Gold Price Forecasting

- The trend of gold price is irregular over all years. This gold price dataset is taken from world gold council and measured in national currency unit per troy ounce. 1 troy ounce is equal to the weight of the gold = 31.1 grams. This dataset contains all the currencies of the world. But, this analysis based on Indian currency.
- ARIMA models are designed to capture the autoregressive (AR) and moving average (MA) components of a time series. The autoregressive component accounts for the relationship between an observation and a certain number of lagged observations, while the moving average component considers the dependency between an observation and a residual error from a regression of previous observations. The "integrated" part of ARIMA represents the differencing operation used to make the time series stationary, meaning that its statistical properties do not vary with time.
- SARIMAX extends ARIMA by incorporating seasonal components. It includes additional parameters to model the seasonal patterns and trends in the data, enabling the capture of repeating patterns that occur over fixed time intervals, such as daily, monthly, or yearly cycles. Additionally, SARIMAX allows for the inclusion of exogenous variables, which are external factors that can affect the time series being analyzed. These variables can provide additional information and improve the accuracy of the forecasting model.
- Prophet has the ability to produce accurate forecasts with minimal effort. It decomposes a time series into three main components: trend, seasonality, and holidays. It automatically detects and handles various types of seasonal patterns, such as daily, weekly, monthly, and yearly seasonality. It can also handle irregular seasonalities. Sometimes, Prophet forecast better than ARIMA and SARIMAX.
- Let's dive into this gold price forecasting and check that which time series forecaster is forecasting best.

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Loading Data

In [2]:
```
data = pd.read_csv("/kaggle/input/gold-price-dataset/Gold Price.csv")
data.head()
```

Out[2]:

	Name	US dollar	Euro	Japanese yen	Pound sterling	Canadian dollar	Swiss franc	Indian rupee	Chinese renmimbi	US dollar.1	...	Vietnamese dong	Egyptian pound	Korean won	Euro.1	Russian ruble	US dollar.2	South African rand	Chi renmim
0	12/29/1978	78.53	55.93	NaN	64.00	NaN	NaN	NaN	NaN	78.53	...	NaN	NaN	NaN	55.93	NaN	78.53	NaN	
1	1/1/1979	78.53	55.93	NaN	64.00	NaN	NaN	NaN	NaN	78.53	...	NaN	NaN	NaN	55.93	NaN	78.53	NaN	
2	1/2/1979	78.80	56.02	132.96	64.43	59.66	90.97	14.66	NaN	78.80	...	NaN	NaN	31.05	56.02	NaN	78.80	11.49	
3	1/3/1979	75.96	54.68	134.67	62.42	59.81	92.56	14.73	NaN	75.96	...	NaN	NaN	31.21	54.68	NaN	75.96	11.35	
4	1/4/1979	77.54	55.82	134.53	63.97	59.75	92.69	14.81	NaN	77.54	...	NaN	NaN	31.38	55.82	NaN	77.54	11.40	

5 rows × 25 columns

In [3]:
```
data.shape
```

Out[3]:
```
(11581, 25)
```

Null Data

In [4]:
```
nulldata = data.isnull().sum()
nulldata[nulldata>0]
```

Out[4]:
```
Japanese yen             2
Canadian dollar          2
Swiss franc              2
Indian rupee             2
Chinese renmimbi      1571
Turkish lira          1045
Saudi riyal              2
Indonesian rupiah        2
UAE dirham               2
Thai baht                2
Vietnamese dong       2851
Egyptian pound        2591
Korean won               2
Russian ruble         3661
South African rand       2
Chinese renmimbi.1    1571
Canadian dollar.1        2
Australian dollar        2
dtype: int64
```

In [5]:
```
data = data.rename(columns = {"Name":"Date"})
data.head()
```

Out[5]:

	Date	US dollar	Euro	Japanese yen	Pound sterling	Canadian dollar	Swiss franc	Indian rupee	Chinese renmimbi	US dollar.1	...	Vietnamese dong	Egyptian pound	Korean won	Euro.1	Russian ruble	US dollar.2	South African rand	Chi renmim
0	12/29/1978	78.53	55.93	NaN	64.00	NaN	NaN	NaN	NaN	78.53	...	NaN	NaN	NaN	55.93	NaN	78.53	NaN	
1	1/1/1979	78.53	55.93	NaN	64.00	NaN	NaN	NaN	NaN	78.53	...	NaN	NaN	NaN	55.93	NaN	78.53	NaN	
2	1/2/1979	78.80	56.02	132.96	64.43	59.66	90.97	14.66	NaN	78.80	...	NaN	NaN	31.05	56.02	NaN	78.80	11.49	
3	1/3/1979	75.96	54.68	134.67	62.42	59.81	92.56	14.73	NaN	75.96	...	NaN	NaN	31.21	54.68	NaN	75.96	11.35	
4	1/4/1979	77.54	55.82	134.53	63.97	59.75	92.69	14.81	NaN	77.54	...	NaN	NaN	31.38	55.82	NaN	77.54	11.40	

5 rows × 25 columns

Datetime Extraction

In [6]:
```
data['Date'] = pd.to_datetime(data['Date'],format = "%m/%d/%Y")
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Week'] = data['Date'].dt.week
data['Day'] = data['Date'].dt.day
data.tail()
```

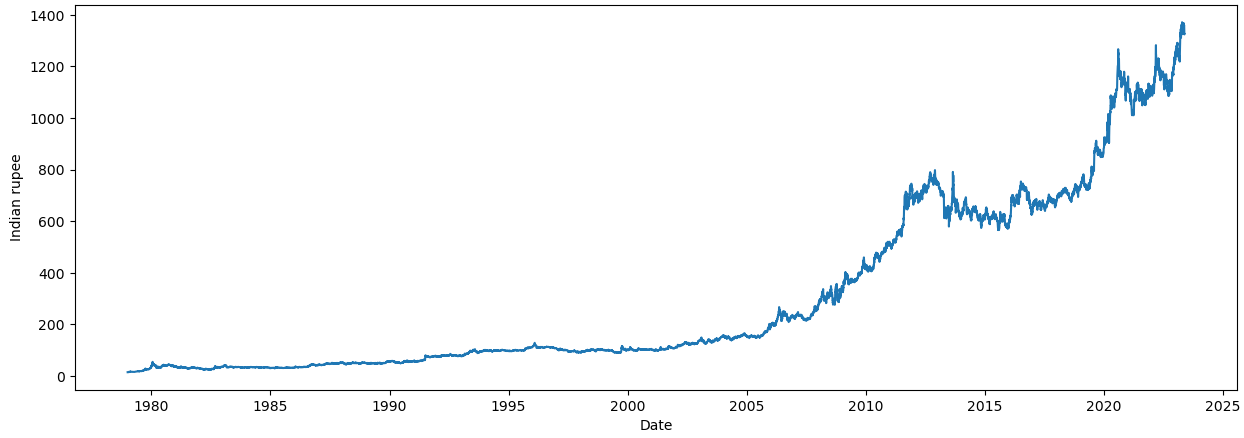
Out[6]:

	Date	US dollar	Euro	Japanese yen	Pound sterling	Canadian dollar	Swiss franc	Indian rupee	Chinese renmimbi	US dollar.1	...	Russian ruble	US dollar.2	South African rand	Chinese renmimbi.1	Canadian dollar.1	Australian dollar	Year	Mo
11576	2023-05-15	701.84	758.52	847.09	933.43	616.47	457.62	1359.32	589.37	701.84	...	2591.61	701.84	2272.44	589.37	616.47	643.75	2023	
11577	2023-05-16	697.52	753.78	844.10	928.72	611.10	455.10	1349.43	586.75	697.52	...	2610.43	697.52	2261.25	586.75	611.10	642.19	2023	
11578	2023-05-17	686.03	744.52	835.80	915.59	602.36	449.11	1330.05	579.92	686.03	...	2562.67	686.03	2250.99	579.92	602.36	633.61	2023	
11579	2023-05-18	681.13	742.70	837.01	913.37	598.01	449.12	1323.86	578.30	681.13	...	2532.52	681.13	2247.36	578.30	598.01	631.84	2023	
11580	2023-05-19	681.58	741.99	836.91	911.77	599.45	447.68	1325.86	578.21	681.58	...	2534.20	681.58	2256.24	578.21	599.45	628.04	2023	

5 rows × 29 columns

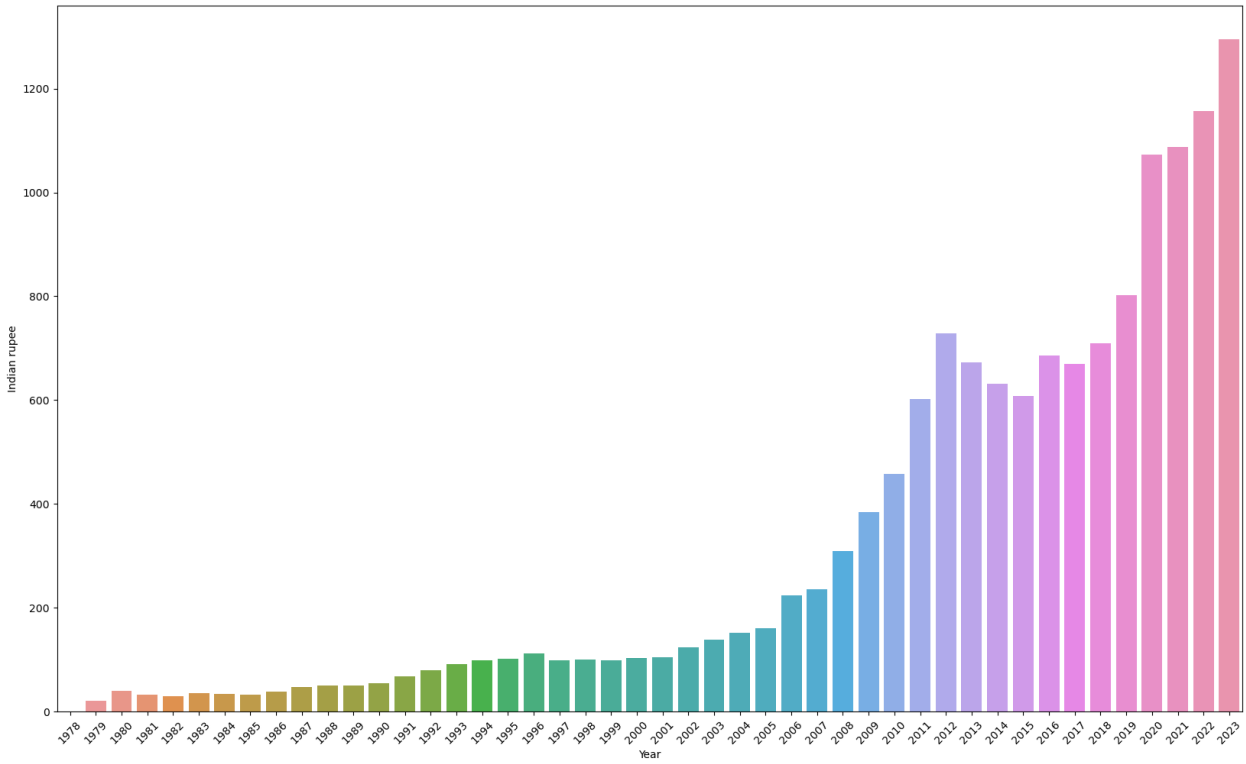
The Trend

In [7]:
```
plt.subplots(figsize = (15,5))
sns.lineplot(data,x = data['Date'], y = data['Indian rupee'])
plt.show()
```



Visualizing over years

In [8]:
```
plt.subplots(figsize = (20,12))
sns.barplot(data,x = data['Year'], y = data['Indian rupee'],ci=None)
plt.xticks(rotation = 45)
plt.show()
```



Separating the data of 10 years from now

In [9]:
```
data10 = data[(data['Year']>2013)]
data10.shape
```

Out[9]:
```
(2448, 29)
```

In [10]:
```
dataIndia = data10[['Date','Year','Month','Week','Day','Indian rupee']]
dataIndia.shape
```

Out[10]:
```
(2448, 6)
```

In [11]:
```
dataIndia.head()
```

Out[11]:

	Date	Year	Month	Week	Day	Indian rupee
9133	2014-01-01	2014	1	1	1	609.17
9134	2014-01-02	2014	1	1	2	623.05
9135	2014-01-03	2014	1	1	3	627.90
9136	2014-01-06	2014	1	2	6	635.28
9137	2014-01-07	2014	1	2	7	625.47

In [12]:
```
dataIndia.isnull().sum()
```

Out[12]:
```
Date            0
Year            0
Month           0
Week            0
Day             0
Indian rupee    0
dtype: int64
```

In [13]:
```
plt.subplots(figsize = (15,5))
sns.lineplot(dataIndia,x = dataIndia['Date'], y = dataIndia['Indian rupee'])
plt.show()
```

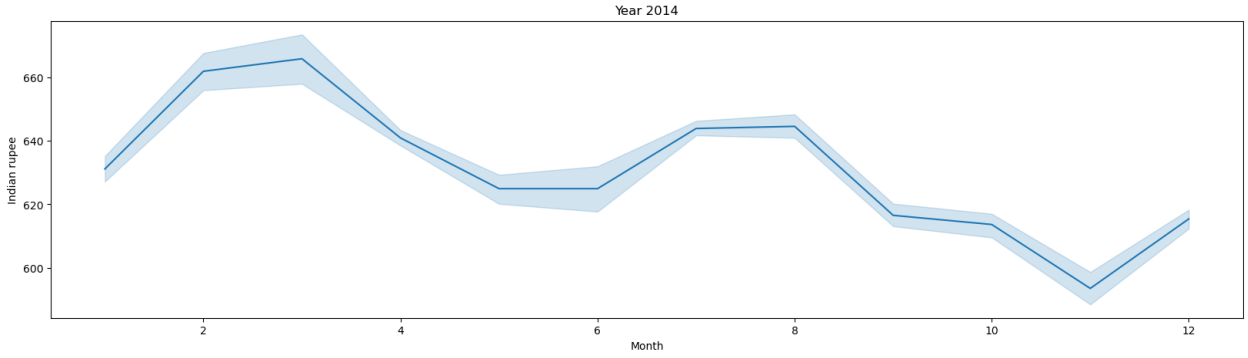
In [14]:
```
years = list(dataIndia['Year'].value_counts().index.sort_values())
years
```

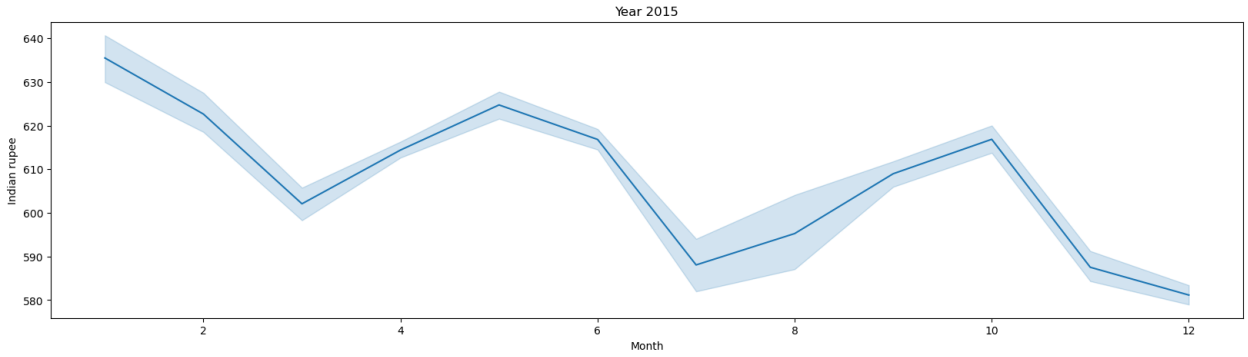
Out[14]:
```
[2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]
```

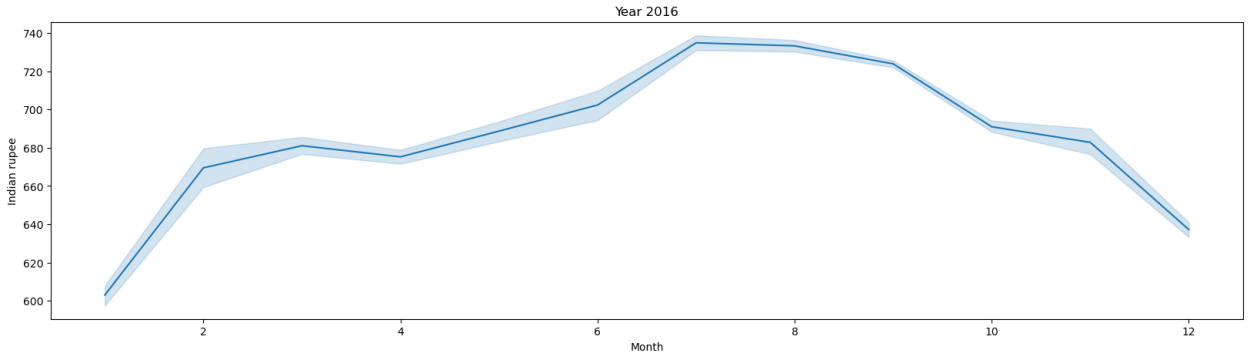
Helper function for collecting and plotting data of each year after separation

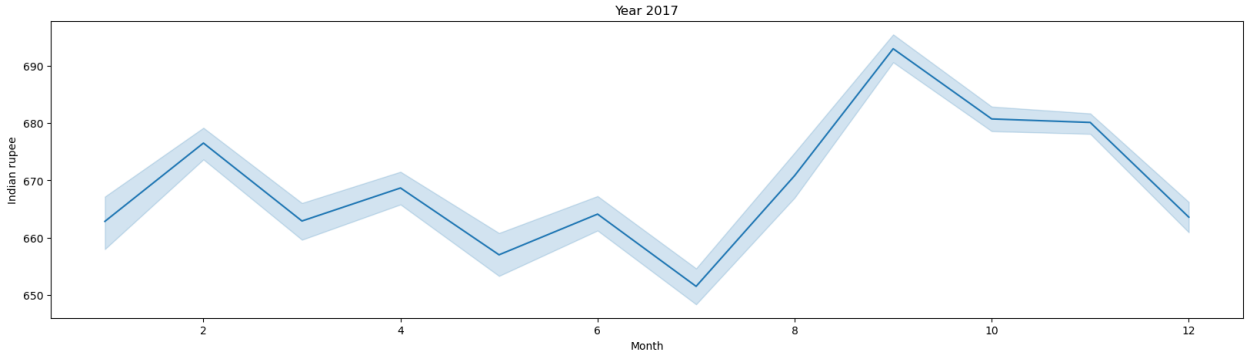
In [15]:
```
def Yearly_data(year):
    grp = dataIndia[(dataIndia.Year == year)]
    plt.subplots(figsize = (20,5))
    sns.lineplot(grp,x = grp.Month, y = grp['Indian rupee'])
    plt.title(f'Year {year}')
    plt.show()
```

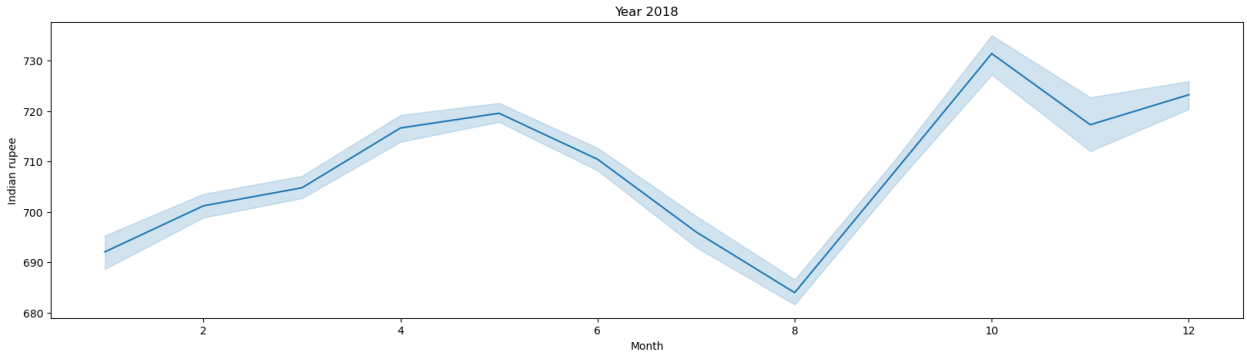
In [16]:
```
for i in range(len(years)):
    Yearly_data(years[i])
```

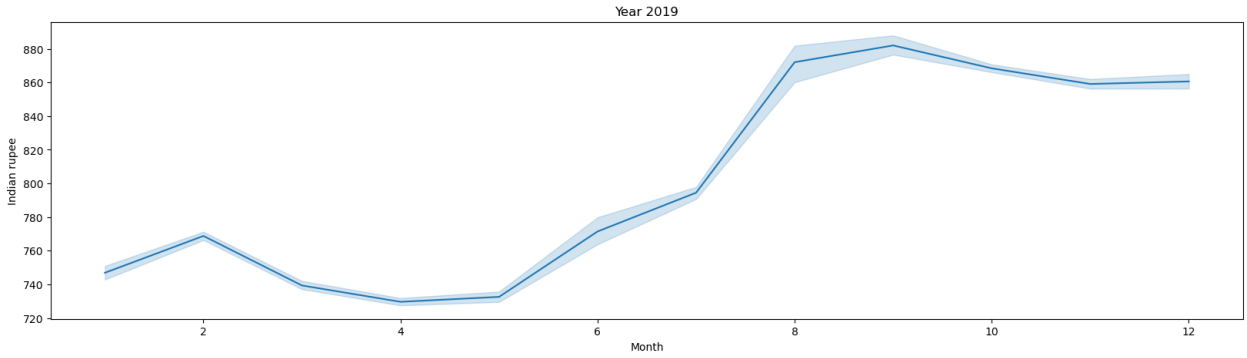


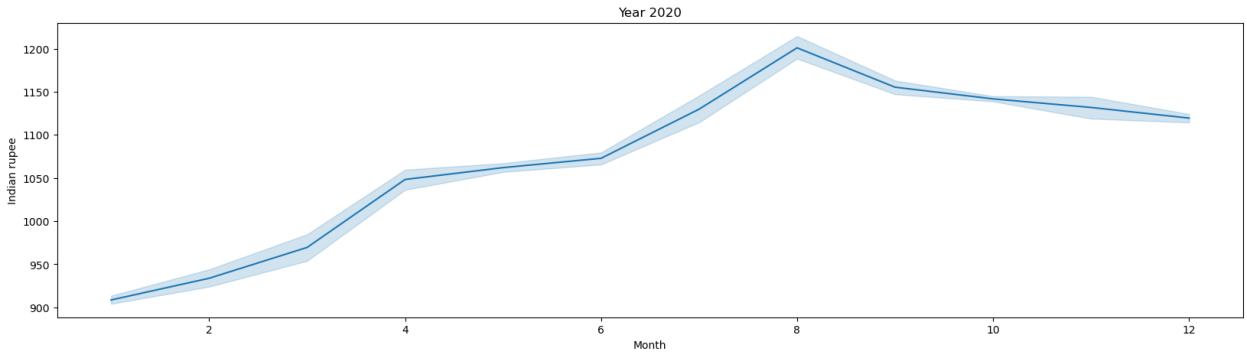


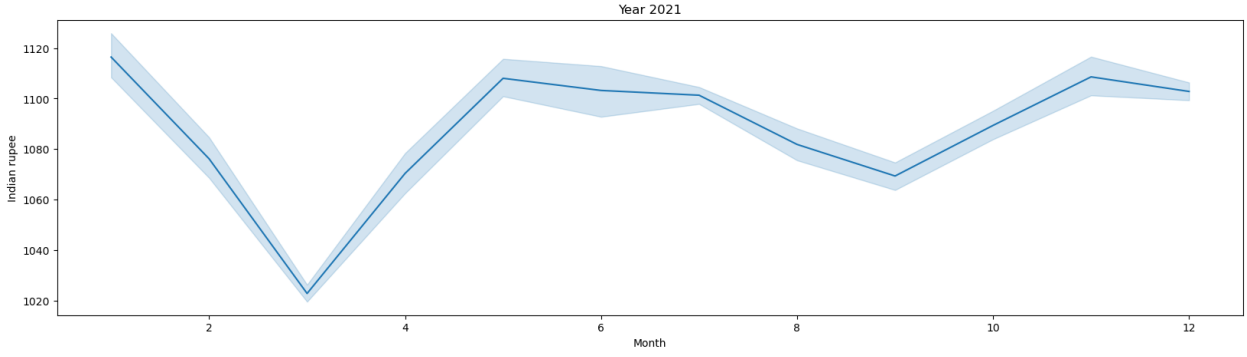


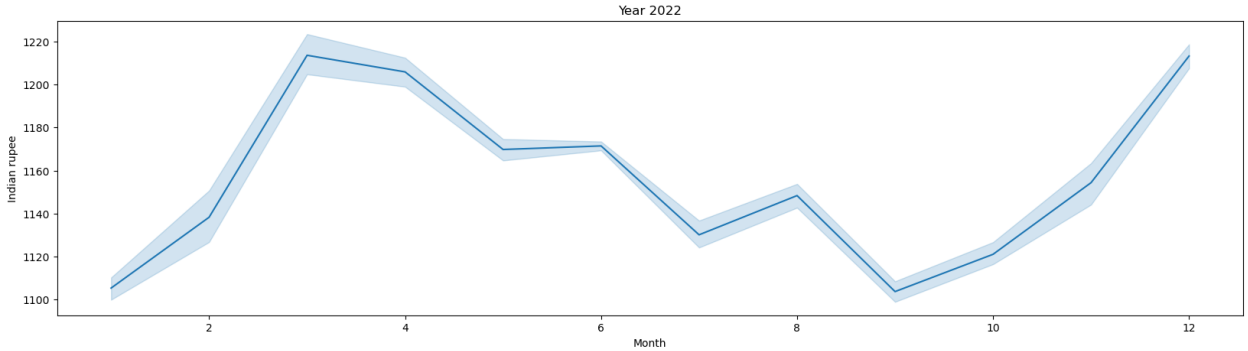

Year 2017
Indian rupee
690
680
670
660
650
2
4
6
8
10
12
Month

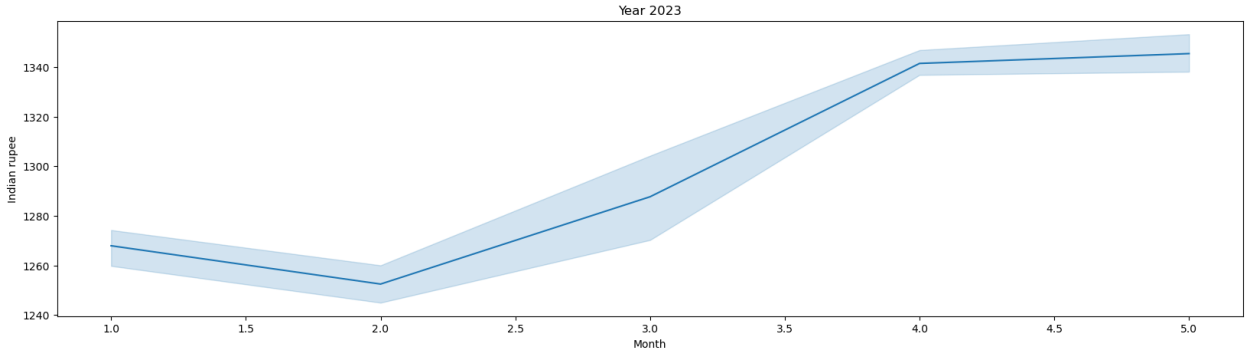

Year 2018
Indian rupee
730
720
710
700
690
680
2
4
6
8
10
12
Month


Year 2019
Indian rupee
880
860
840
820
800
780
760
740
720
2
4
6
8
10
12
Month


Year 2020
Indian rupee
1200
1150
1100
1050
1000
950
900
2
4
6
8
10
12
Month


Year 2021
Indian rupee
1120
1100
1080
1060
1040
1020
2
4
6
8
10
12
Month





> - It's tough to say which month is best to buy, but if we go with the majority, **April to August** will be the best month for buying gold as found from the past 10 years data.

In [17]:
```
dataIndia.head()
```

Out[17]:

	Date	Year	Month	Week	Day	Indian rupee
9133	2014-01-01	2014	1	1	1	609.17
9134	2014-01-02	2014	1	1	2	623.05
9135	2014-01-03	2014	1	1	3	627.90
9136	2014-01-06	2014	1	2	6	635.28
9137	2014-01-07	2014	1	2	7	625.47

Let's try ARIMA

In [18]:
```
dataIndia.set_index(dataIndia['Date'],inplace = True)
dataIndia = dataIndia.drop(columns = 'Date')
dataIndia.tail()
```

Out[18]:

Date	Year	Month	Week	Day	Indian rupee
2023-05-15	2023	5	20	15	1359.32
2023-05-16	2023	5	20	16	1349.43
2023-05-17	2023	5	20	17	1330.05
2023-05-18	2023	5	20	18	1323.86
2023-05-19	2023	5	20	19	1325.86

Helper function for checking stationarity

In [19]:
```
def check_stationarity(series):
    from statsmodels.tsa.stattools import adfuller
    result = adfuller(series)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    if (result[1] <= 0.05) & (result[4]['5%'] > result[0]):
        print("\u001b[32mStationary\u001b[0m")
    else:
        print("\x1b[31mNon-stationary\x1b[0m")
```

In [20]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
```

In [21]:
```
check_stationarity(dataIndia['Indian rupee'])
```

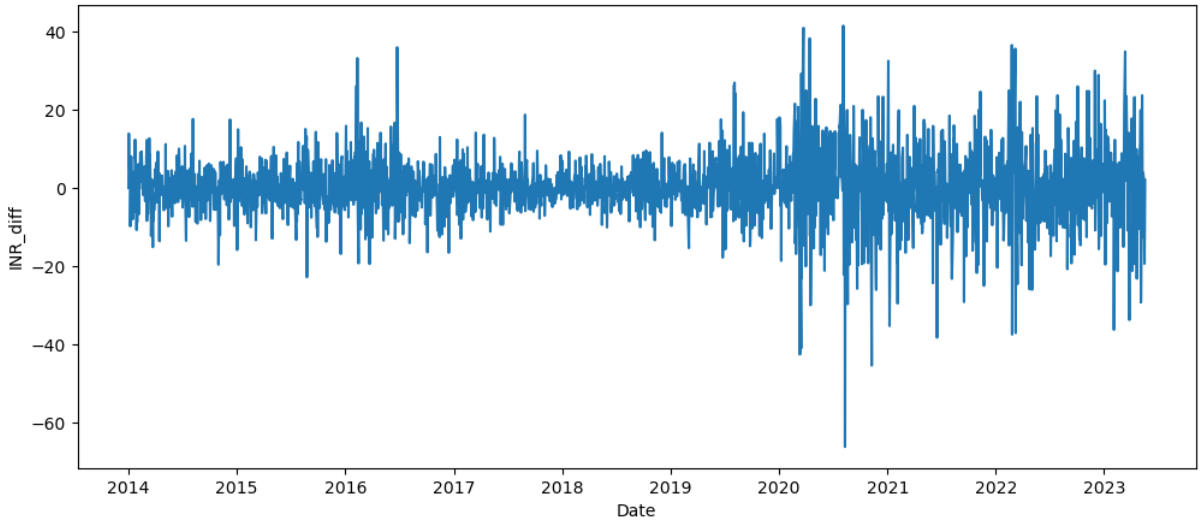
```
ADF Statistic: 0.327753
p-value: 0.978583
Critical Values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
Non-stationary
```

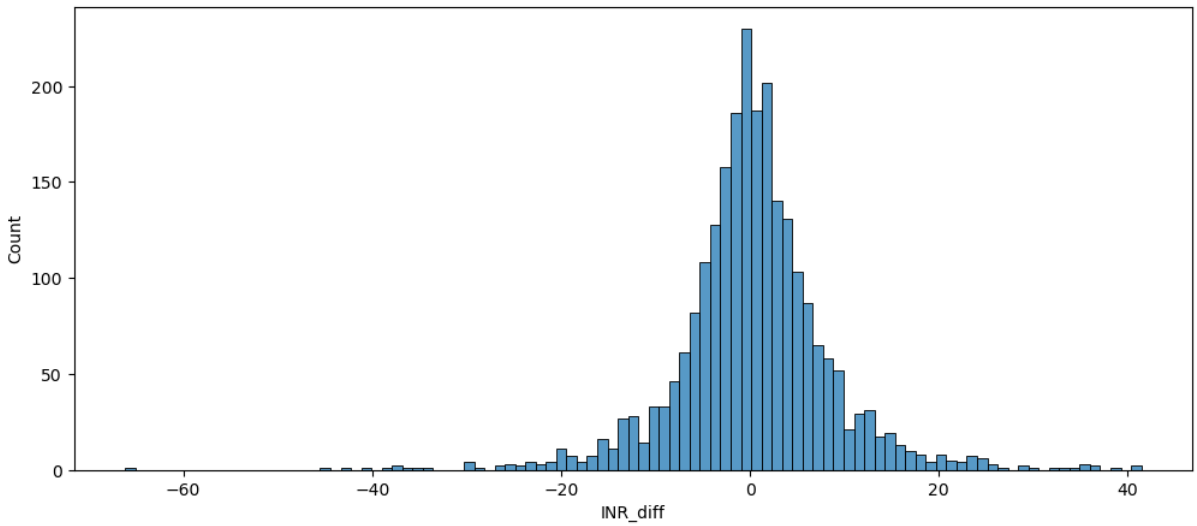
Try differencing

In [22]:
```
dataIndia['INR_diff'] = dataIndia['Indian rupee'].diff().fillna(0)

check_stationarity(dataIndia['INR_diff'])
```

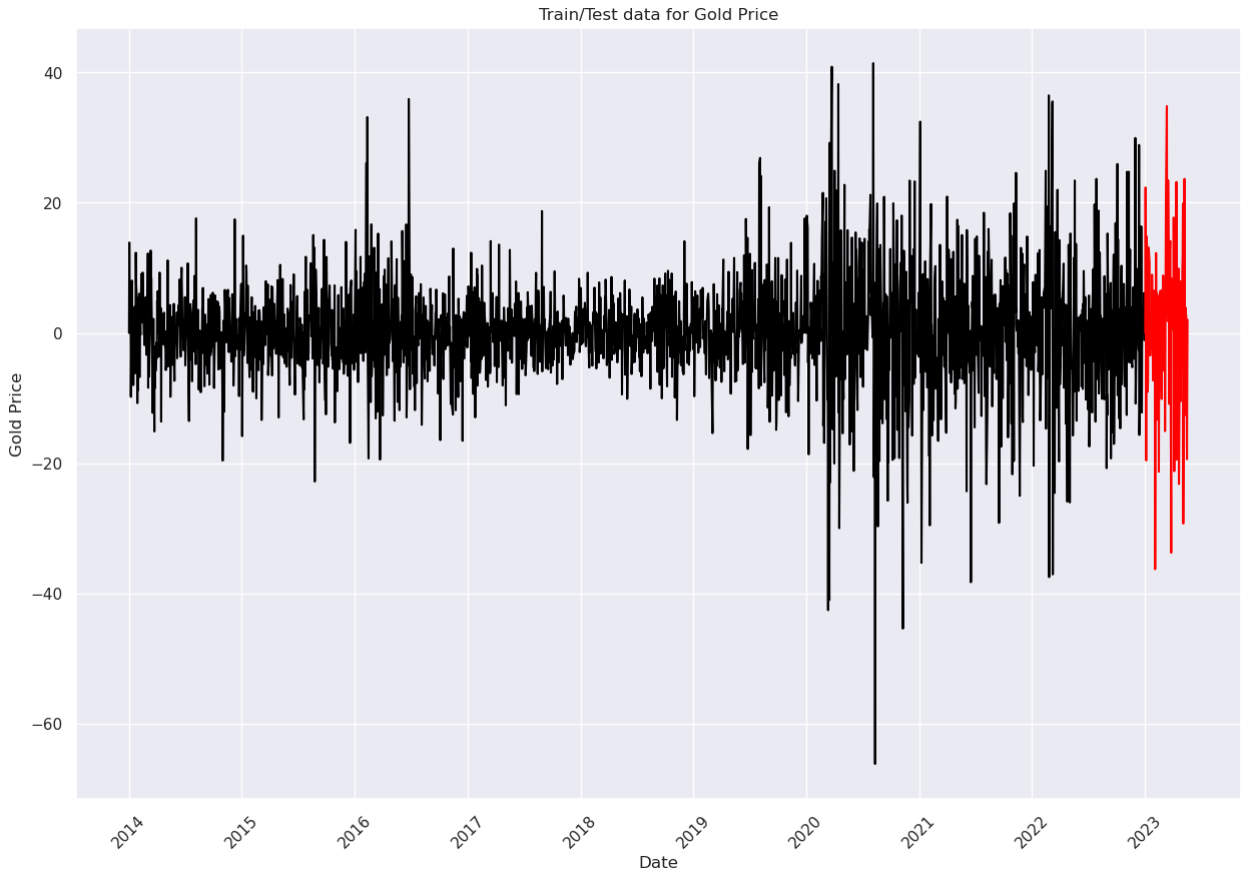
```
ADF Statistic: -14.456611
p-value: 0.000000
Critical Values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
Stationary
```

In [23]:
```
f, ax = plt.subplots(nrows=1, ncols=1, figsize=(12,5))

sns.lineplot(data=dataIndia, x=dataIndia.index, y='INR_diff')
plt.show()

f, ax = plt.subplots(nrows=1, ncols=1, figsize=(12,5))

sns.histplot(dataIndia['INR_diff'] )
plt.show()
```



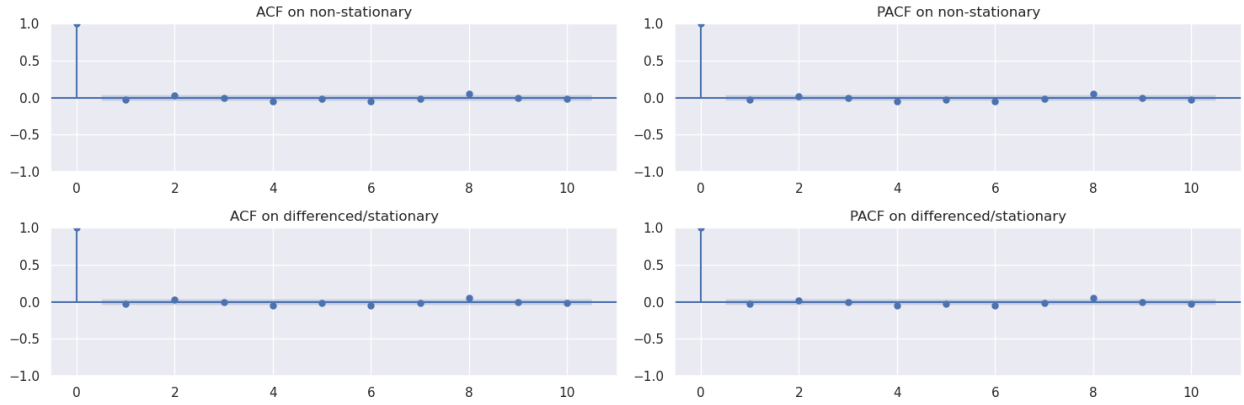


In [24]:
```
train = dataIndia[dataIndia.index<pd.to_datetime("2023-01-01",format = "%Y-%m-%d")]
test = dataIndia[dataIndia.index>pd.to_datetime("2023-01-01",format = "%Y-%m-%d")]
train = train[['INR_diff']]
test = test[['INR_diff']]
sns.set()
plt.subplots(figsize = (15,10))
plt.plot(train,color ='black')
plt.plot(test,color = 'red')
plt.ylabel("Gold Price")
plt.xlabel("Date")
plt.xticks(rotation = 45)
plt.title("Train/Test data for Gold Price")
plt.show()
```



ACF and PACF

In [25]:
```
f, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,5))

plot_acf(train['INR_diff'], lags=10, ax=ax[0, 0], title='ACF on non-stationary')
plot_pacf(train['INR_diff'], lags=10, ax=ax[0, 1], method='ols', title='PACF on non-stationary')

plot_acf(train['INR_diff'], lags=10, ax=ax[1, 0], title='ACF on differenced/stationary')
plot_pacf(train['INR_diff'], lags=10, ax=ax[1, 1], method='ols', title='PACF on differenced/stationary')

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



> - Differencing of 1 will give perfect results.

Helper function for checking order

In [26]:
```
def check_order(p,d,q):
    y = train['INR_diff']
    from statsmodels.tsa.arima.model import ARIMA
    ARIMAmodel = ARIMA(y, order = (p,d,q))
    ARIMAmodel = ARIMAmodel.fit()
    print(ARIMAmodel.summary(),"\n")

    y_pred = ARIMAmodel.get_forecast(len(test.index))
    y_pred_df = y_pred.conf_int(alpha = 0.05)
    y_pred_df["Predictions"] = ARIMAmodel.predict(start = y_pred_df.index[0], end = y_pred_df.index[-1])
    y_pred_df.index = test.index
    y_pred_out = y_pred_df["Predictions"]
```

```
from sklearn.metrics import mean_squared_error
arima_rmse = np.sqrt(mean_squared_error(test['INR_diff'].values, y_pred_df["Predictions"]))
print("RMSE: ",arima_rmse,'\n')

ARIMAmodel.plot_diagnostics(figsize = (20,12))


sns.set()
plt.subplots(figsize = (15,5))
plt.plot(train,color ='black',label = 'Train')
plt.plot(test,color = 'red',label = 'Test')
plt.plot(y_pred_out, color='Yellow', label = 'ARIMA Predictions')
plt.ylabel("Gold Price")
plt.xlabel("Date")
plt.xticks(rotation = 45)
plt.title("Train/Test and predictions for Gold Price")
plt.tight_layout()
plt.show()
```

In [27]:
```
check_order(0,0,0)
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

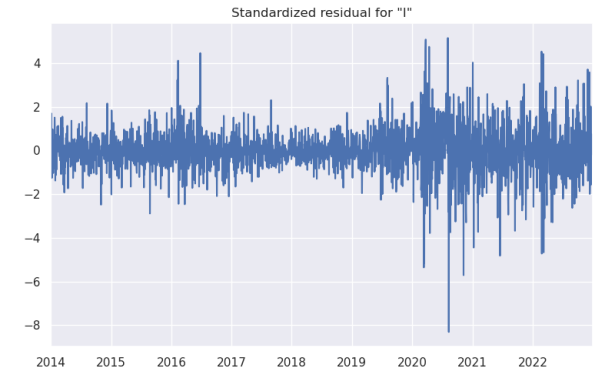
Dep. Variable:	INR_diff	No. Observations:	2348
Model:	ARIMA	Log Likelihood	-8211.769
Date:	Mon, 29 May 2023	AIC	16427.537
Time:	14:02:57	BIC	16439.060
Sample:	01-01-2014	HQIC	16431.734
	- 12-30-2022		
Covariance Type:	opg		

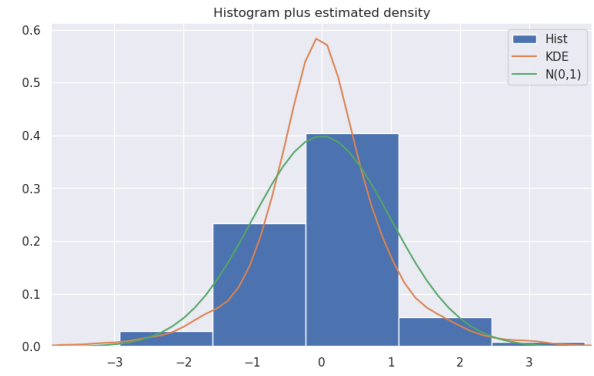
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.2631	0.165	1.591	0.112	-0.061	0.587
sigma2	63.8632	0.924	69.133	0.000	62.053	65.674

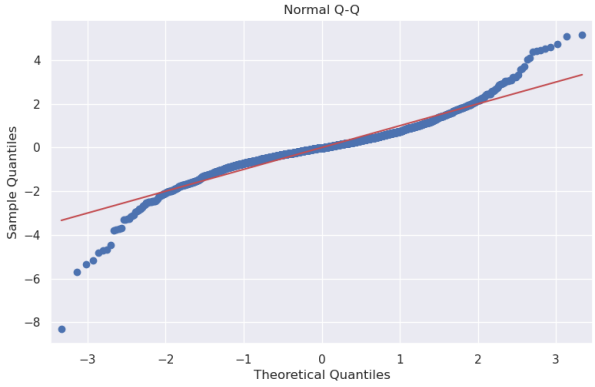
Ljung-Box (L1) (Q):	2.05	Jarque-Bera (JB):	3753.84
Prob(Q):	0.15	Prob(JB):	0.00
Heteroskedasticity (H):	3.55	Skew:	-0.20
Prob(H) (two-sided):	0.00	Kurtosis:	9.18

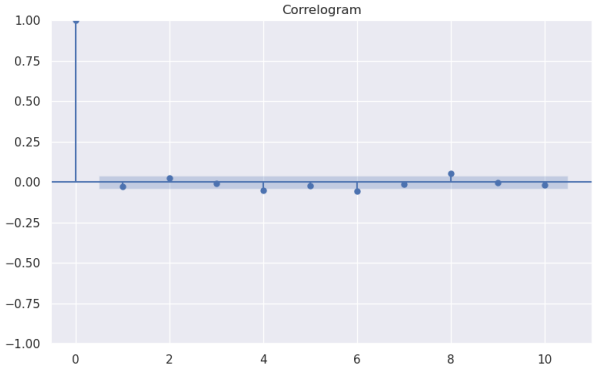
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

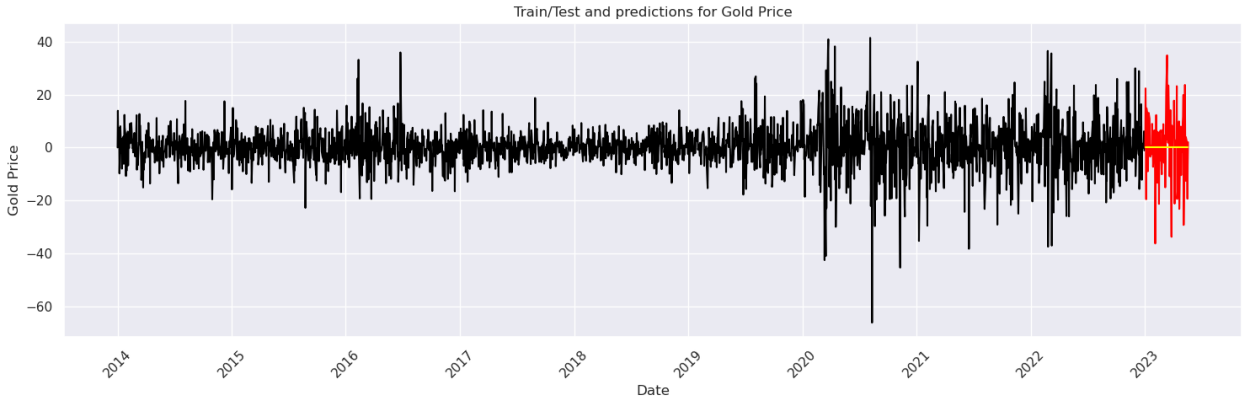
RMSE: 12.136689951443172











In [28]:
```
check_order(0,1,0)
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

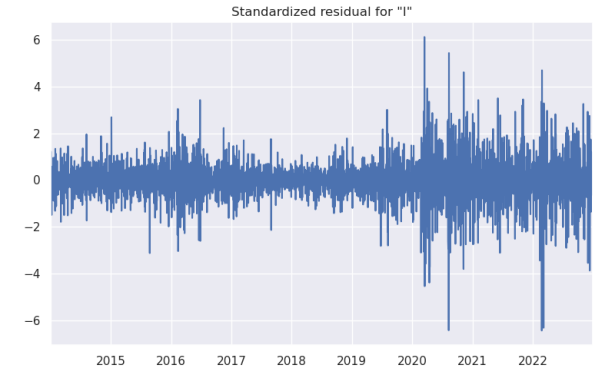
Dep. Variable:	INR_diff	No. Observations:	2348
Model:	ARIMA(0, 1, 0)	Log Likelihood	-9056.354
Date:	Mon, 29 May 2023	AIC	18114.709
Time:	14:03:01	BIC	18120.470
Sample:	01-01-2014	HQIC	18116.807
	- 12-30-2022		
Covariance Type:	opg		

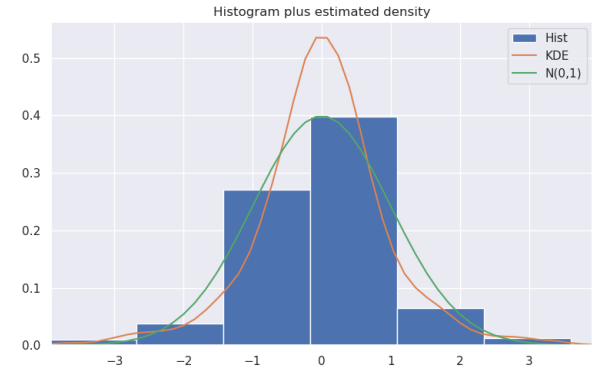
	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	131.5669	2.096	62.767	0.000	127.459	135.675

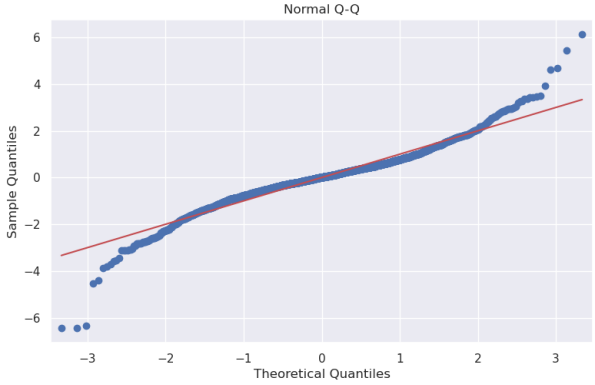
Ljung-Box (L1) (Q):	650.50	Jarque-Bera (JB):	2190.15
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	3.53	Skew:	-0.21
Prob(H) (two-sided):	0.00	Kurtosis:	7.71

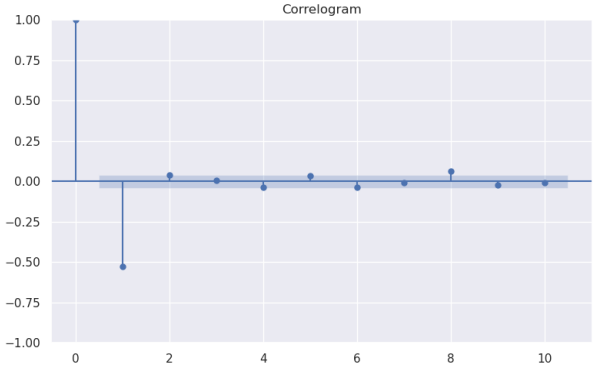
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

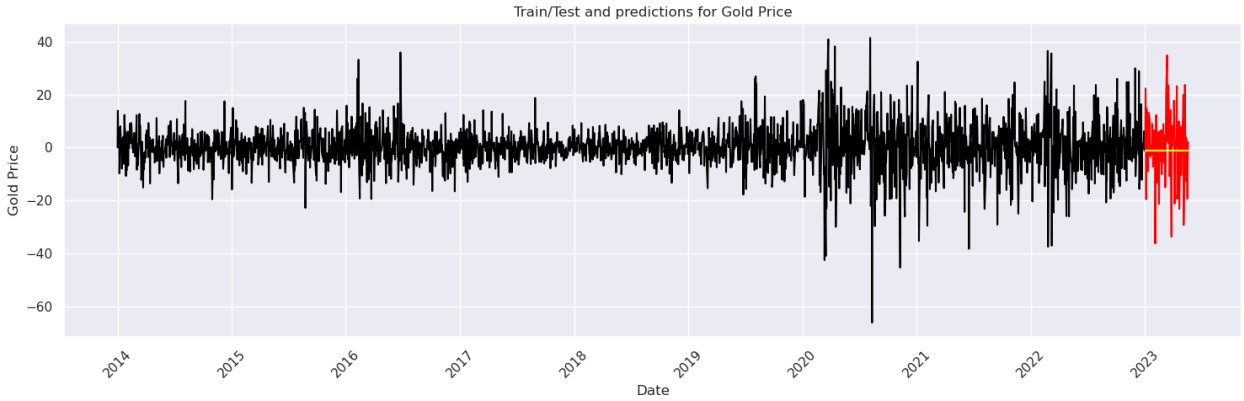
RMSE: 12.29554964204532











In [29]:
```
check_order(2,1,2)
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

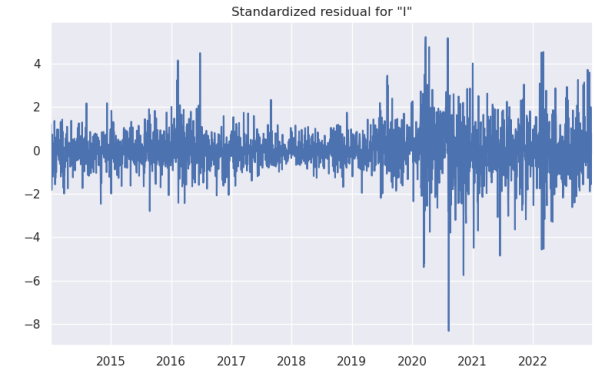
Dep. Variable:	INR_diff	No. Observations:	2348
Model:	ARIMA(2, 1, 2)	Log Likelihood	-8211.424
Date:	Mon, 29 May 2023	AIC	16432.848
Time:	14:03:04	BIC	16461.652
Sample:	01-01-2014	HQIC	16443.339
	- 12-30-2022		
Covariance Type:	opg		

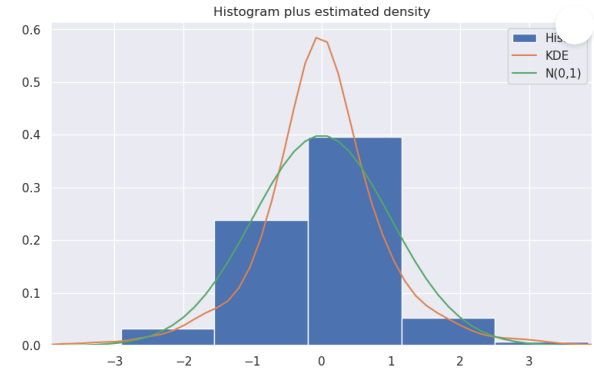
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.0286	0.015	-70.860	0.000	-1.057	-1.000
ar.L2	-0.0295	0.015	-2.038	0.042	-0.058	-0.001
ma.L1	-7.11e-06	375.996	-1.89e-08	1.000	-736.938	736.938
ma.L2	-1.0000	376.016	-0.003	0.998	-737.978	735.978
sigma2	63.8080	2.4e+04	0.003	0.998	-4.7e+04	4.71e+04

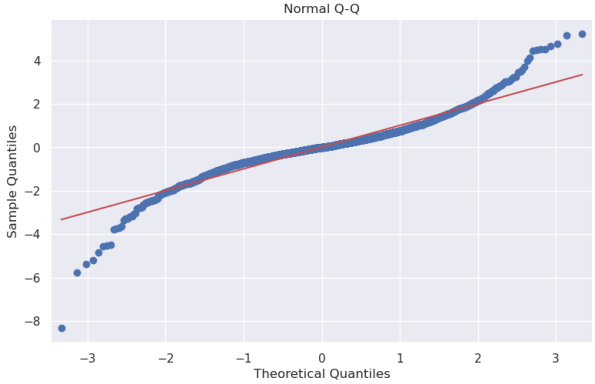
Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	3835.65
Prob(Q):	0.98	Prob(JB):	0.00
Heteroskedasticity (H):	3.56	Skew:	-0.21
Prob(H) (two-sided):	0.00	Kurtosis:	9.25

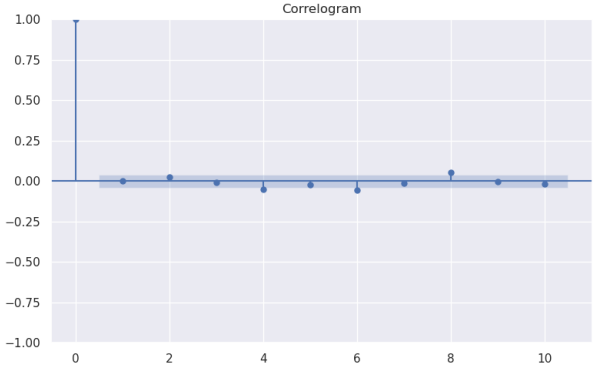
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

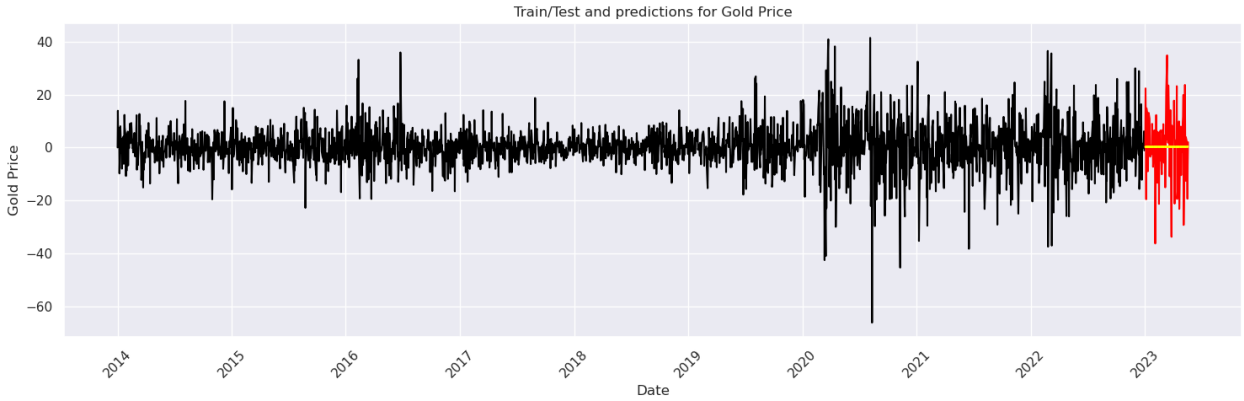
RMSE: 12.137488093483809











In [30]:
```
check_order(1,1,2)
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

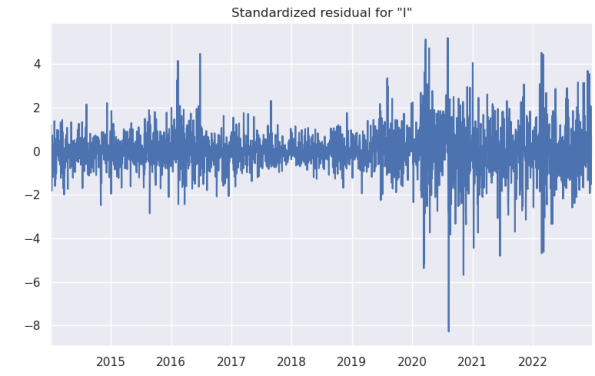
Dep. Variable:	INR_diff	No. Observations:	2348
Model:	ARIMA(1, 1, 2)	Log Likelihood	-8211.833
Date:	Mon, 29 May 2023	AIC	16431.665
Time:	14:03:08	BIC	16454.709
Sample:	01-01-2014	HQIC	16440.058
	- 12-30-2022		
Covariance Type:	opg		

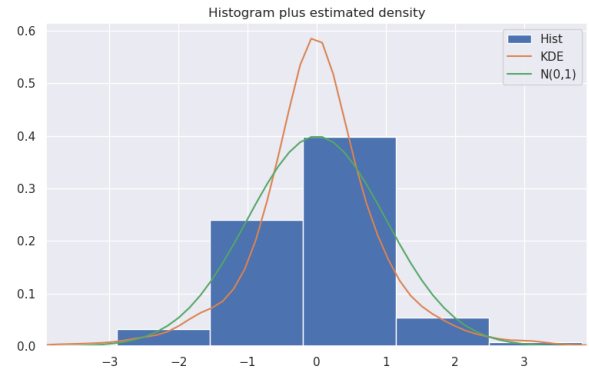
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.9939	0.011	-88.261	0.000	-1.016	-0.972
ma.L1	-0.0092	0.029	-0.319	0.750	-0.066	0.048
ma.L2	-0.9907	0.027	-36.776	0.000	-1.043	-0.938
sigma2	63.8507	1.986	32.156	0.000	59.959	67.743

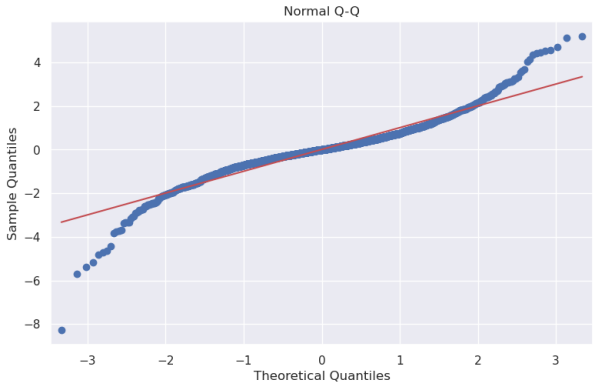
Ljung-Box (L1) (Q):	1.55	Jarque-Bera (JB):	3729.15
Prob(Q):	0.21	Prob(JB):	0.00
Heteroskedasticity (H):	3.56	Skew:	-0.22
Prob(H) (two-sided):	0.00	Kurtosis:	9.16

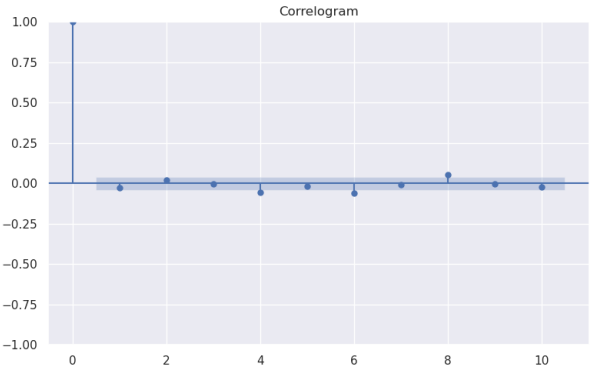
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

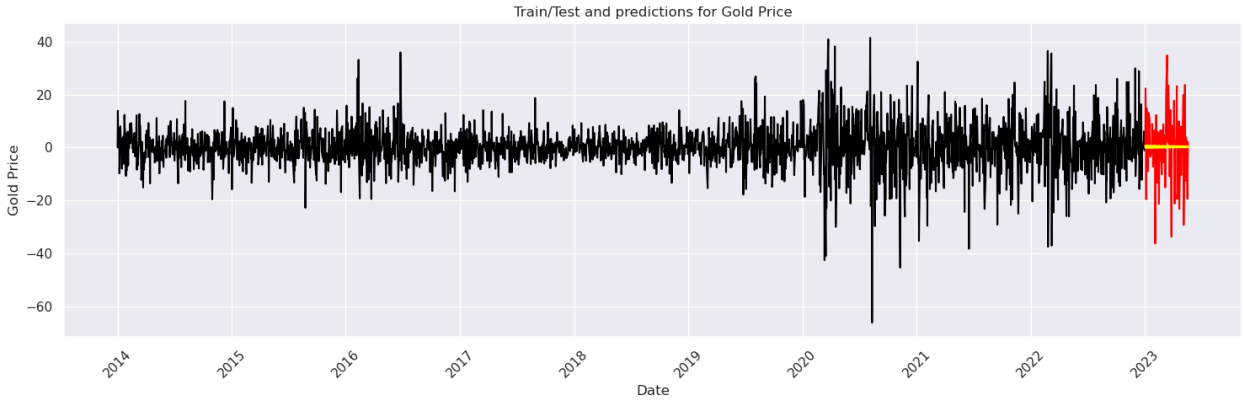
RMSE: 12.141379595326978











- After checking standard orders, we found that none of the orders are predicting the gold price well.

Let's Try SARIMAX

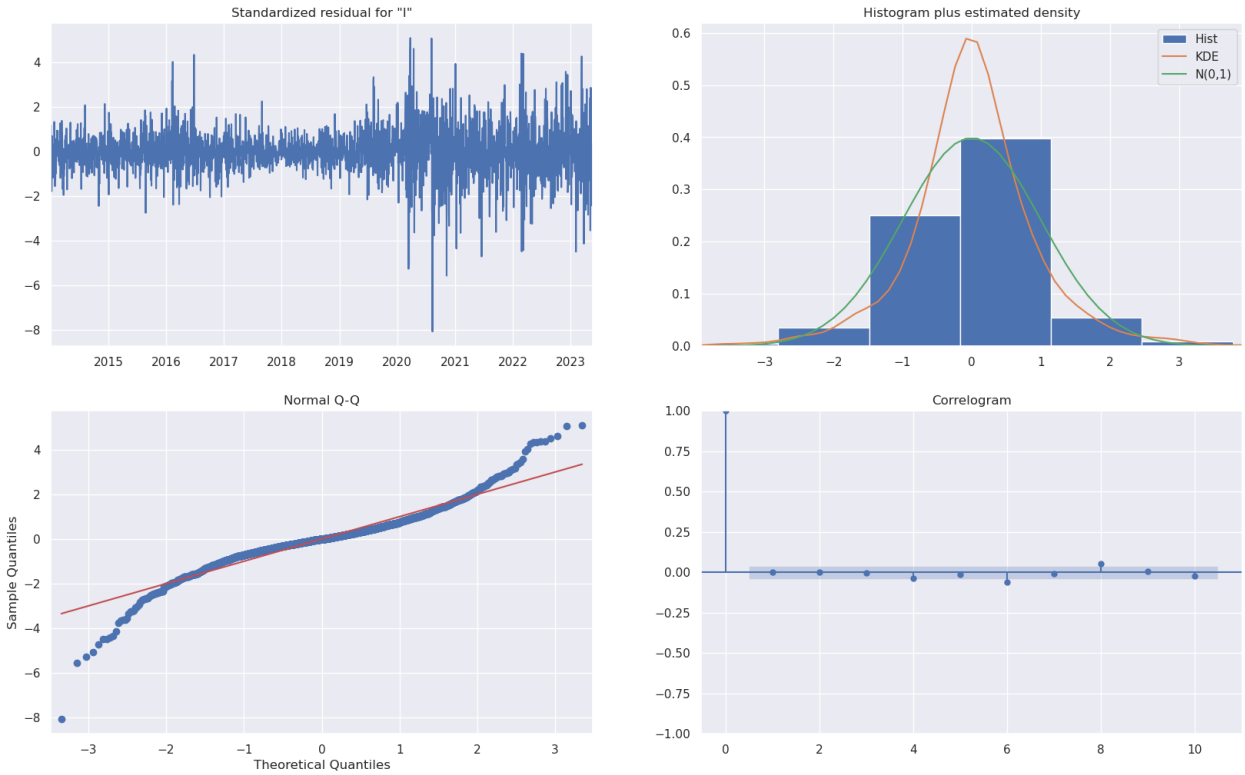
Helper function for checking order for SARIMAX

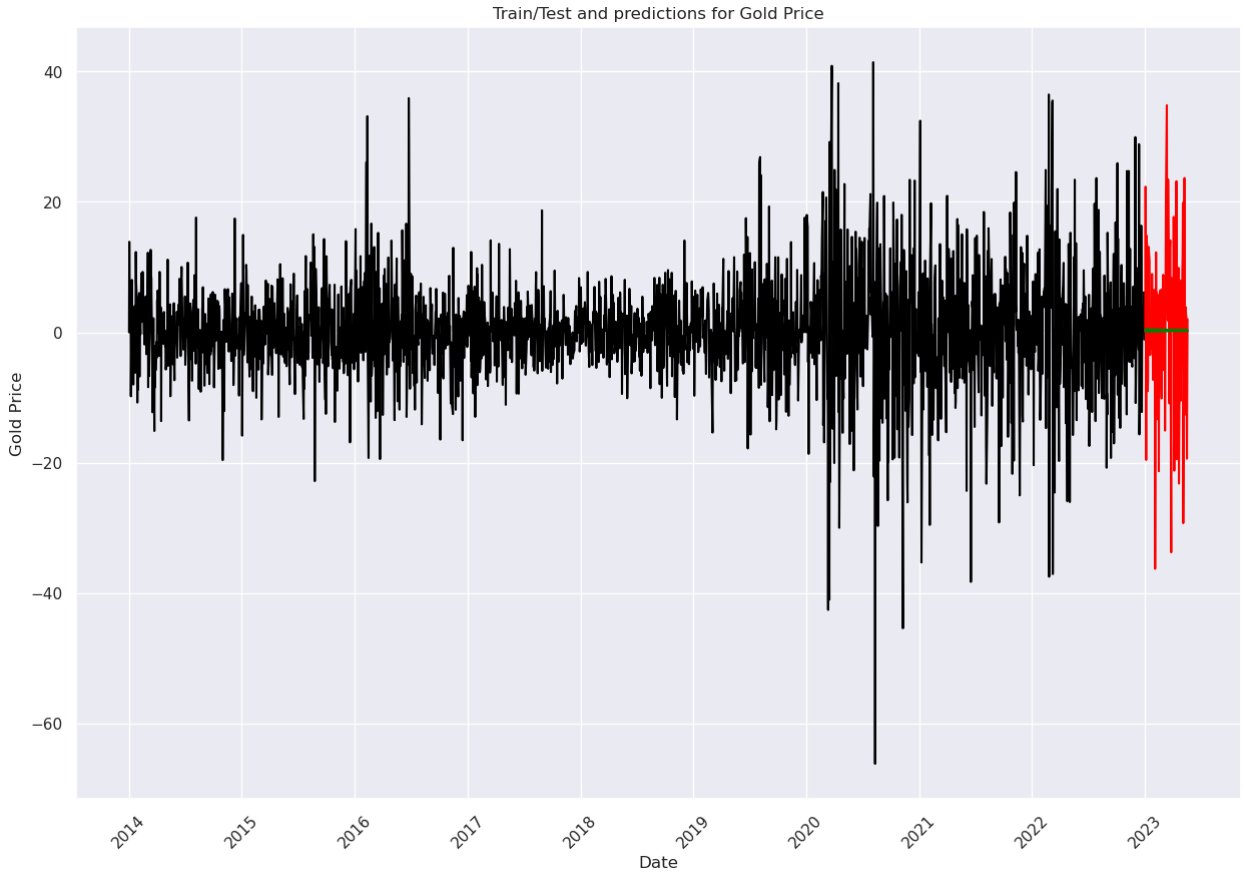
In [31]:
```
def check_seasonal_order(p,d,q):
    y = dataIndia['INR_diff']
    from statsmodels.tsa.statespace.sarimax import SARIMAX
    SARIMAmodel = SARIMAX(y, order = (p,d,q),seasonal_order = (0,0,0,12))
    ResultModel = SARIMAmodel.fit()
    print(ResultModel.summary(),"\n")

    y_pred = ResultModel.get_forecast(len(test.index))
    y_pred_df = y_pred.conf_int(alpha = 0.05)
    y_pred_df["Predictions"] = ResultModel.predict(start = y_pred_df.index[0], end = y_pred_df.index[-1])
    y_pred_df.index = test.index
    y_pred_out = y_pred_df["Predictions"]


    from sklearn.metrics import mean_squared_error
    sarima_rmse = np.sqrt(mean_squared_error(test['INR_diff'].values, y_pred_df["Predictions"]))
    print("RMSE: ",sarima_rmse,'\n')

    ResultModel.plot_diagnostics(figsize = (20,12))


    sns.set()
    plt.subplots(figsize = (15,10))
    plt.plot(train,color ='black',label = 'Train')
    plt.plot(test,color = 'red',label = 'Test')
    plt.plot(y_pred_out, color='green', label = 'ARIMA Predictions')
    plt.ylabel("Gold Price")
    plt.xlabel("Date")
    plt.xticks(rotation = 45)
    plt.title("Train/Test and predictions for Gold Price")
    plt.show()
```

In [32]:
```
check_seasonal_order(2,1,2)
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided, so inferred f
requency B will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  3.57609D+00    |proj g|=  5.90767D-02

At iterate    5    f=  3.53466D+00    |proj g|=  2.25843D-02

At iterate   10    f=  3.52324D+00    |proj g|=  8.20462D-04

At iterate   15    f=  3.52321D+00    |proj g|=  6.80598D-04

At iterate   20    f=  3.52317D+00    |proj g|=  1.76131D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     24     29      1     0     0   1.402D-05   3.523D+00
  F =   3.5231662851013223

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
                                 SARIMAX Results
==========================================================================
Dep. Variable:              INR_diff   No. Observations:                 2448
Model:               SARIMAX(2, 1, 2)   Log Likelihood               -8624.711
Date:                Mon, 29 May 2023   AIC                           17259.422
Time:                        14:03:13   BIC                           17288.435
Sample:                    01-01-2014   HQIC                          17269.967
                         - 05-19-2023
Covariance Type:                  opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1         -1.0202      0.024    -42.217      0.000      -1.068      -0.973
ar.L2         -0.0261      0.015     -1.700      0.089      -0.056       0.004
ma.L1         -0.0077      0.024     -0.322      0.748      -0.055       0.039
ma.L2         -0.9922      0.022    -45.446      0.000      -1.035      -0.949
sigma2        67.2326      1.555     43.239      0.000      64.185      70.280
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              3369.79
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               3.42   Skew:                            -0.23
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.73
===================================================================================

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

RMSE:  12.135846463698607
```





- SARIMAX is also not predicting well as it was expected

Let's Try Prophet

Model Selection

In [33]:
```
dataIndia = dataIndia.reset_index().drop(columns = 'INR_diff')
dataIndia.head()
```

Out[33]:

	Date	Year	Month	Week	Day	Indian rupee
0	2014-01-01	2014	1	1	1	609.17
1	2014-01-02	2014	1	1	2	623.05
2	2014-01-03	2014	1	1	3	627.90
3	2014-01-06	2014	1	2	6	635.28
4	2014-01-07	2014	1	2	7	625.47

In [34]:
```
from sklearn.model_selection import train_test_split
x = dataIndia[['Year','Month','Week','Day']]
y = dataIndia[['Indian rupee']]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2, random_state = 99)
```

In [35]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[35]:
```
0.8415031930138567
```

In [36]:
```
model.score(x_train,y_train)
```

Out[36]:
```
0.8493969510646942
```

> - There are no overfitting issues, but let's find the best model with the best accuracy

Importing models

In [37]:
```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
from xgboost import XGBRegressor
xgb = XGBRegressor()
from lightgbm import LGBMRegressor
lgbm = LGBMRegressor()
```

In [38]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

Helper function for model evaluation

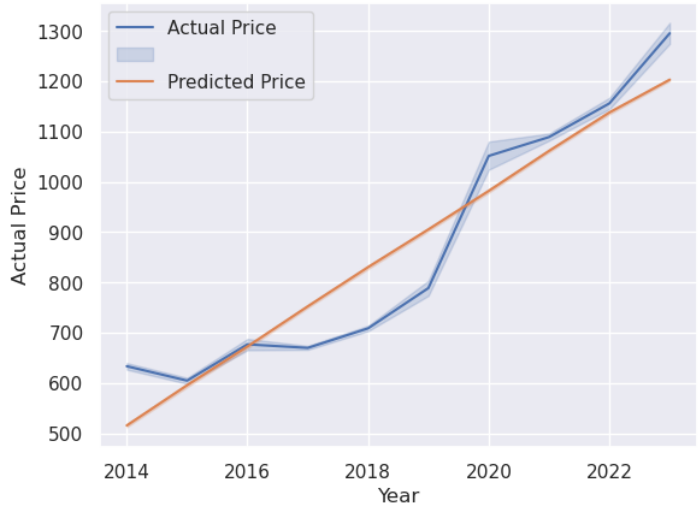
In [39]:
```
def evaluation_metrics(model):
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)
    mse = mean_squared_error(y_test,y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test,y_pred)
    r2score = r2_score(y_test,y_pred)
    trainscore = model.score(x_train,y_train)

    print(f'Mean Squared Error = {mse}')
    print(f'Mean Absolute Error = {mae}')
    print(f'Root Mean Squared Error = {rmse}')
    print(f'r2 score = {r2score}')

    Comparison = x_test.copy("deep")
    Comparison['Actual Price'] = y_test
    Comparison['Predicted Price'] = y_pred

    sns.lineplot(Comparison, x = 'Year',y = 'Actual Price')
    sns.lineplot(Comparison, x = 'Year',y = 'Predicted Price')
    plt.legend(['Actual Price','','Predicted Price'])
    plt.show()

    return {'Training Score': trainscore,
           'Testing Score': r2score}
```

In [40]:
```
results = {}
```

Linear Regression

In [41]:
```
results['Linear Regression'] = evaluation_metrics(model)
```

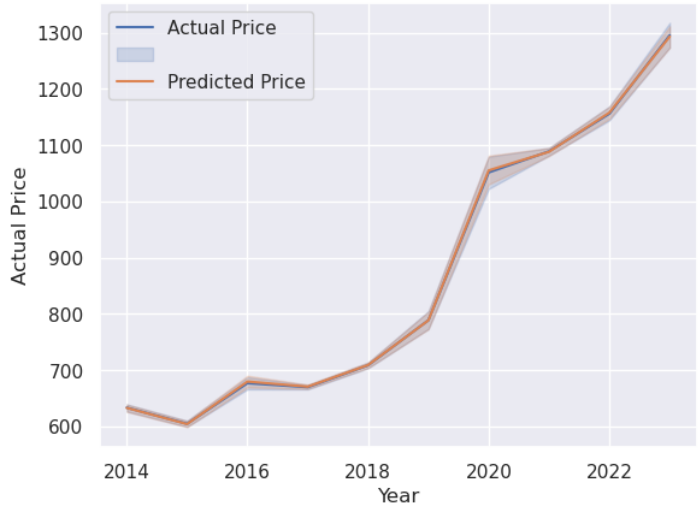
```
Mean Squared Error = 7985.63573389267
Mean Absolute Error = 74.62807807771928
Root Mean Squared Error = 89.36238433419662
r2 score = 0.8415031930138567
```



Decision Tree Regressor

In [42]:
```
results['Decision Tree Regressor'] = evaluation_metrics(dtr)
```

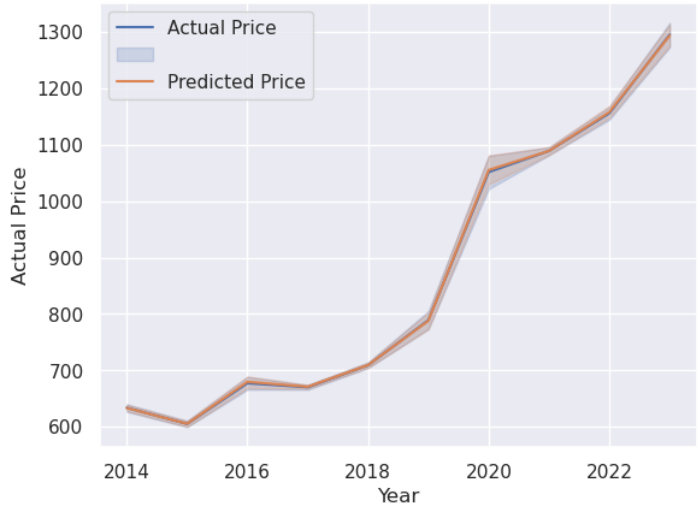
```
Mean Squared Error = 76.2006957142857
Mean Absolute Error = 5.906591836734695
Root Mean Squared Error = 8.729300986578805
r2 score = 0.9984875885448197
```



Random Forest Regressor

In [43]:
```
results['Random Forest Regressor'] = evaluation_metrics(rfr)
```

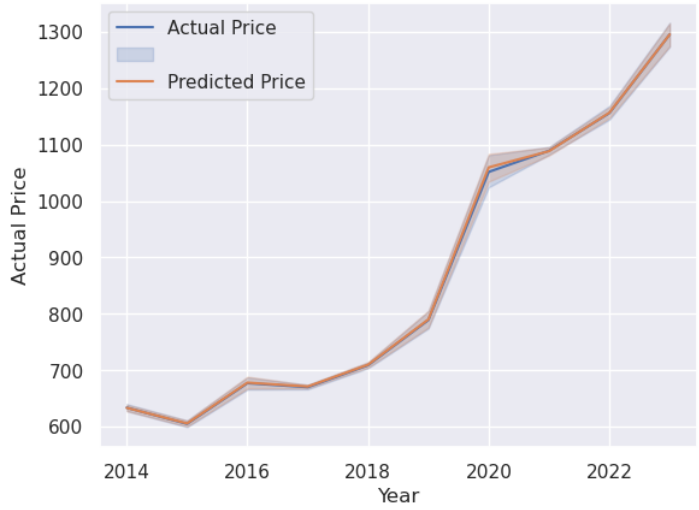
```
Mean Squared Error = 70.18512658581592
Mean Absolute Error = 5.604885918367294
Root Mean Squared Error = 8.377656389815467
r2 score = 0.9986069839856886
```



XGB Regressor

In [44]:
```
results['XGB Regressor'] = evaluation_metrics(xgb)
```

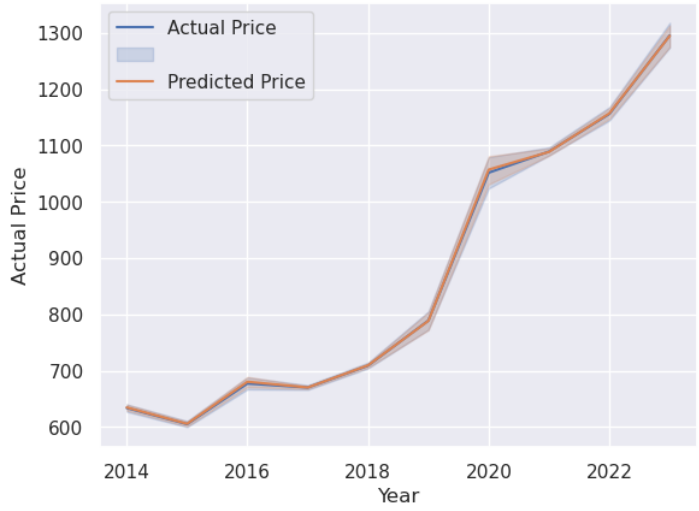
```
Mean Squared Error = 115.93633434047514
Mean Absolute Error = 7.125565788424744
Root Mean Squared Error = 10.767373604573919
r2 score = 0.9976989259942507
```



LGBM Regressor

In [45]:
```
results['LGBM Regressor'] = evaluation_metrics(lgbm)
```

```
Mean Squared Error = 132.88575799566127
Mean Absolute Error = 8.35661415136721
Root Mean Squared Error = 11.527608511554392
r2 score = 0.9973625182717947
```



Model comparison

In [46]:
```
resultsdf = pd.DataFrame(results)
resultsdf
```

Out[46]:

	Linear Regression	Decision Tree Regressor	Random Forest Regressor	XGB Regressor	LGBM Regressor
Training Score	0.849397	1.000000	0.999789	0.999713	0.998575
Testing Score	0.841503	0.998488	0.998607	0.997699	0.997363

> - Random Forest Regressor is best in all of the models

In [47]:
```
pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.1.1)
Requirement already satisfied: setuptools>=42 in /opt/conda/lib/python3.7/site-packages (from prophet) (59.8.0)
Requirement already satisfied: holidays>=0.14.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.21.13)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.64.1)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.3.5)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.5.3)
Requirement already satisfied: cmdstanpy>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.1.0)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.4.0)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.2)
Requirement already satisfied: wheel>=0.37.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.38.4)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.21.6)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (0.5.12)
Requirement already satisfied: backports.zoneinfo in /opt/conda/lib/python3.7/site-packages (from holidays>=0.14.2->prophet) (0.2.1)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.14.2->prophet) (0.3.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.14.2->prophet) (2.2.4)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet) (4.1.4)
Requirement already satisfied: pytz in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet) (2022.7.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (1.4.4)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (3.0.9)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (23.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.8.0->prophet) (1.16.0)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from kiwisolver>=1.0.1->matplotlib>=2.0.0->prophet)
(4.4.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recomm
ended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

Making time dataframe for Prophet

In [48]:
```
time_data = dataIndia[['Date','Indian rupee']]
time_data.columns = ['ds','y']
time_data.head()
```

Out[48]:

	ds	y
0	2014-01-01	609.17
1	2014-01-02	623.05
2	2014-01-03	627.90
3	2014-01-06	635.28
4	2014-01-07	625.47

In [49]:
```
from prophet import Prophet
m = Prophet(daily_seasonality=True)
model_fit = m.fit(time_data)
```

```
14:03:46 - cmdstanpy - INFO - Chain [1] start processing
14:03:48 - cmdstanpy - INFO - Chain [1] done processing
```

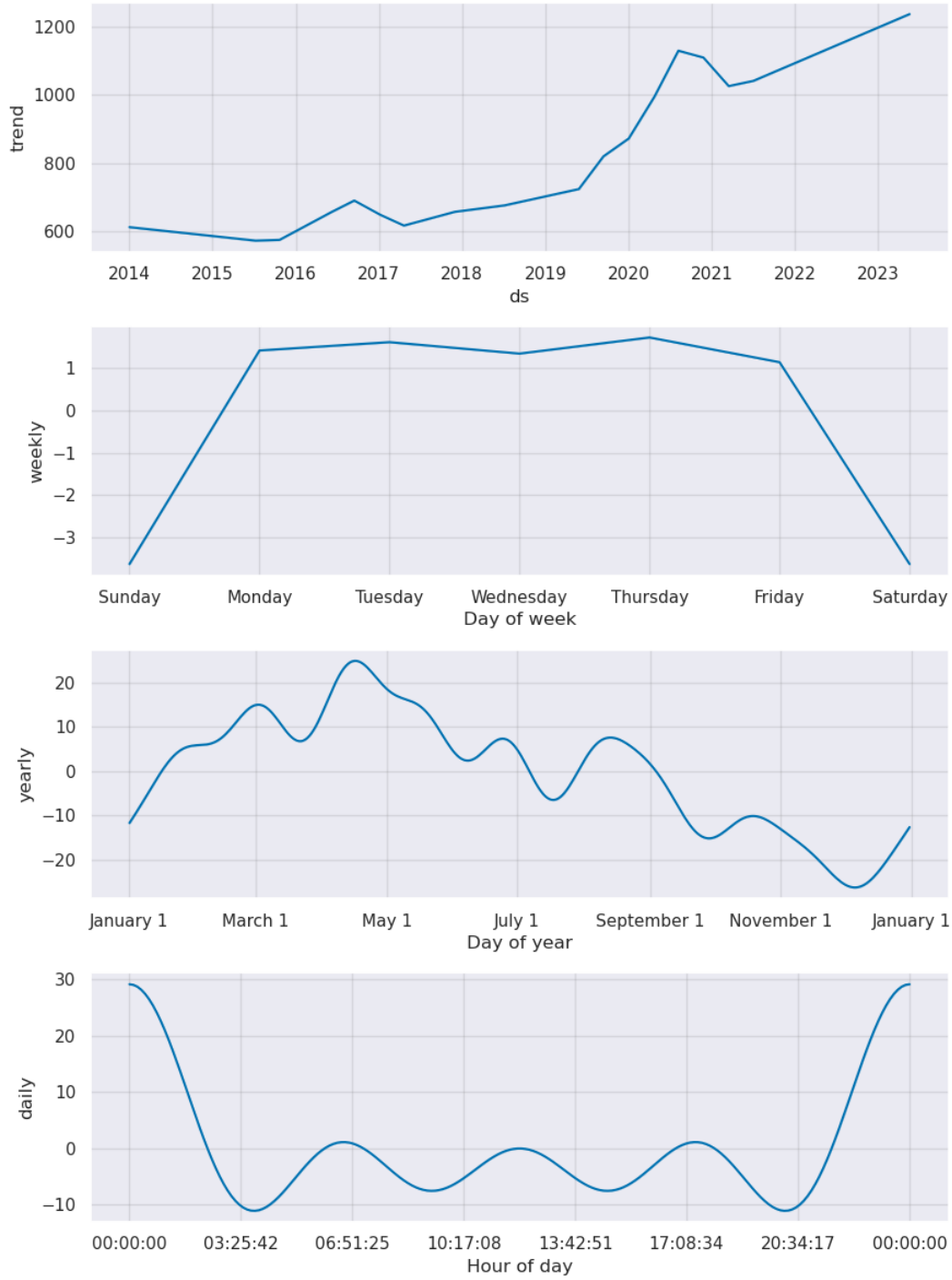
In [50]:
```
forecast = model_fit.predict(time_data)
forecast[['ds','yhat','yhat_upper','yhat_lower']].tail()
```

Out[50]:

	ds	yhat	yhat_upper	yhat_lower
2443	2023-05-15	1281.642774	1315.635540	1247.050731
2444	2023-05-16	1281.841554	1316.169976	1247.779814
2445	2023-05-17	1281.524603	1316.686618	1244.436156
2446	2023-05-18	1281.806680	1315.260755	1247.167047
2447	2023-05-19	1281.064695	1318.194072	1247.558369

In [51]:
```
model_fit.plot(forecast);
```

In [52]:
```
model_fit.plot_components(forecast);
```



Month of June and August are awesome

Forecasting for next 6 months

In [53]:
```
m2 = Prophet(daily_seasonality=True)
m2.fit(time_data)
future = m2.make_future_dataframe(periods = 336)
future.tail(10)
```

```
14:03:54 - cmdstanpy - INFO - Chain [1] start processing
14:03:56 - cmdstanpy - INFO - Chain [1] done processing
```

Out[53]:

	ds
2774	2024-04-10
2775	2024-04-11
2776	2024-04-12
2777	2024-04-13
2778	2024-04-14
2779	2024-04-15
2780	2024-04-16
2781	2024-04-17
2782	2024-04-18
2783	2024-04-19

In [54]:
```
forecast2 = m2.predict(future)
forecast2[['ds','yhat','yhat_upper','yhat_lower']].tail()
```

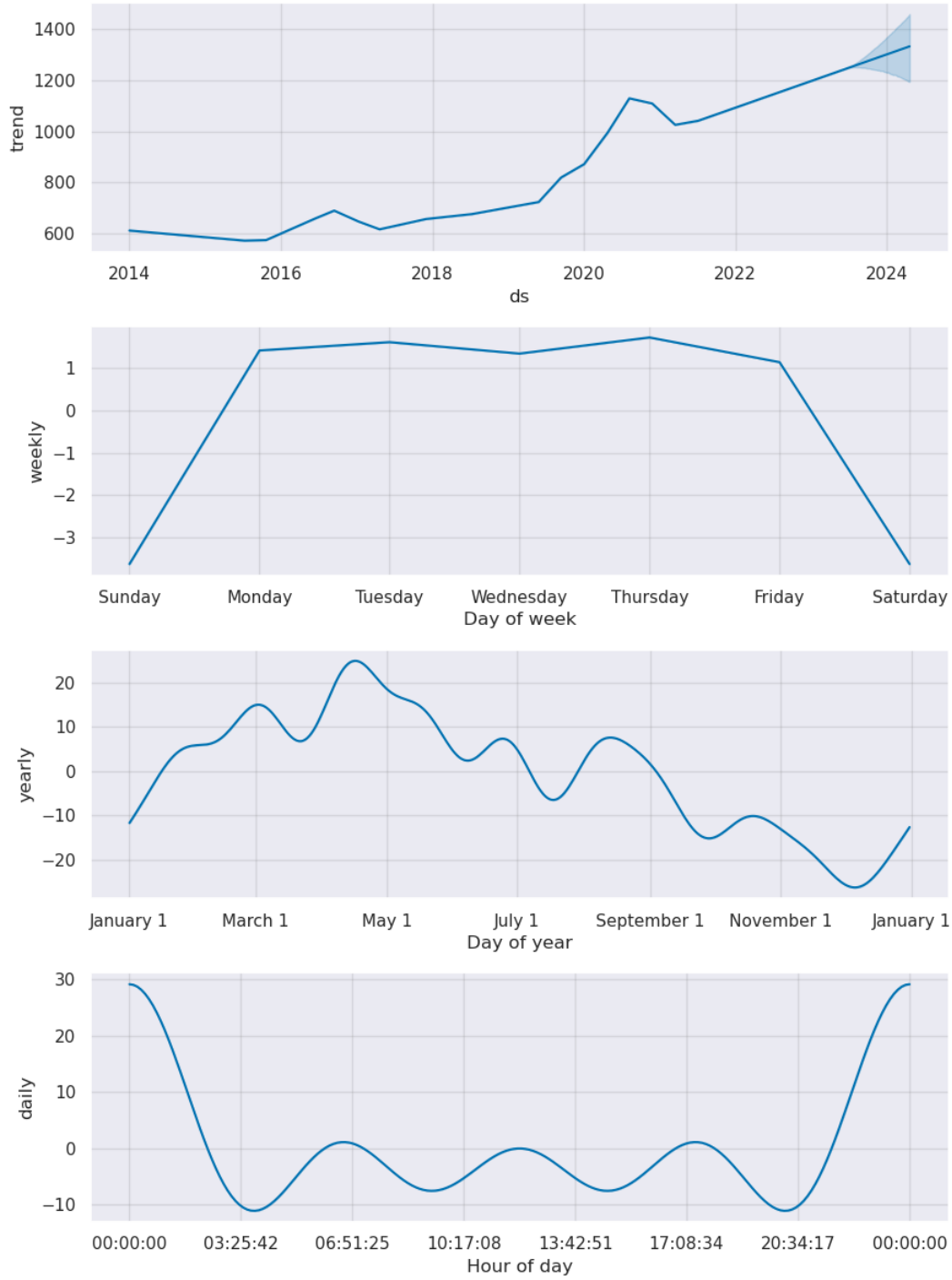
Out[54]:

	ds	yhat	yhat_upper	yhat_lower
2779	2024-04-15	1387.098424	1513.910925	1247.925641
2780	2024-04-16	1387.651489	1507.342940	1250.645201
2781	2024-04-17	1387.623199	1513.693186	1243.390041
2782	2024-04-18	1388.140714	1517.181194	1248.928742
2783	2024-04-19	1387.593558	1514.821713	1250.925275

In [55]:
```
m2.plot(forecast2);
```

- In the starting of 2024, the gold price will inflate

In [56]:
```
m2.plot_components(forecast2);
```



In [57]:
```
predictions = forecast2[['ds','yhat']]
predictions = predictions.rename(columns = {'ds':'Date','yhat':'Indian rupee'})
predictions.head()
```

Out[57]:

	Date	Indian rupee
0	2014-01-01	630.941574
1	2014-01-02	631.996564
2	2014-01-03	632.092243
3	2014-01-06	634.467958
4	2014-01-07	635.382572

In [58]:
```
plt.subplots(figsize = (15,8))
sns.lineplot(time_data,x = 'ds',y='y',legend='full')
sns.lineplot(predictions,x = 'Date',y = 'Indian rupee',legend='full')
plt.legend(labels = ['Actual','','Prediction'])
plt.show()
```

In [59]:
```
predictions['Date'] = pd.to_datetime(predictions['Date'])
predictions['Year'] = predictions['Date'].dt.year
predictions['Month'] = predictions['Date'].dt.month
predictions['Week'] = predictions['Date'].dt.week
predictions['Day'] = predictions['Date'].dt.day
```

Model Evaluation

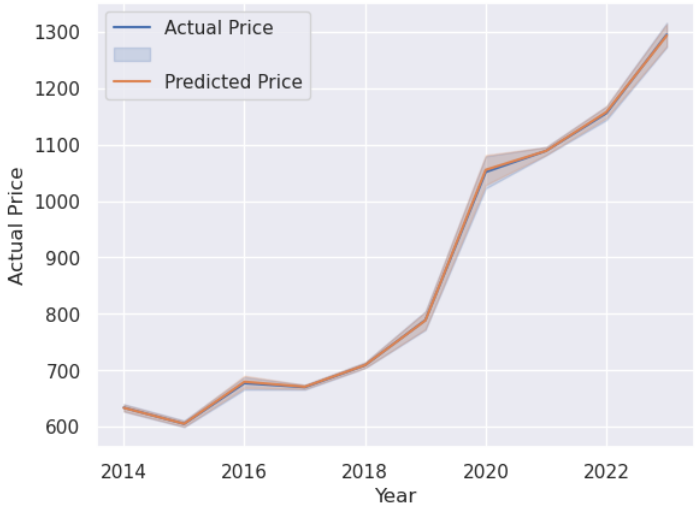
In [60]:
```
x_time  = predictions.drop(columns=['Date','Indian rupee'])
y_time = predictions[['Indian rupee']]
```

In [61]:
```
x_time_train,x_time_test,y_time_train,y_time_test = train_test_split(x_time,y_time,test_size=0.2,random_state=99)
```

In [62]:
```
rfr_time_model = RandomForestRegressor()
rfr_time_model.fit(x_time_train,y_time_train)
rfr_time_pred = rfr_time_model.predict(x_time_test)
```

In [63]:
```
results['Random Forest Evaluation Regressor'] = evaluation_metrics(rfr_time_model)
```

```
Mean Squared Error = 68.51278588663261
Mean Absolute Error = 5.57226306122445
Root Mean Squared Error = 8.277245066242306
r2 score = 0.9986401761659791
```



In [64]:
```
del results['Linear Regression']
del results['Decision Tree Regressor']
del results['XGB Regressor']
del results['LGBM Regressor']
```

Comparing accuracy of predicted model and forecasted model

In [65]:
```
CompareRFR = pd.DataFrame(results)
CompareRFR
```

Out[65]:

	Random Forest Regressor	Random Forest Evaluation Regressor
Training Score	0.999789	0.999805
Testing Score	0.998607	0.998640

What If !!! We take the whole data (1979 - 2023) : Will it be better forcasting?

In [66]:
```
dataIndiaAll = data[['Date','Year','Month','Week','Day','Indian rupee']]
dataIndiaAll = dataIndiaAll.dropna().reset_index(drop=True)
dataIndiaAll.head()
```

Out[66]:

	Date	Year	Month	Week	Day	Indian rupee
0	1979-01-02	1979	1	1	2	14.66
1	1979-01-03	1979	1	1	3	14.73
2	1979-01-04	1979	1	1	4	14.81
3	1979-01-05	1979	1	1	5	15.07
4	1979-01-08	1979	1	2	8	15.06

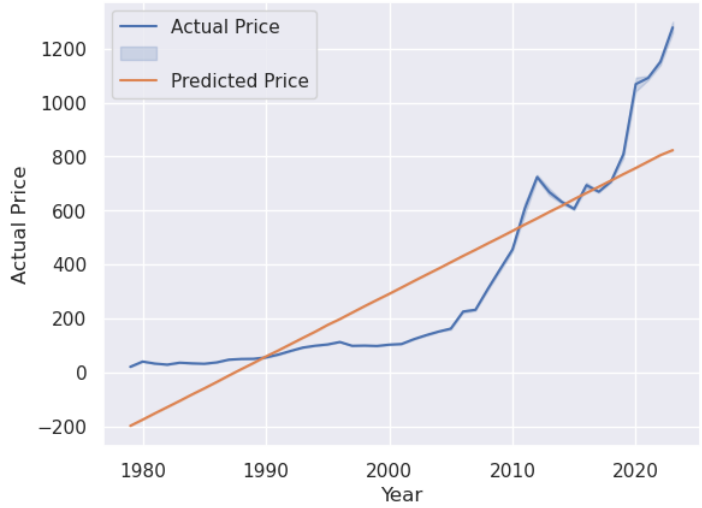
Model selection for whole data

In [67]:
```
x_all = dataIndiaAll[['Year','Month','Week','Day']]
y_all = dataIndiaAll[['Indian rupee']]
x_train,x_test,y_train,y_test = train_test_split(x_all,y_all,test_size = 0.2, random_state = 99)
```

In [68]:
```
AllDataResults = {}
```

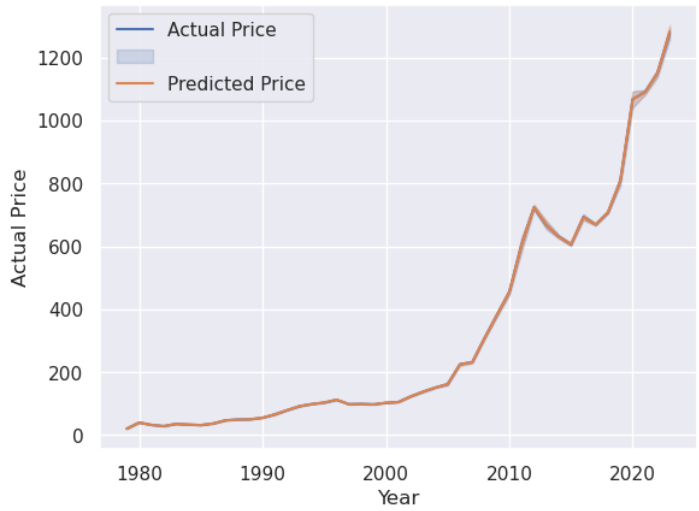
In [69]:
```
AllDataResults['Linear Regression'] = evaluation_metrics(model)
```

```
Mean Squared Error = 26197.346839378588
Mean Absolute Error = 129.9110398940336
Root Mean Squared Error = 161.85594471436193
r2 score = 0.7648229167312672
```



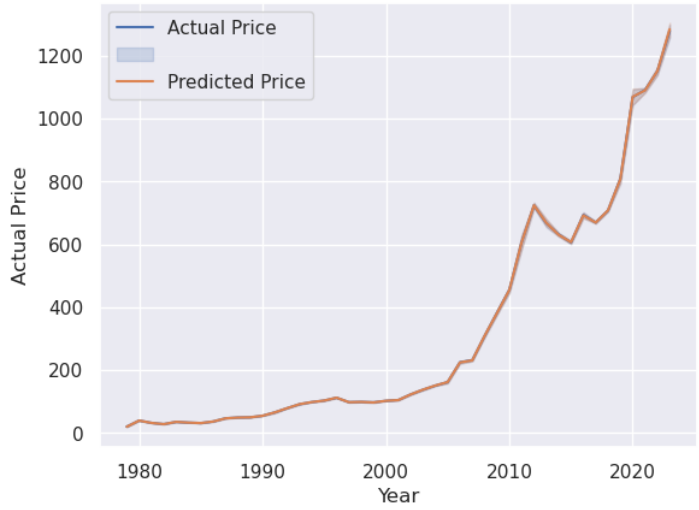
In [70]:
```
AllDataResults['Decision Tree Regressor'] = evaluation_metrics(dtr)
```

```
Mean Squared Error = 20.869293911917087
Mean Absolute Error = 2.3210060449050083
Root Mean Squared Error = 4.568292231448979
r2 score = 0.9998126535598366
```



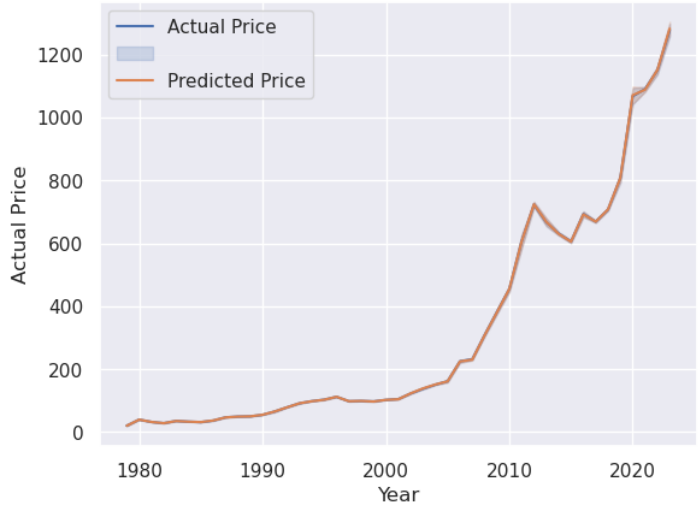
In [71]:
```
AllDataResults['Random Forest Regressor'] = evaluation_metrics(rfr)
```

```
Mean Squared Error = 17.457940534645775
Mean Absolute Error = 2.156791796200346
Root Mean Squared Error = 4.178270040895606
r2 score = 0.9998432777349557
```



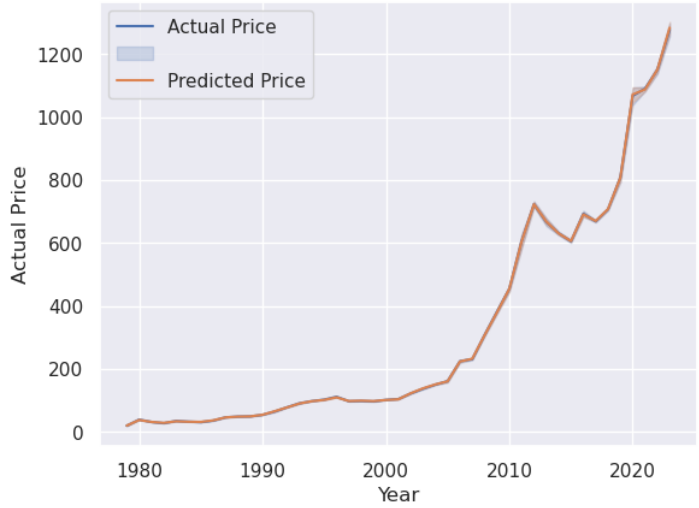
In [72]:
```
AllDataResults['XGB Regressor'] = evaluation_metrics(xgb)
```

```
Mean Squared Error = 27.853747351798226
Mean Absolute Error = 3.096116317656785
Root Mean Squared Error = 5.2776649525901345
r2 score = 0.9997499531879902
```



In [73]:
```
AllDataResults['LGBM Regressor'] = evaluation_metrics(lgbm)
```

```
Mean Squared Error = 53.66628227226071
Mean Absolute Error = 4.6011194030607
Root Mean Squared Error = 7.325727422738352
r2 score = 0.9995182306127391
```



In [74]:
```
AllDataResultsdf = pd.DataFrame(AllDataResults)
```

```
AllDataResultsdf
```

Out[74]:

	Linear Regression	Decision Tree Regressor	Random Forest Regressor	XGB Regressor	LGBM Regressor
Training Score	0.767883	1.000000	0.999969	0.999885	0.999539
Testing Score	0.764823	0.999813	0.999843	0.999750	0.999518

> - Random Forest Regressor is performimg best, but LGBM Regressor is performing very accurate to 4 decimals.

Forecasting whole data

In [75]:
```
time_data_all = dataIndiaAll[['Date','Indian rupee']]
time_data_all.columns = ['ds','y']
time_data_all.head()
```

Out[75]:

	ds	y
0	1979-01-02	14.66
1	1979-01-03	14.73
2	1979-01-04	14.81
3	1979-01-05	15.07
4	1979-01-08	15.06

In [76]:
```
m3 = Prophet(daily_seasonality=True)
m3.fit(time_data_all)
```

```
14:04:32 - cmdstanpy - INFO - Chain [1] start processing
14:04:45 - cmdstanpy - INFO - Chain [1] done processing
```

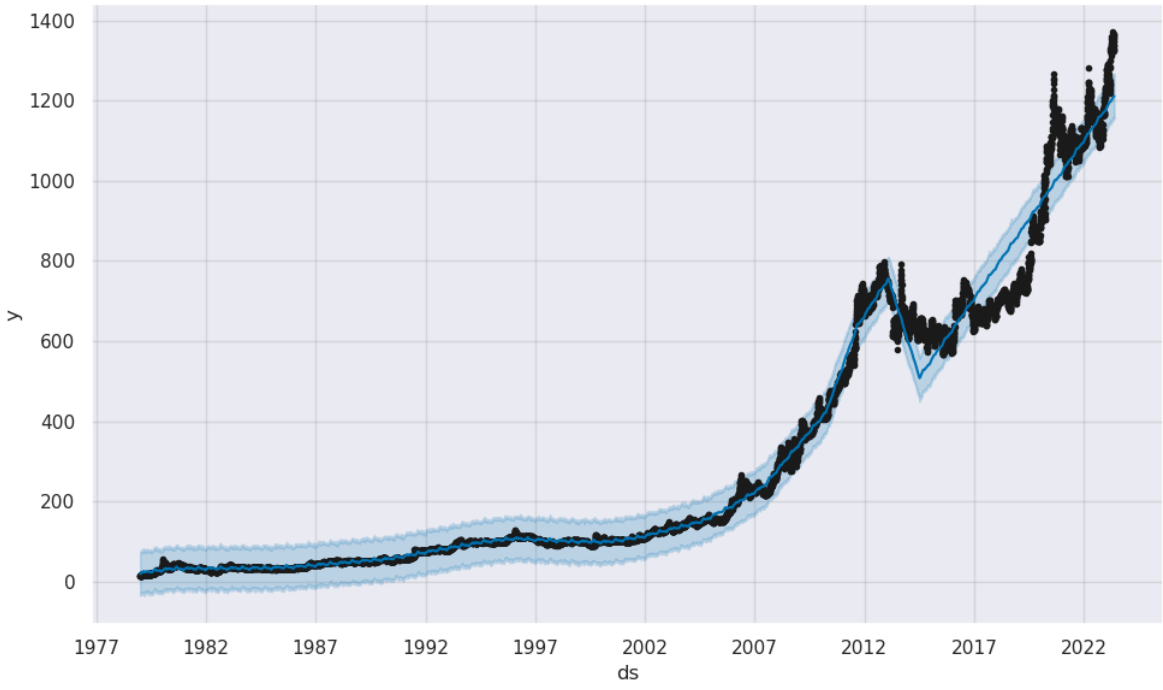
Out[76]:
```
<prophet.forecaster.Prophet at 0x7e051132c650>
```

In [77]:
```
forecast3 = m3.predict(time_data_all)
forecast3[['ds','yhat','yhat_upper','yhat_lower']].tail()
```

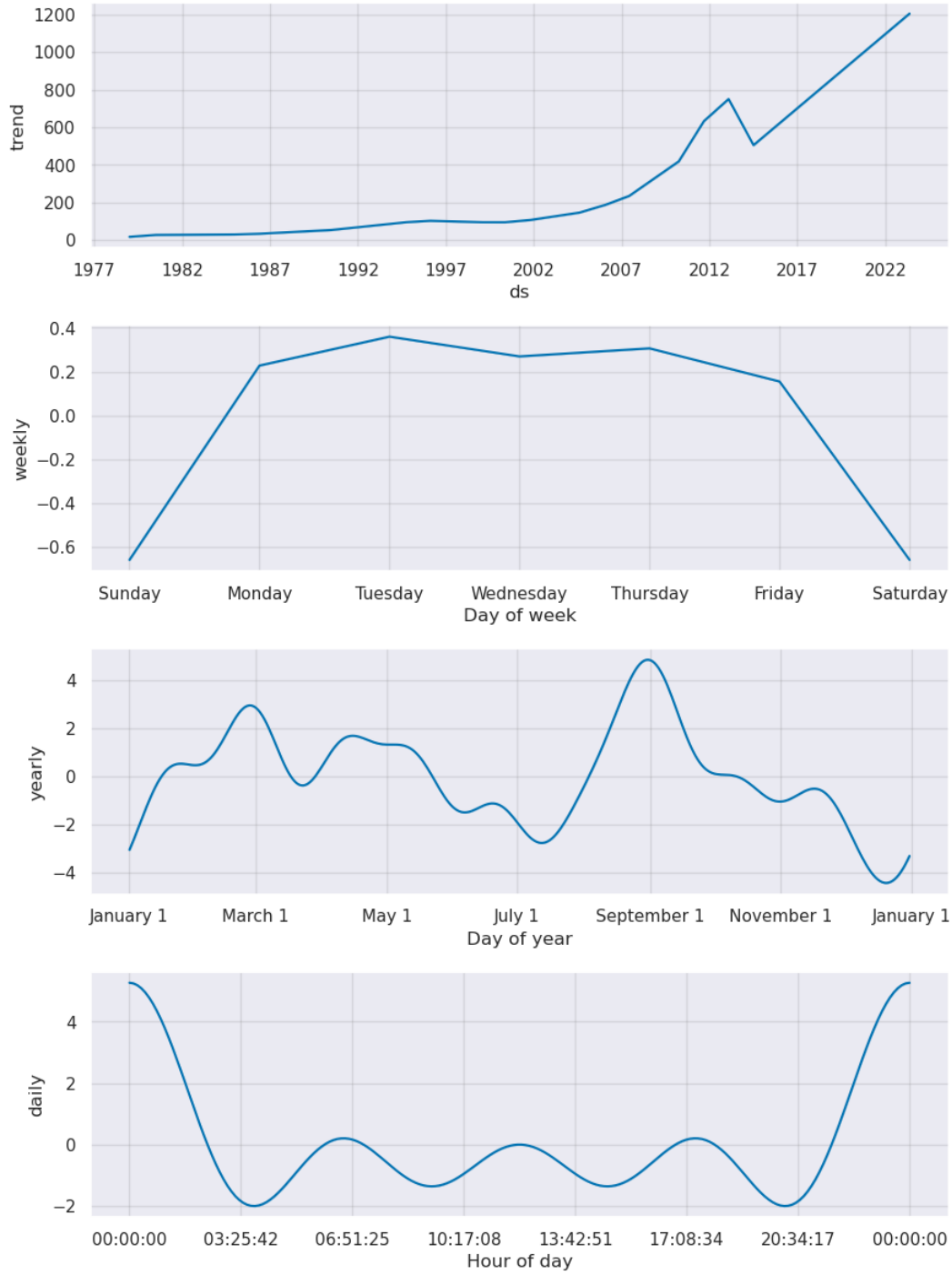
Out[77]:

	ds	yhat	yhat_upper	yhat_lower
11574	2023-05-15	1211.003923	1261.959662	1156.653545
11575	2023-05-16	1211.259981	1268.571187	1158.375886
11576	2023-05-17	1211.281962	1261.144123	1158.656924
11577	2023-05-18	1211.419061	1261.241815	1156.857749
11578	2023-05-19	1211.357644	1263.998107	1157.240909

In [78]:
```
m3.plot(forecast3);
```



In [79]:
```
m3.plot_components(forecast3);
```



This is different: April, June, and July are best month here

In [80]:
```
m4 = Prophet(daily_seasonality=True)
m4.fit(time_data_all)
future2 = m4.make_future_dataframe(periods = 336)
future2.tail(10)
```

```
14:04:54 - cmdstanpy - INFO - Chain [1] start processing
14:05:08 - cmdstanpy - INFO - Chain [1] done processing
```

Out[80]:

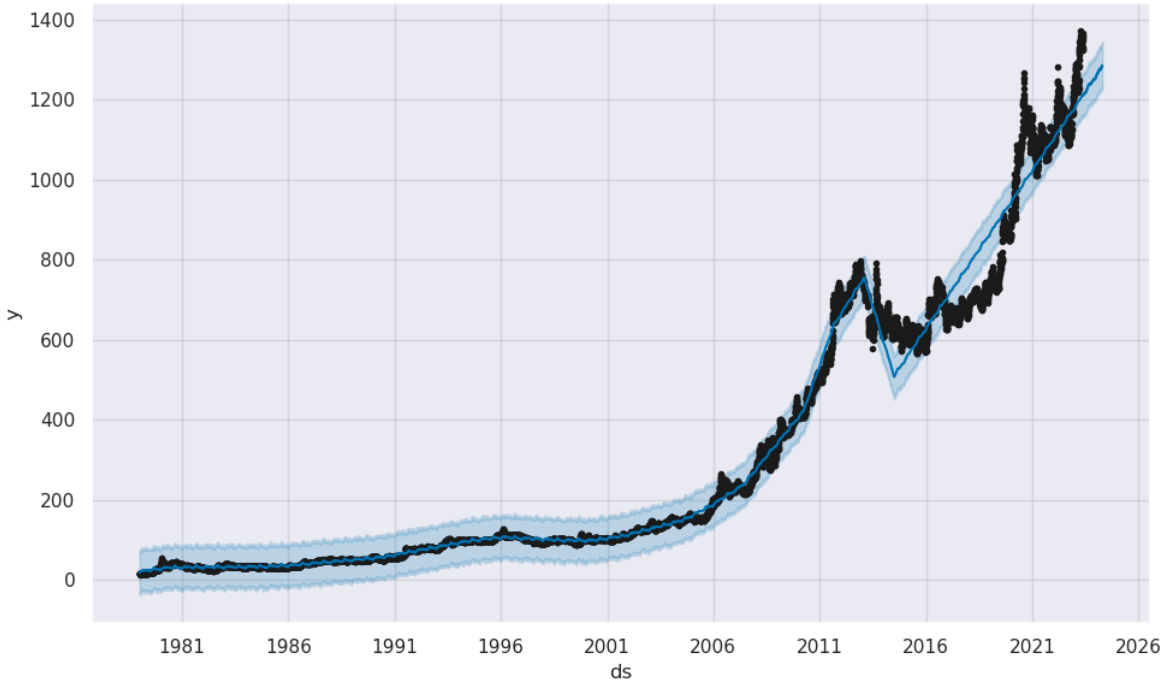
	ds
11905	2024-04-10
11906	2024-04-11
11907	2024-04-12
11908	2024-04-13
11909	2024-04-14
11910	2024-04-15
11911	2024-04-16
11912	2024-04-17
11913	2024-04-18
11914	2024-04-19

In [81]:
```
forecast4 = m4.predict(future2)
forecast4[['ds','yhat','yhat_upper','yhat_lower']].tail()
```

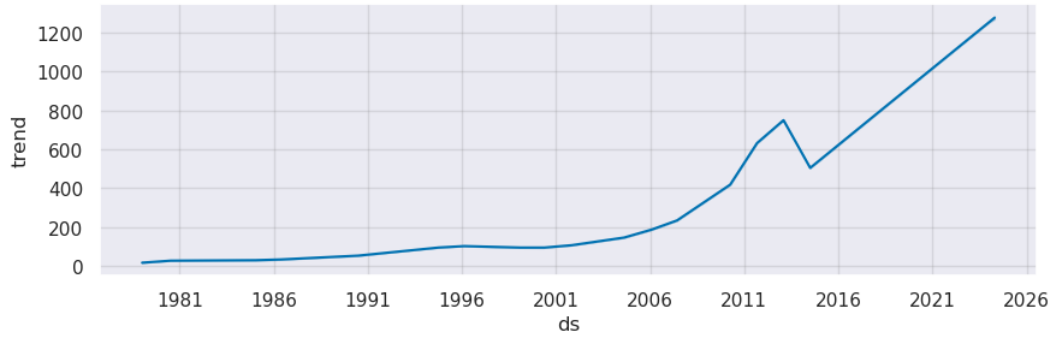
Out[81]:

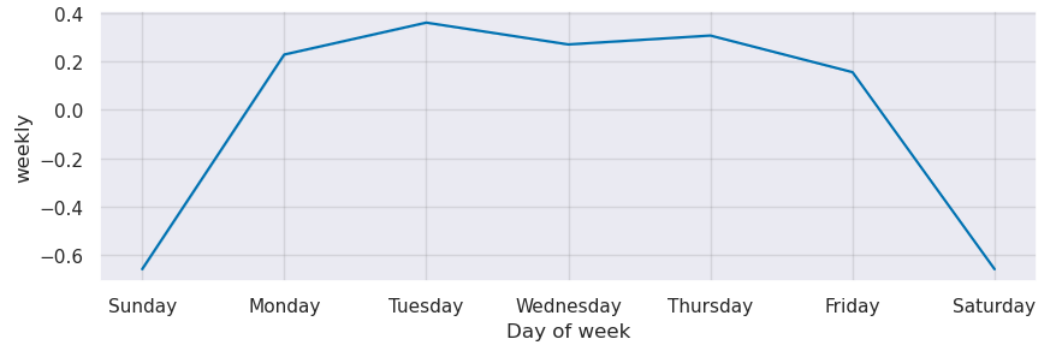
	ds	yhat	yhat_upper	yhat_lower
11910	2024-04-15	1284.354958	1341.617513	1232.523239
11911	2024-04-16	1284.696916	1343.013181	1227.940609
11912	2024-04-17	1284.807629	1344.774432	1229.777855
11913	2024-04-18	1285.037346	1343.159418	1227.441624
11914	2024-04-19	1285.073246	1343.482370	1231.961452

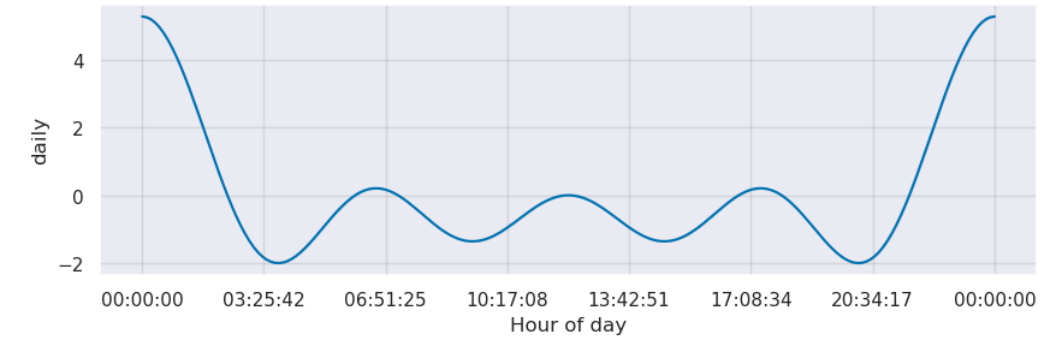
In [82]:
```
m4.plot(forecast4);
```



In [83]:
```
m4.plot_components(forecast4);
```





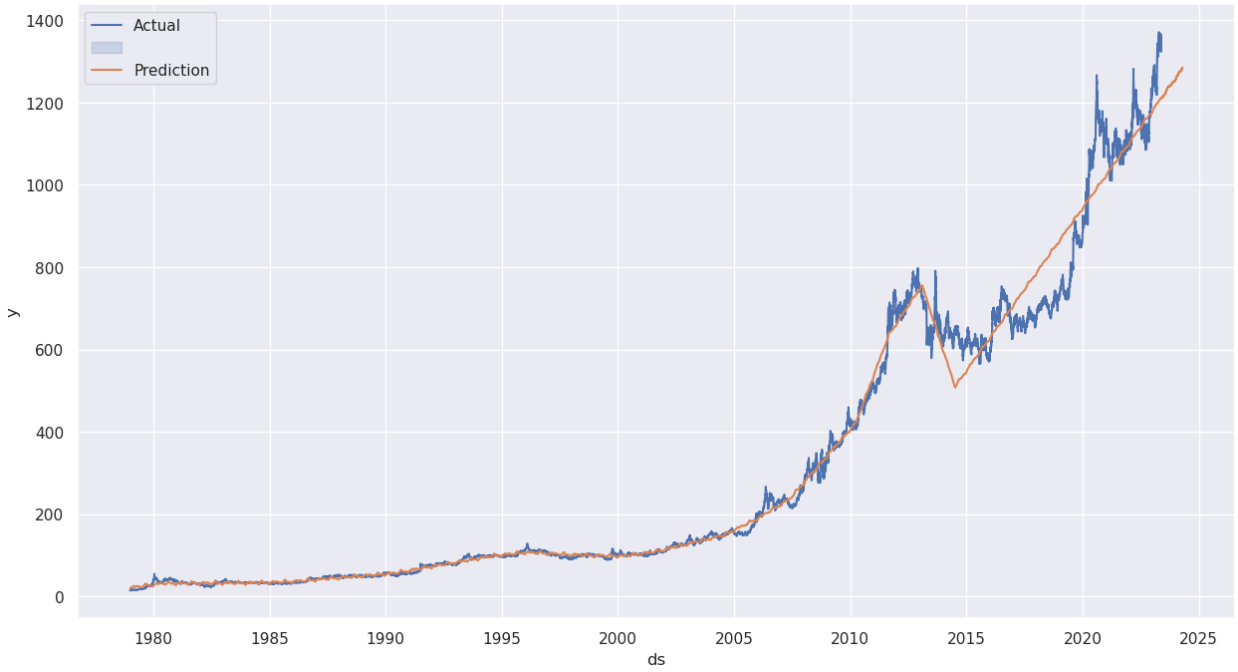


In [84]:
```
predictions2 = forecast4[['ds','yhat']]
predictions2 = predictions2.rename(columns = {'ds':'Date','yhat':'Indian rupee'})
predictions2.head()
```

Out[84]:

	Date	Indian rupee
0	1979-01-02	18.571801
1	1979-01-03	18.717701
2	1979-01-04	18.997408
3	1979-01-05	19.094261
4	1979-01-08	19.919275

In [85]:
```
plt.subplots(figsize = (15,8))
sns.lineplot(time_data_all,x = 'ds',y='y',legend='full')
sns.lineplot(predictions2,x = 'Date',y = 'Indian rupee',legend='full')
plt.legend(labels = ['Actual','','Prediction'])
plt.show()
```



Doesn't looks good

In [86]:
```
predictions2['Date'] = pd.to_datetime(predictions2['Date'])
predictions2['Year'] = predictions2['Date'].dt.year
predictions2['Month'] = predictions2['Date'].dt.month
predictions2['Week'] = predictions2['Date'].dt.week
predictions2['Day'] = predictions2['Date'].dt.day
```

Model Evaluation for whole data

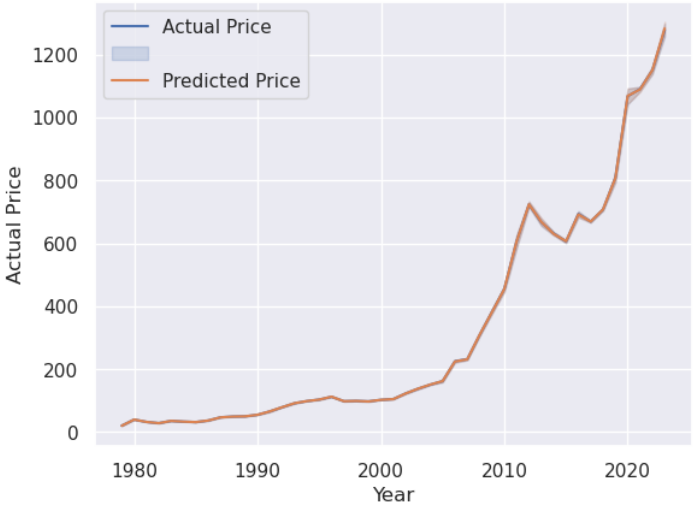
In [87]:
```
x_time_all  = predictions2.drop(columns=['Date','Indian rupee'])
y_time_all = predictions2[['Indian rupee']]
```

In [88]:
```
x_time_train,x_time_test,y_time_train,y_time_test = train_test_split(x_time_all,y_time_all,test_size=0.2,random_state=99)
```

In [89]:
```
rfr_time_all_model = RandomForestRegressor()
rfr_time_all_model.fit(x_time_train,y_time_train)
rfr_time_all_pred = rfr_time_all_model.predict(x_time_test)
```

In [90]:
```
AllDataResults['Random Forest Evaluation Regressor'] = evaluation_metrics(rfr_time_all_model)
```

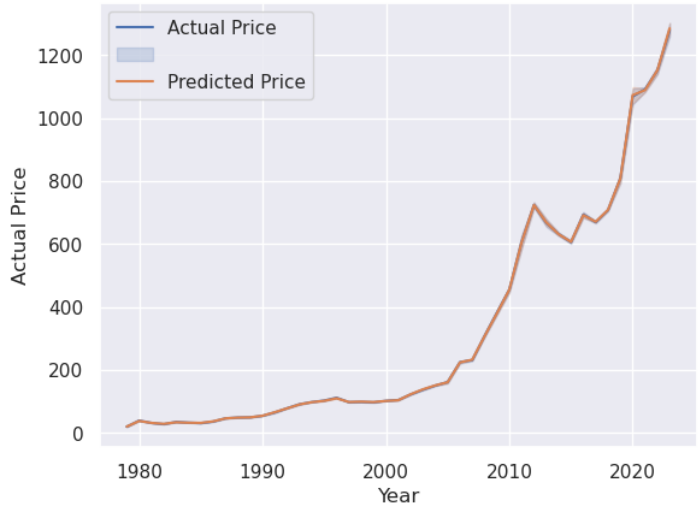
```
Mean Squared Error = 17.095929741446355
Mean Absolute Error = 2.142367227979272
Root Mean Squared Error = 4.134722450352182
r2 score = 0.999846527554221
```



In [91]:
```
lgbm_time_all_model = LGBMRegressor()
lgbm_time_all_model.fit(x_time_train,y_time_train)
lgbm_time_all_pred = lgbm_time_all_model.predict(x_time_test)
```

In [92]:
```
AllDataResults['LGBM Evaluation Regressor'] = evaluation_metrics(lgbm_time_all_model)
```

```
Mean Squared Error = 53.66628227226071
Mean Absolute Error = 4.6011194030607
Root Mean Squared Error = 7.325727422738352
r2 score = 0.9995182306127391
```



In [93]:
```
del AllDataResults['Linear Regression']
del AllDataResults['Decision Tree Regressor']
del AllDataResults['XGB Regressor']
```

Comparing selected and predicted models for the whole data

In [94]:
```
CompareAll = pd.DataFrame(AllDataResults)
CompareAll
```

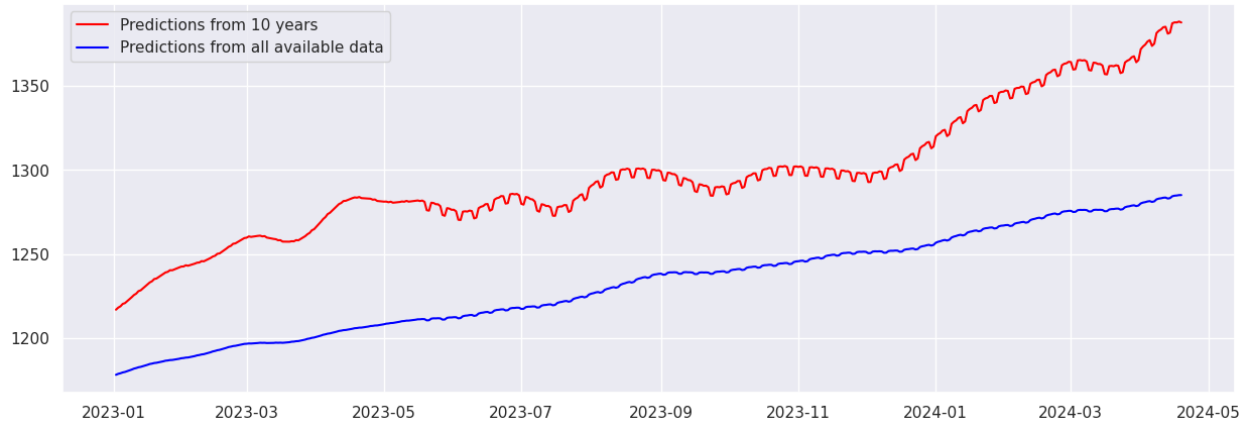
Out[94]:

	Random Forest Regressor	LGBM Regressor	Random Forest Evaluation Regressor	LGBM Evaluation Regressor
Training Score	0.999969	0.999539	0.999970	0.999539
Testing Score	0.999843	0.999518	0.999847	0.999518

> - LGBM Regressor model is very very accurate as it was before the forecasting, but Random Forest Regressor has the highest accuracy

Looking the difference between 10 years prediction and the whole data prediction

In [95]:
```
analysis1 = predictions[predictions['Year']>2022]
analysis2 = predictions2[predictions2['Year']>2022]

plt.subplots(figsize = (15,5))
plt.plot(analysis1['Date'],analysis1['Indian rupee'],color = 'red',label = 'Predictions from 10 years')
plt.plot(analysis2['Date'],analysis2['Indian rupee'],color = 'blue',label = 'Predictions from all available data')
plt.legend()
plt.show()
```



Conclusion:

1. Always large data doesn't forecast good, however model accuracy is pretty good as in LGBM Regressor.
2. The common month of low gold price in both the forecasting is June.
3. Compared to the monthly forecast of the 10 years past data, first forecasting results are good. But, the second forecasting results does not match any of the trends of the whole data forecasting components plot. Clearly, the whole data did not forecasted well.
4. The real way of evaluating this forecasting to see the gold price in June and August. Exciting!!!