INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT AKRUDI, PUNE

DOCUMENTATION ON

"Gold Price Prediction"

PG-eDBDA May-2021

Submited by:

Group No.:28

Sravani Alla (1352)

Deepshri Patil (1336)

Mr. Prashant Karhale — **Centre Coordinator**

Mr.Akshay Tilekar — **Project Guide**

INDEX

Table of Contents

List Of Figures

1. Introduction

Historically, gold had been used as a form of currency in various parts of the world including USA. In recent times also, gold has maintained its value and has been used as a means for assessing the financial strength of a country. Big investors have also been attracted to this precious metal and invested huge amounts in it. Recently, emerging world economies, such as China, Russia, and India have been big buyers of gold, whereas USA, South Africa, and Australia are among the big seller of this commodity. Chinese and Indian traditional events also affect the price of the gold. In that time more money is poured for purchase of this commodity. Small investors also find this commodity for safe investment rather than alternate investment options, which bear in-built investment risks. Internal financial conditions of the aforementioned countries play a vital role for setting spot rates for gold. Governmental investments in gold are largely decided by their financial conditions, and interest rates, as they are indicators of the strength of their economy. When US interest rates become lower, more economic activity is witnessed in US, thus capital inflows in gold market are observed. Similarly, when interest rates lowered in China from 5.31(2010) to 4.35 (2016), it bought gold aggressively. Global investors, either countries or giant companies, tend to invest elsewhere if they foresee a significant decline in gold prices. In such a scenario, some investors turn to some other form of investment, such as US Bonds, or stock exchange. Therefore, various phenomena are interconnected with gold rates and affect the price also.

The spot price is the current market price at which commodity is purchased or sold for immediate payment and delivery. It is differentiated from the futures price, which is the price at which the two parties agree to transact on future date. Gold spot rates are decided twice a day based on supply and demand in gold market. Fractional change in gold price may result in huge profit or loss for these investors as well as government banks. Forecasting rise and decline in the daily gold rates, can help investors to decide when to buy (or sell) the commodity. Various studies have been conducted by researchers to forecast gold rates, each of them insightful in their own right. We in this project forecast gold rates using various machine learning algorithms for forecasting and compare their results. We also identify which attributes influence the gold rates

the most, some of which were not even used before. In this we are using Time Series Model for forecasting the gold price.

1.1 Project Scope

The scope of this project is focused on GOLD price forecasting, trends and 1 year predictions. The goal of this project is to understand and apply time-series models like ARIMA, SARIMA in forecasting the price of gold .

1.2 Problem Statement

Many investors buy gold as a safe haven to protect themselves against a possible catastrophe, profit from these tremendous increases in the price of gold, diversify their portfolio and protect themselves against inflation.

With recent fears of an economic recession looming in the distance, investors are looking to recession-proof their portfolios. Gold has historically been touted as an asset that booms in a recession because unlike fiat currencies such as the Dollar, Yen and the Pound, it has inherent value as a commodity currency.

However, with the rapidly changing economic landscape, does this still hold true? Is gold a good commodity to buy now for later profits?

1.3 Data is taken from

SPDR® Gold Shares (NYSE Arca : GLD) is a cost-effective and convenient way to invest in gold without buying the real gold. The historical prices of SPDR® Gold Shares (NYSE Arca : GLD) was downloaded from Yahoo. Data spans from the inception of this share from 1/2/2001 to the date of download, 7/13/2021.

Flow Of Project:

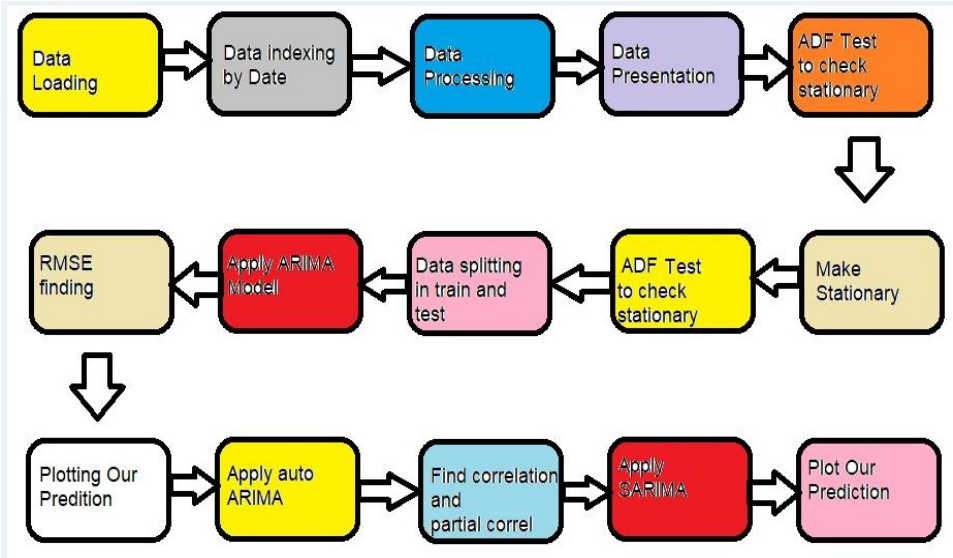


Fig.1

2. **Exploratory Data Analysis**

There are 5227 rows of data with 7 columns. With each data, there's the Opening price, High Price, Low Price, Close Price, Adjusted Close Price and Volume.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2001-01-02	268.399994	268.399994	268.399994	268.399994	268.399994	0.0
1	2001-01-03	268.000000	268.000000	268.000000	268.000000	268.000000	1.0
2	2001-01-04	267.299988	267.299988	267.299988	267.299988	267.299988	1.0
3	2001-01-05	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
4	2001-01-08	268.000000	268.000000	268.000000	268.000000	268.000000	0.0

Shape of our Data:

```
data.shape
```

```
(5227, 7)
```

2.1 Checking Missing Data:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5227 entries, 0 to 5226
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       5227 non-null   object 
 1   Open       5120 non-null   float64
 2   High       5120 non-null   float64
 3   Low        5120 non-null   float64
 4   Close      5120 non-null   float64
 5   Adj Close  5120 non-null   float64
 6   Volume     5120 non-null   float64
dtypes: float64(6), object(1)
memory usage: 286.0+ KB
```

There are some missing data in our dataset. However, if you look at the dataframe with the dates, we do not have prices for every date. That's because the stock market is closed on weekends and holidays. We need to fill in the missing days to make this data set as daily time series data.

these 2 missing dates are added to the dataframe →

	Date	Open	High	Low	Close	Adj Close	Volume
0	2001-01-02	268.399994	268.399994	268.399994	268.399994	268.399994	0.0
1	2001-01-03	268.000000	268.000000	268.000000	268.000000	268.000000	1.0
2	2001-01-04	267.299988	267.299988	267.299988	267.299988	267.299988	1.0
3	2001-01-05	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
4	2001-01-08	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
5	2001-01-09	267.500000	267.500000	267.500000	267.500000	267.500000	0.0
6	2001-01-10	264.700012	264.700012	264.700012	264.700012	264.700012	0.0
7	2001-01-11	264.000000	264.000000	264.000000	264.000000	264.000000	0.0
8	2001-01-12	263.899994	263.899994	263.899994	263.899994	263.899994	0.0
9	2001-01-15	NaN	NaN	NaN	NaN	NaN	NaN

2.2 Forward Filling

Forward Fill Missing Dates for days where this is no pricing information, we re-sample the Day and fill in the missing values from the previous day.

```
[ ]  data= data.resample("D").ffill().reset_index()
```

After this transformation, the dataframe now has 7500 rows of data and here's the plot of the Close Price for all 7500 rows.

GOLD Data After Forward Filling is: (7500, 7)

	Date	Open	High	Low	Close	Adj Close	Volume
0	2001-01-02	268.399994	268.399994	268.399994	268.399994	268.399994	0.0
1	2001-01-03	268.000000	268.000000	268.000000	268.000000	268.000000	1.0
2	2001-01-04	267.299988	267.299988	267.299988	267.299988	267.299988	1.0
3	2001-01-05	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
4	2001-01-06	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
5	2001-01-07	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
6	2001-01-08	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
7	2001-01-09	267.500000	267.500000	267.500000	267.500000	267.500000	0.0
8	2001-01-10	264.700012	264.700012	264.700012	264.700012	264.700012	0.0
9	2001-01-11	264.000000	264.000000	264.000000	264.000000	264.000000	0.0

```
[ ]  data[['Close']].isna().sum() #checking NAN values
```

```
Close    197
dtype: int64
```

```
[ ]  data= data.interpolate() #filling the NAN values
     print(data)
```

```
           Date        Open        High  ...       Close   Adj Close  Volume
0    2001-01-02  268.399994  268.399994  ...  268.399994  268.399994     0.0
1    2001-01-03  268.000000  268.000000  ...  268.000000  268.000000     1.0
2    2001-01-04  267.299988  267.299988  ...  267.299988  267.299988     1.0
3    2001-01-05  268.000000  268.000000  ...  268.000000  268.000000     0.0
4    2001-01-06  268.000000  268.000000  ...  268.000000  268.000000     0.0
...         ...         ...         ...  ...         ...         ...     ...
7495 2021-07-11  1803.599976  1810.099976  ...  1810.000000  1810.000000   218.0
7496 2021-07-12  1802.599976  1805.500000  ...  1805.500000  1805.500000   218.0
7497 2021-07-13  1808.099976  1812.000000  ...  1809.400024  1809.400024   147.0
7498 2021-07-14  1813.099976  1829.000000  ...  1824.300049  1824.300049   707.0
7499 2021-07-15  1831.599976  1833.000000  ...  1828.400024  1828.400024   707.0

[7500 rows x 7 columns]
```

```
[ ]  data.isnull().sum() #Checking null values present or not
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

```
[ ]  # Read in the gld prices file with no missing data
     # Convert the date column from string to date and make it the index
     dateparser = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
     data= pd.read_csv('/content/Dataset.csv', index_col='Date', parse_dates=['Date'], date_parser=dateparser)
     print("GOLD Prices after forward filling is: " + str(data.shape))
     data.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2001-01-02	268.399994	268.399994	268.399994	268.399994	268.399994	0.0
2001-01-03	268.000000	268.000000	268.000000	268.000000	268.000000	1.0
2001-01-04	267.299988	267.299988	267.299988	267.299988	267.299988	1.0
2001-01-05	268.000000	268.000000	268.000000	268.000000	268.000000	0.0
2001-01-06	268.000000	268.000000	268.000000	268.000000	268.000000	0.0

2.3 Checking Missing Dates

Check for missing dates

```
[ ]  data.index
```

```
DatetimeIndex(['2001-01-02', '2001-01-03', '2001-01-04', '2001-01-05',
               '2001-01-06', '2001-01-07', '2001-01-08', '2001-01-09',
               '2001-01-10', '2001-01-11',
               ...
               '2021-07-06', '2021-07-07', '2021-07-08', '2021-07-09',
               '2021-07-10', '2021-07-11', '2021-07-12', '2021-07-13',
               '2021-07-14', '2021-07-15'],
              dtype='datetime64[ns]', name='Date', length=7500, freq=None)
```

```
[ ]  period = 7500
     pd.date_range('2001-01-02', freq='D', periods=period)
```

```
DatetimeIndex(['2001-01-02', '2001-01-03', '2001-01-04', '2001-01-05',
               '2001-01-06', '2001-01-07', '2001-01-08', '2001-01-09',
               '2001-01-10', '2001-01-11',
               ...
               '2021-07-06', '2021-07-07', '2021-07-08', '2021-07-09',
               '2021-07-10', '2021-07-11', '2021-07-12', '2021-07-13',
               '2021-07-14', '2021-07-15'],
              dtype='datetime64[ns]', length=7500, freq='D')
```

```
[ ]  sns.set(style="darkgrid")
     plt.figure(figsize=(16,8))
     sns.lineplot(x=data.index, y='Close', data=data, linewidth=1.5).set_title('GLD Daily closing Price')
     plt.show()
```

3. Checking Data is Stationary

3.1 Dicky-Fuller Test:

The time series data has to be stationary before the modeling. We use augmented Dicky-Fuller test on the Close Price.

```
adfuller_result = adfuller(data['Close'])
print('ADF Statistic: ', adfuller_result[0])
print('p-value: ', adfuller_result[1])
```

```
ADF Statistic:  -0.7949924907096847
p-value:  0.8205885619637587
```

Based on the results above, the data is not stationary because the p-value is greater than 0.05.

4.Making Data Stationary

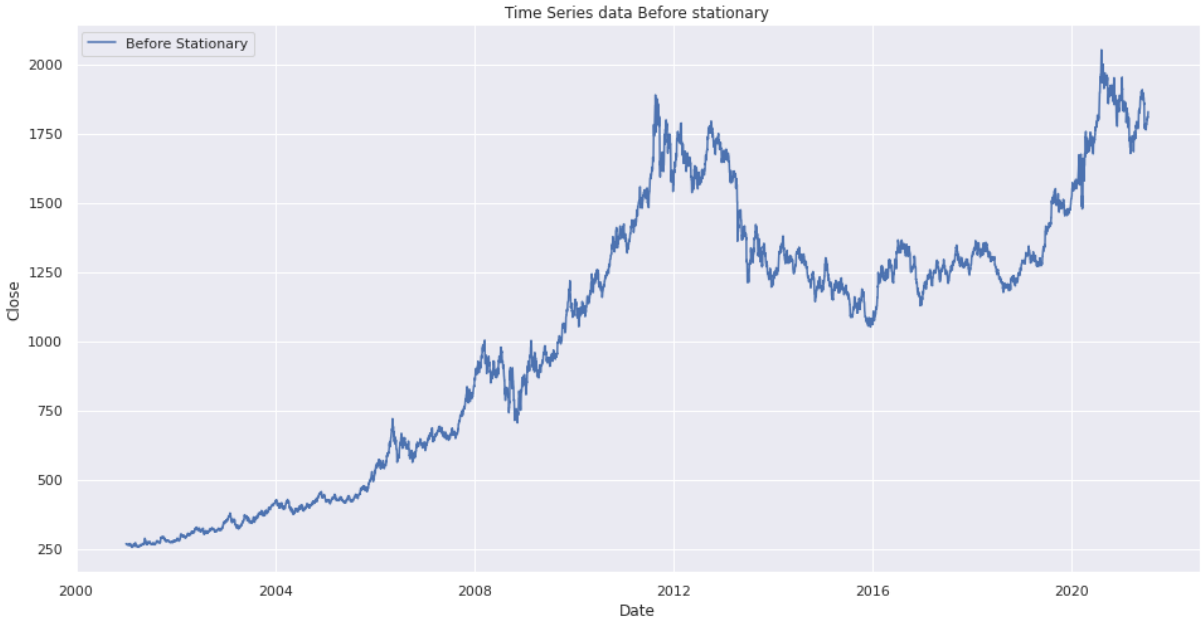
There are different ways to make time series data stationary. A few methods include the difference once method, difference twice and square root method. We are going to use all 3 and

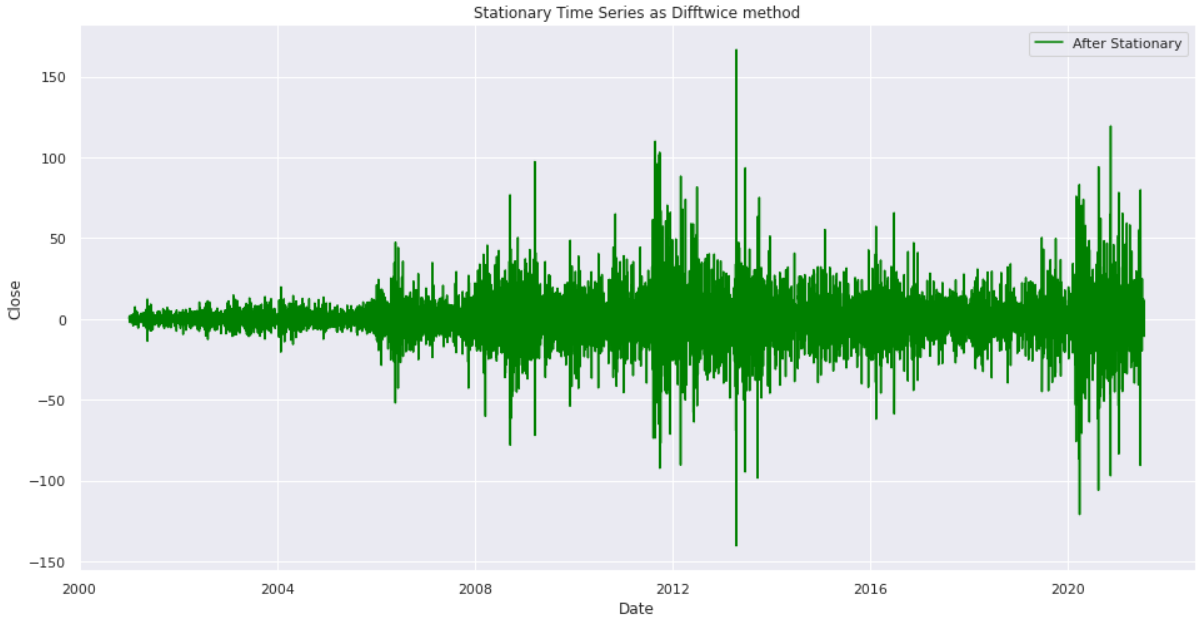
pick the best method. After we apply each of the methods, we apply the Dicky-Fuller test for checking data is stationary or not.

Method	ADF Statistic	P-Value
Difference Once	-18.232739426570856	2.3651174598377077e-30
Difference Twice	-1.1576360537067278	0.6915491435005863
Square Root	-24.524963587759327	0.0

Both Differencing once and The Square Root methods didn't produce a p-value less than 0.05. So we should eliminate it. Difference twice methods produced a p-value less than 0.05

Let’s compare the time series data before differencing twice and after.





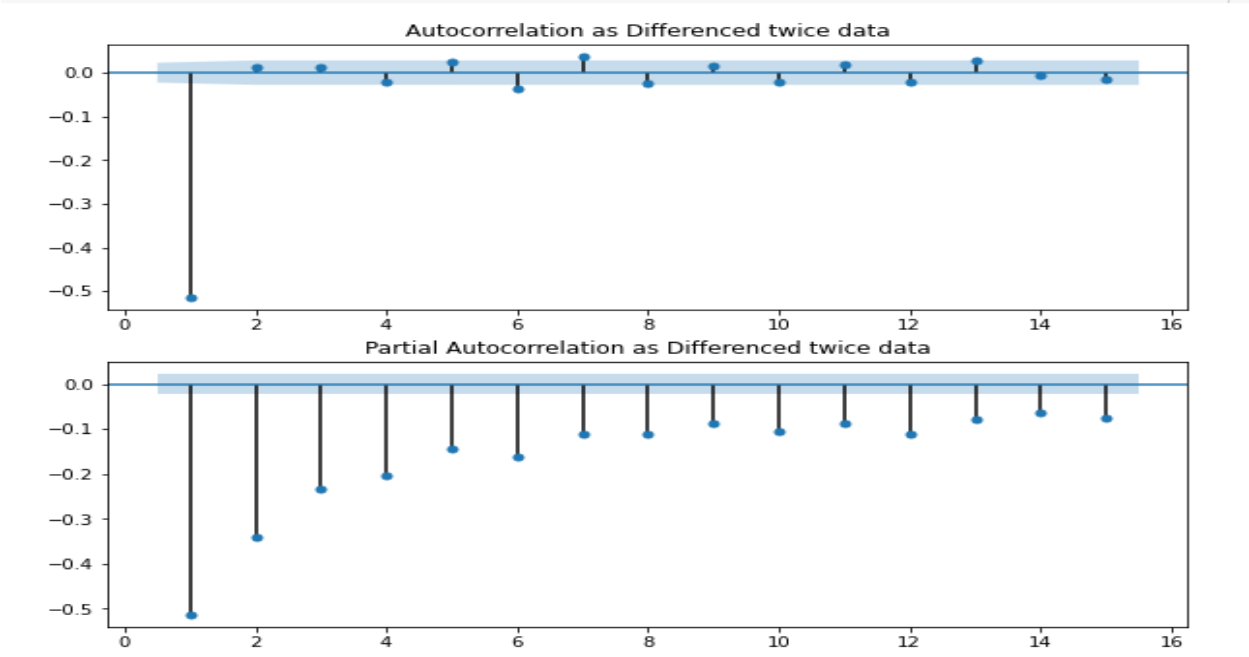
5.Choosing The Model Order For Time Series

5.1 Using ACF And PACF Methods:

Autocorrelation Function (ACF) is the correlation between the time series and the same time series lag by X number of steps.

Partial Autocorrelation (PACF) is the correlation between the time series and the lag version of itself after we subtract the effect of correlation at smaller lags. So, it's just the correlation associated with just that particular lag. By plotting both the ACF and PACF charts, it can help us select the right model order.

```
[ ] # Plot ACF and PACF with stationary data_difftwice

    fig_diff, (ax1, ax2) = plt.subplots(2,1, figsize=(10,8))
    # Plot ACF for data_diff
    plot_acf(data_difftwice['Close'], lags=15, zero=False, ax=ax1, title='Autocorrelation as Differenced twice data')

    # Plot PACF for data_difftwice
    plot_pacf(data_difftwice['Close'], lags=15, zero=False, ax=ax2, title='Partial Autocorrelation as Differenced twice data')
    plt.show()
```



We are making the judgements based on how the ACF and PACF graphs look. Sometimes it may not be as clear to make a conclusion. so we are going use pmdarima.

5.2 Using PMDARIMA

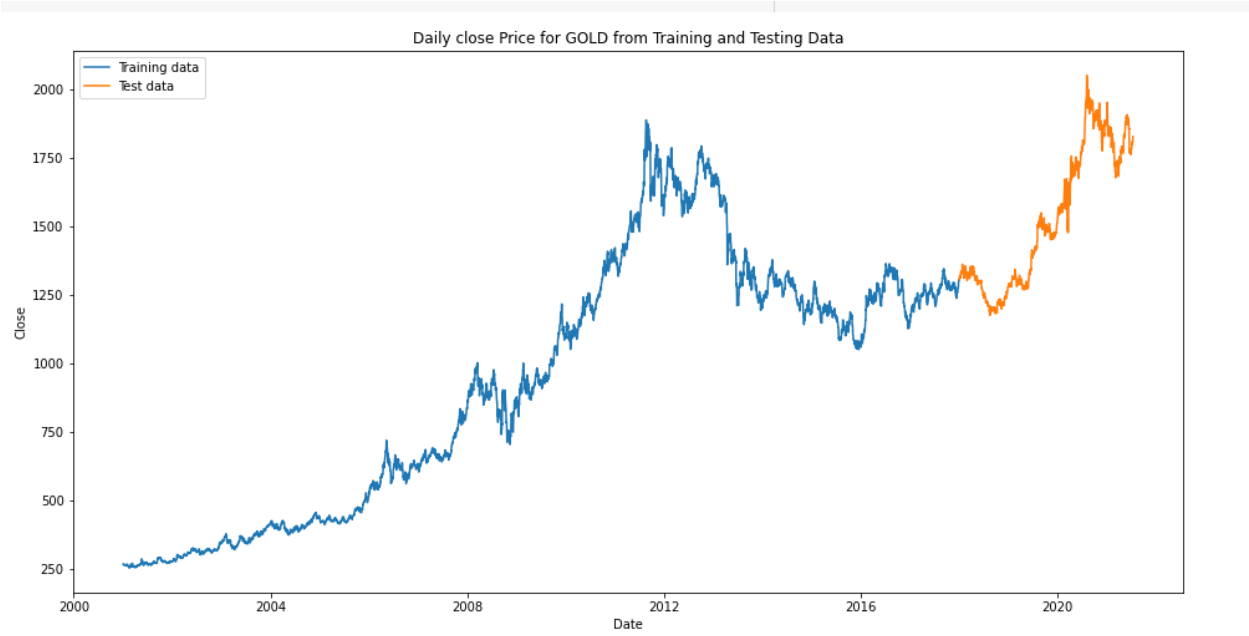
```
import pmdarima as pm
from pmdarima.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Load/split your data
y = data_difftwice['Close']
train, test = train_test_split(y, train_size=150)

# Fit your model
model = pm.auto_arima(train, seasonal=True, m=12,trace = True) #seasonal =False

# make your forecasts
forecasts = model.predict(test.shape[0])  # predict N steps into the future

# Visualize the forecasts (blue=train, green=forecasts)
x = np.arange(y.shape[0])
plt.plot(x[:150], train, c='blue')
plt.plot(x[150:], forecasts, c='green')
plt.show()
```

This is the same results as the ACF and PACF analysis where we determined it was a MA(1) model order.

6. Train And Test

As we are dealing with time series data, we cannot split the data randomly. The earlier data should always be the training data and the later data should be in the test set. There are 15 years of data. We are going to use the first 17 years (2001 - 2017) as training data and the last 4 years (2018 - 2021) as test data.

The train data set has 6208 rows of data and the test data set had 1292 rows of data. This is how they look when plotted together.



7. Using ARIMA Model

7.1 Method 1: ARIMA Model with No Seasonality & One-Step Ahead Forecast

Non-seasonal order

- p: autoregressive order
- d: differencing order

●q: moving average order

Seasonal order (leave it blank if there's no seasonality)

●P: seasonal autoregressive order

●D: seasonal differencing order

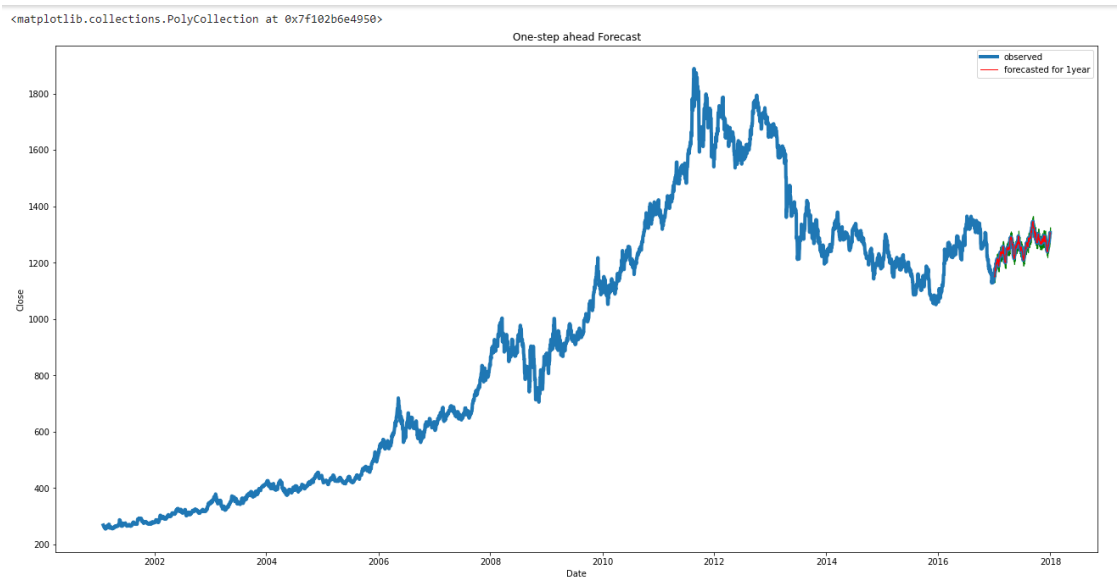
●Q: seasonal moving average order

●S: number of time steps per cycle

```
# Fit a model
model = SARIMAX(train_data['Close'], order=(1,0,0), trend= 'c')
results = model.fit()
# we want to start the prediction from one year back (365 days)
pred_1year_traindata = results.get_prediction(start=-365, dynamic=False)
# predicting mean for these 365 days
pred_mean_1year_traindata = pred_1year_traindata.predicted_mean

# Get confidence intervals of predicted
confidence_intervals = pred_1year_traindata.conf_int()

# Select lower and upper confidence limits
lower_limits = confidence_intervals.loc[:,'lower Close']
upper_limits = confidence_intervals.loc[:,'upper Close']
```

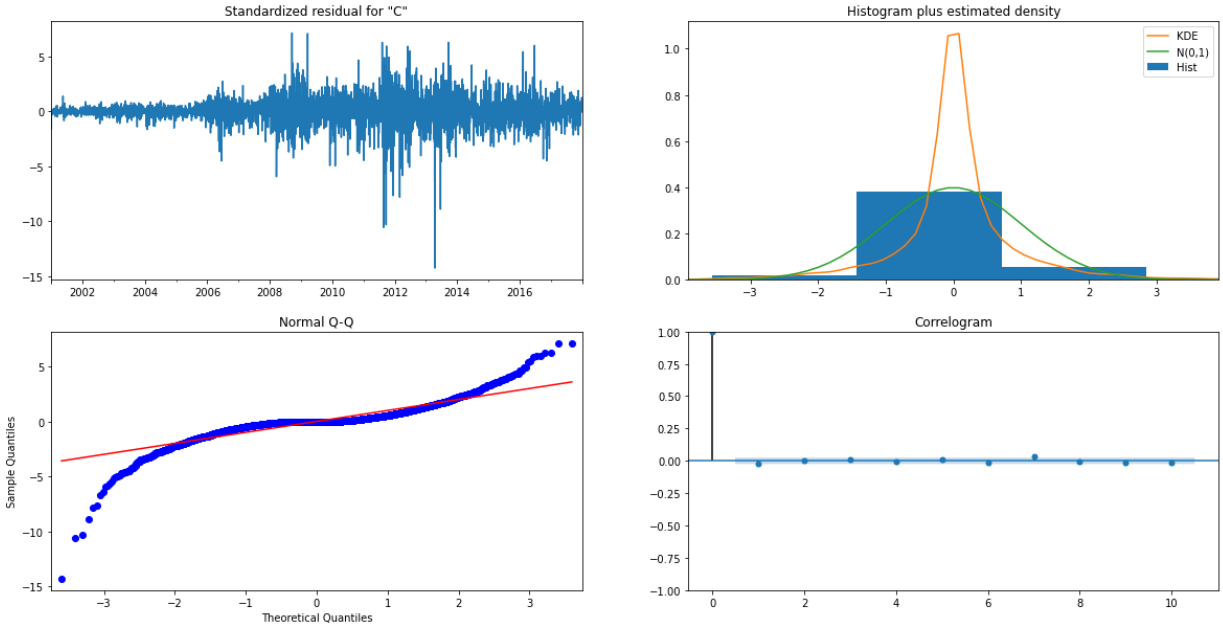
Based on our previous findings, difference order = 2 and the moving average order is 1. We are assuming there's no seasonality here, that's why P,D,Q are not provided in our model.

We created a fit based on the model above and used it to predict the close price of gold for the last 365 days of the training data. We then compared the prediction against the real prices for the last 365 days if the training set.



Overall, the above forecasts aligns very well with the true values for the last 365 days of the training data and it falls within the confidence intervals. Let's look at the model diagnostics results next.

7.1.1 Model Diagnostics :



We have observed from above graph as follows:

●Standardized Residual Plot - The graph doesn't seem to show a trend. That's what we want.

●Histogram Plus Estimated Density - This shows the distribution of the residuals. The green line shows a normal distribution and the orange line needs to be as close to the green line.

●Normal Q-Q - This shows how the distribution of the residuals compared to a normal distribution. Most of the residuals are on the line except the ends.

●Correlogram - ACF plot of the residuals. 95% of the data where lag > 0 should not be significant. That means, they need to be within the blue shaded area.

Checking the accuracy of our model as given below:

```
print('MAE: {}'.format(mean_absolute_error(real_values, pred_mean_1year_traindata)))
print('MSE: {}'.format(mean_squared_error(real_values, pred_mean_1year_traindata)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(real_values, pred_mean_1year_traindata))))
```

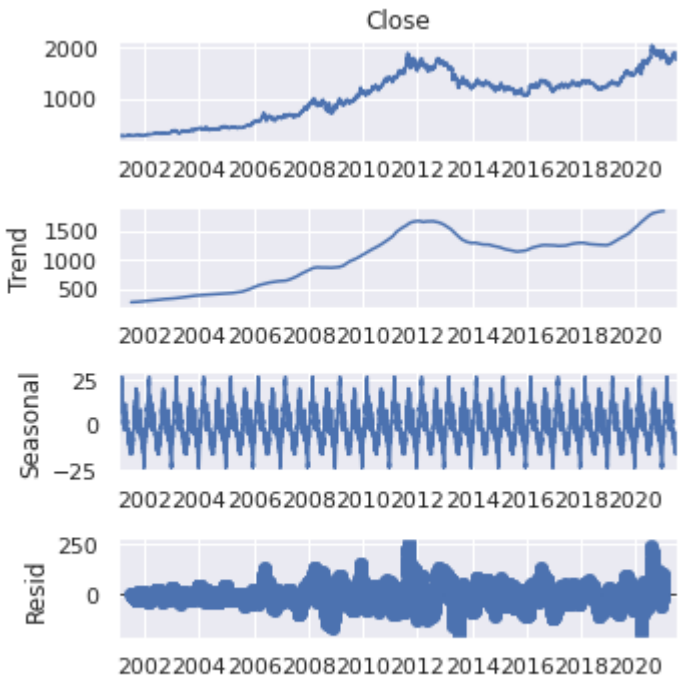
```
MAE: 4.430306121001212
MSE: 45.43995198138826
RMSE: 6.740916256814667
```

7.2 Method 2: Auto ARIMA Model with Seasonality & One-Step Ahead Forecast.

The previous method doesn't take into consideration any seasonality. Let's use seasonal_decompose to check for any seasonality.

```
# for checking seasonal time series as seasonal_decompose
from statsmodels.tsa.seasonal import seasonal_decompose
pd.plotting.register_matplotlib_converters()
decomp_results = seasonal_decompose(data['Close'], model='additive', freq=365)
decomp_results.plot()
plt.show()
```

The results is:



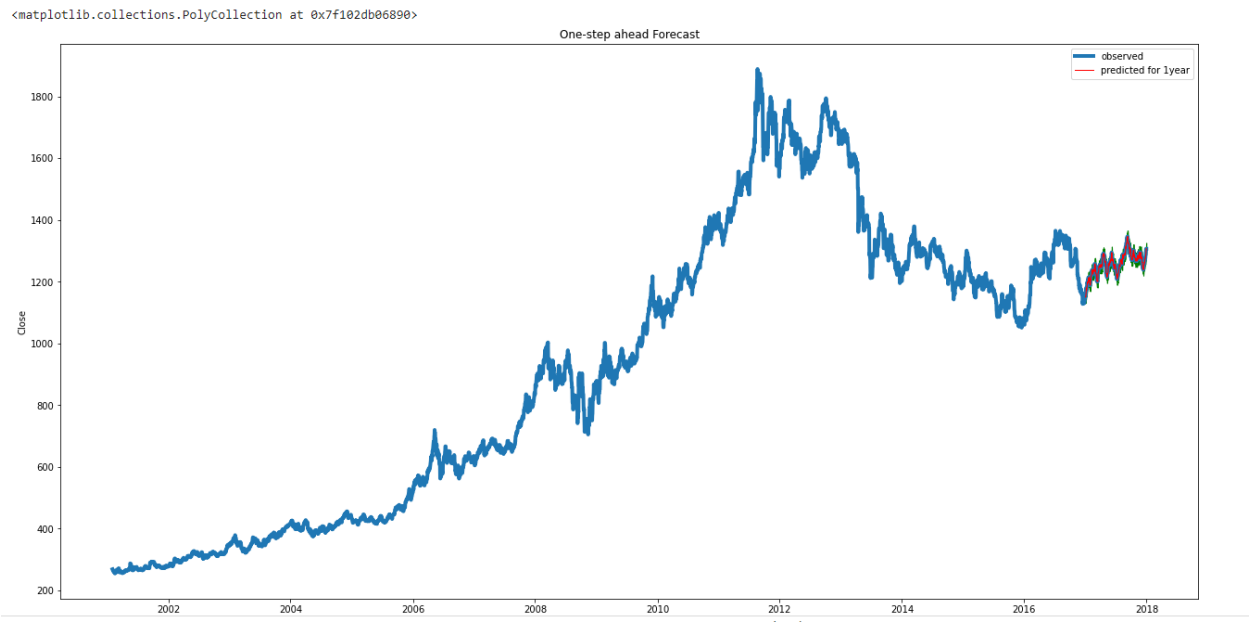
The plots above shows that the trend in prices of gold is not consistent but there is some obvious seasonality. Instead of guessing, we are going to use Auto Arima to perform an automatic grid search to discover the optimal order for an ARIMA model. It will also highlight if there's any seasonality in the data.

For Auto Arima to work, we need to create a univariate dataset. We need to drop all other columns other than Close Price. This is the final result of the dataset.

```
results = auto_arima(arima_data,
                     seasonal=True,
                     start_p = 1,
                     start_q = 1,
                     start_P=1,
                     start_Q=1,
                     max_P=3,
                     max_Q=3,
                     m=7,
                     information_criterion='aic',
                     trace=True,
                     error_action='ignore',
                     stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(1,0,1)[7] intercept   : AIC=56541.281, Time=14.48 sec
 ARIMA(0,1,0)(0,0,0)[7] intercept   : AIC=56537.512, Time=0.16 sec
 ARIMA(1,1,0)(1,0,0)[7] intercept   : AIC=56537.700, Time=2.02 sec
 ARIMA(0,1,1)(0,0,1)[7] intercept   : AIC=56537.756, Time=1.92 sec
 ARIMA(0,1,0)(0,0,0)[7]             : AIC=56538.460, Time=0.12 sec
 ARIMA(0,1,0)(1,0,0)[7] intercept   : AIC=56538.392, Time=1.15 sec
 ARIMA(0,1,0)(0,0,1)[7] intercept   : AIC=56538.402, Time=1.26 sec
 ARIMA(0,1,0)(1,0,1)[7] intercept   : AIC=56540.679, Time=3.68 sec
 ARIMA(1,1,0)(0,0,0)[7] intercept   : AIC=56536.647, Time=0.37 sec
 ARIMA(1,1,0)(0,0,1)[7] intercept   : AIC=56537.710, Time=1.48 sec
 ARIMA(1,1,0)(1,0,1)[7] intercept   : AIC=56539.980, Time=5.10 sec
 ARIMA(2,1,0)(0,0,0)[7] intercept   : AIC=56538.200, Time=0.72 sec
 ARIMA(1,1,1)(0,0,0)[7] intercept   : AIC=56538.456, Time=1.88 sec
 ARIMA(0,1,1)(0,0,0)[7] intercept   : AIC=56536.693, Time=1.19 sec
 ARIMA(2,1,1)(0,0,0)[7] intercept   : AIC=56540.189, Time=1.56 sec
 ARIMA(1,1,0)(0,0,0)[7]             : AIC=56537.712, Time=0.18 sec

Best model:  ARIMA(1,1,0)(0,0,0)[7] intercept
Total fit time: 37.342 seconds
```

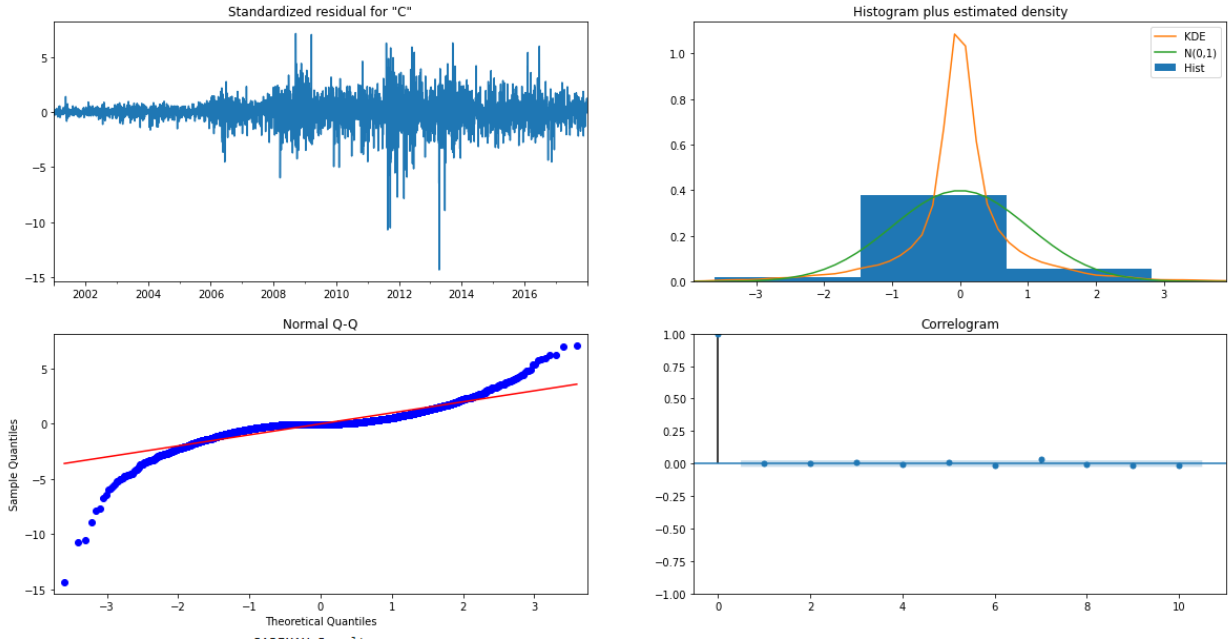
We then compared the prediction against the real prices for the last 365 days if the training set just like model 1.



Just like method 1, the forecast aligns very well with the true values for the last 365 days of the training data and it falls within the confidence intervals. Let's look at the model diagnostics results next.

7.2.1 Model Diagnostics Results

```
# Create the 4 diagnostic plots
auto_arima_results.plot_diagnostics()
plt.show()

# Print the diagnostic summary results
print(auto_arima_results.summary())

# Calculate MAE, MSE, RMSE
print('MAE: {}'.format(mean_absolute_error(real_values, auto_arima_pred_mean_1year_traindata)))
print('MSE: {}'.format(mean_squared_error(real_values, auto_arima_pred_mean_1year_traindata)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(real_values, auto_arima_pred_mean_1year_traindata))))
```




```
MAE: 4.4230364594183404
MSE: 45.33604936619089
RMSE: 6.733204984714997
```

There no difference in the results between Model1 and model 2 methods. MAE for Method 1 is slightly higher than Method 2. We are going to use the Auto Arima method for predicting the future data.

8. Using Auto ARIMA Model Forecast Closing price And Compare with Test Data.

Based on previous findings, we are going to use the fitted model using auto_arima to forecast the closing prices from 1/1/2018 - 07/15/2021 and compare the forecasted results against the test data set.

In This We Have Observed That :

By looking at the plot, the forecasted data showed an upward trend which is aligned with the test data. It correctly predicted that Gold Prices will go up from 2018 - 2021. It is also within the confidence interval. However, the interval is large. This shows that it’s hard to predict the prices of gold day-to-day but it's able to predict a general trend over time.

9.Predicting The Price of Gold ETFs For The Next Year

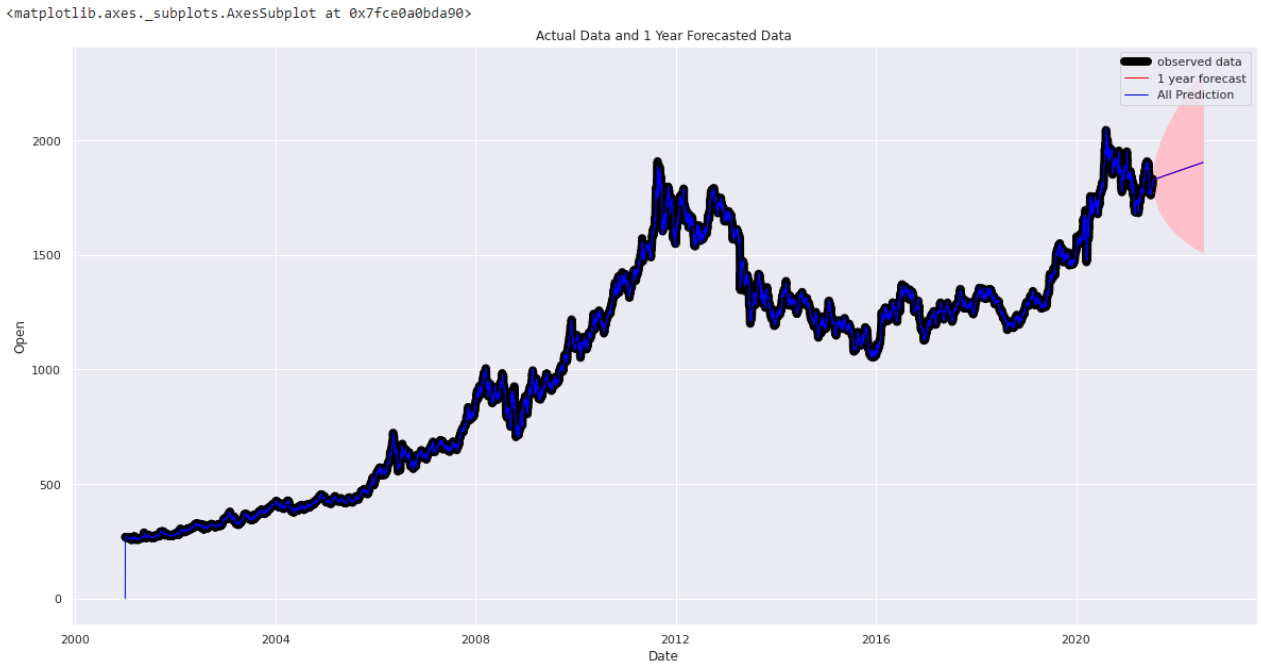
Auto Arima Method

Having validated our forecast results with our test data, we are going to perform an out of sample forecasting using the auto_arima method.. We have data of gold prices up till 07/15/2021. We are going to forecast the price of gold for the next year from 7/16/2019 - 07/16/2021.

Based on previous best fit order model recommended by auto_arima, this model have these parameters:

```
# Create a model as auto arima
auto_arima_model = SARIMAX(auto_arima_train_data,
                           seasonal=True,
                           order=(1,1,0),
                           seasonal_order=(0,0,0,7),
                           trend='c')
```

We use the above model to forecast next year out and also used the model to predict the price for the entire period from 01/02/2001 - 07/15/2022.



The blue line is the prediction results from 01/02/2001 - 07/15/2021. You can see that predicted prices are very well aligned to the actual prices as shown in the black line.

Conclusion

Based on the plots, Auto ARIMA shows a better fit between actual data and predicted data.

The Auto ARIMA model shows an upward trend in gold prices for the next year.

In conclusion, we can say that GLD ETF is a safe asset to buy.

References

- https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3
- https://www.researchgate.net/publication/286452129_Gold_price_Forecasting_In_India_using_ARIMA_modelling
- https://ieeexplore.ieee.org/document/9332471
- https://www.ijeat.org/wp-content/uploads/papers/v10i5/D25370410421.pdf
- http://thesai.org/Downloads/Volume8No12/Paper_13-Predicting_Future_Gold_Rates.pdf
- https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/15/02-Fitting-the-Future.html