: from sklearn.metrics import mean_squared_error
mseARIMA=mean_squared_error(test_time_series['Windsor Gas Price Test'],ARIMA_Predict)
print('The Mean Squared Error ARIMA = {}'.format(round(mseARIMA, 2)))
```

The Mean Squared Error ARIMA = 380.76

### Root mean squared error (RMSE) ARIMA

```
In [31]: from numpy import sqrt
rmseARIMA = sqrt(mseARIMA)
print('The Root Mean Squared Error ARIMA = {}'.format(round(rmseARIMA, 2)))
```

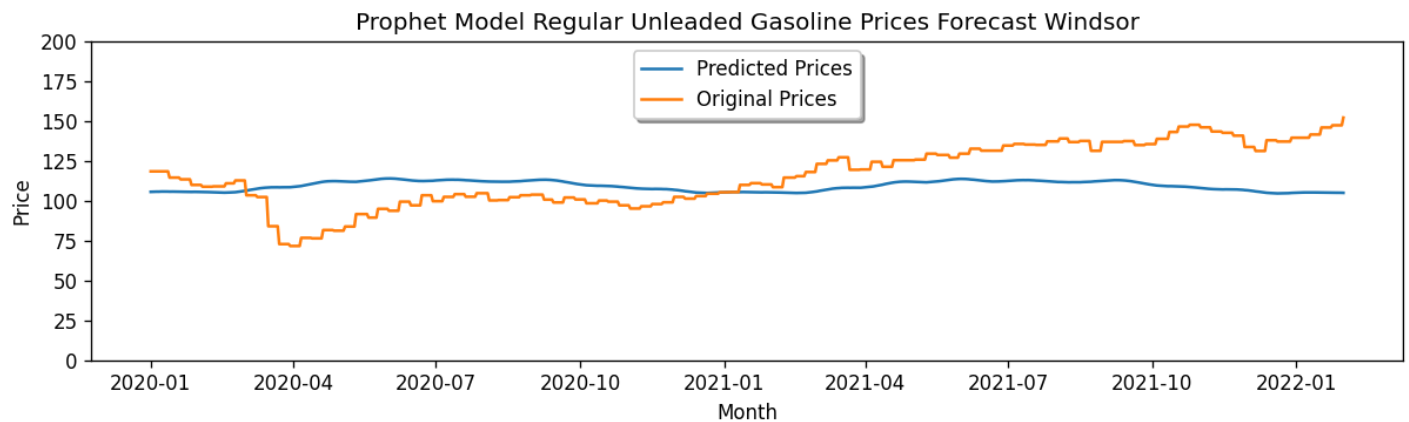
The Root Mean Squared Error ARIMA = 19.51

## Prophet Model

```
In [32]: from fbprophet import Prophet
d={'ds':train['Date'],'y':train['Windsor']}
df_pred=pd.DataFrame(data=d)
model_prophet = Prophet(daily_seasonality=False)
model_prophet_result = model_prophet.fit(df_pred)
```

```
In [33]: future = model_prophet.make_future_dataframe(periods=765)
forecast = model_prophet.predict(future)
forecast = forecast[(forecast['ds'] >= '2020-01-01') & (forecast['ds'] <= '2022-01-31')]
```

```
In [34]: fig, ax = plt.subplots()
ax.plot(forecast['ds'], forecast['yhat'], label='Predicted Prices')
ax.plot(test['Date'], test['Windsor'], label='Original Prices')
plt.ylim([0,200])
legend = ax.legend(loc='upper center', shadow=True)
plt.title('Prophet Model Regular Unleaded Gasoline Prices Forecast Windsor')
plt.xlabel('Month')
plt.ylabel('Price')
plt.show()
```



## Mean Absolute Error (MAE) Prophet

```
In [35]: maeProphet=mean_absolute_error(test['Windsor'],forecast['yhat'])
print('Mean Absolute Error Prophet = {}'.format(round(maeProphet, 2)))
```

Mean Absolute Error Prophet = 17.76

## Mean squared error (MSE) Prophet

```
In [36]: mseProphet = mean_squared_error(test['Windsor'],forecast['yhat'])
print('The Mean Squared Error Prophet = {}'.format(round(mseProphet, 2)))
```

The Mean Squared Error Prophet = 435.28

## Root mean squared error (RMSE) Prophet

```
In [37]: rmseProphet = sqrt(mseProphet)
print('The Root Mean Squared Error Prophet = {}'.format(round(rmseProphet, 2)))
```

The Root Mean Squared Error Prophet = 20.86