PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

UE20CS312 - Data Analytics - Worksheet 3b - AR and MA models

Ishita Bharadwaj - PES1UG20CS648

Collaborated with Hita Juneja - PES1UG20CS645

AR and MA models

Auto Regressive and Moving Average are some of the most powerful, yet simple models for time-series forecasting. They can be used individually or together as ARMA. There are many other variations as well. We will use these models to forecast time-series in this worksheet

Task

Cryptocurrency is all the rage now and it uses the very exciting technology behind blockchain. People even claim it to be revolutionary. But if you have invested in cryptocurrencies, you know how volatile these cryptocurrencies really are! People have become billionaires by investing in crypto, and others have lost all their money on crypto. The most recent instance of this volatility was seen during the Terra Luna crash. Find more info about that here and here if you are interested.

Your task is to effectively forecast the prices of **DogeCoin**, a crypto that started as a meme but now is a crypto that people actually invest in. DogeCoin prices however, are affected even by a single tweet by Elon Musk. The image below tweeted by Elon Musk shot up the prices of DogeCoin by 200%!

You have been provided with the daily prices of DogeCoin from `15-08-2021` to `15-08-2022` a period of 1 year (365 days) in the file `doge.csv`

Please download the data from this Github repo

Data Dictionary

Date - Date on which price was recorded
Price - Price of DogeCoin on a particular day

Data Ingestion and Preprocessing

- Read the file into a `Pandas` DataFrame object

```
import pandas as pd
df = pd.read_csv('doge.csv')
```

```
df.head()
```

```
         Date     Price
0  2021-08-15  0.348722
1  2021-08-16  0.349838
2  2021-08-17  0.345208
3  2021-08-18  0.331844
4  2021-08-19  0.321622
```

Prerequisites

- Set up a new conda env or use an existing one that has `jupyter-notebook` and `ipykernel` installed (Conda envs come with these by default) Reference
- Instead, you can also use a python venv and install `ipykernel` manually (We instead suggest using conda instead for easy setup) Reference
- Install the `statsmodels` package either in your Conda environment or Python venv. Refer to the installation guide

```
!pip install ipykernel
```

```
!pip install statsmodels
```

Points

The problems in this worksheet are for a total of 10 points with each problem having a different weightage.

- Problem 0: 0.5 points
- Problem 1: 1.5 point
- Problem 2: 2 points
- Problem 3: 1 points
- Problem 4: 2 point
- Problem 5: 1 point
- Problem 6: 1 points

HINTS FOR ALL PROBLEMS:

- Consider using `inplace=True` or assign it to new DataFrame, when using pandas transformations. If none of these are done, the DataFrame will remain the same
- Search for functions in the `statsmodels` documentation

Problem 0 (0.5 point)

- Set the index of DataFrame to the `Date` column to make it a time series

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

```
               Price
Date
2021-08-15  0.348722
```

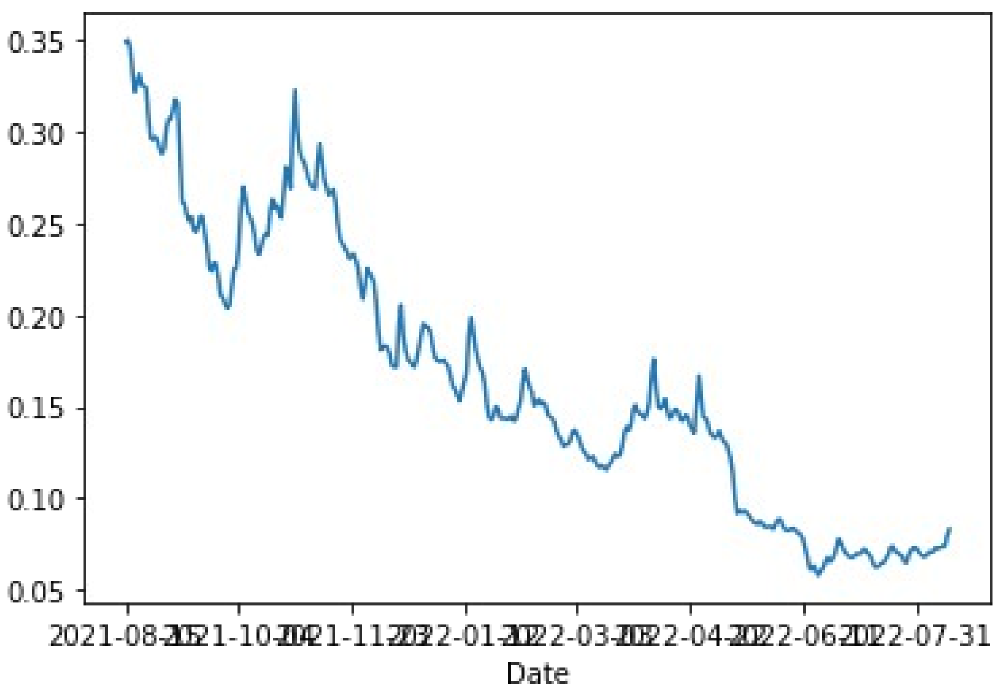
```
2021-08-16  0.349838
2021-08-17  0.345208
2021-08-18  0.331844
2021-08-19  0.321622
...              ...
2022-08-11  0.072978
2022-08-12  0.073563
2022-08-13  0.073670
2022-08-14  0.079436
2022-08-15  0.082882

[366 rows x 1 columns]
```

Problem 1 (1.5 point)

- Plot the time-series. Analyze the stationarity from the time-series. Provide reasoning for stationarity/non-stationarity based on visual inspection of time-series plot (0.5 point)
- Plot the ACF plot of the Time series (upto 50 lags). Analyze the stationarity from ACF plot and provide reasoning (Hint: look at functions in `statsmodels` package) (1 Point)

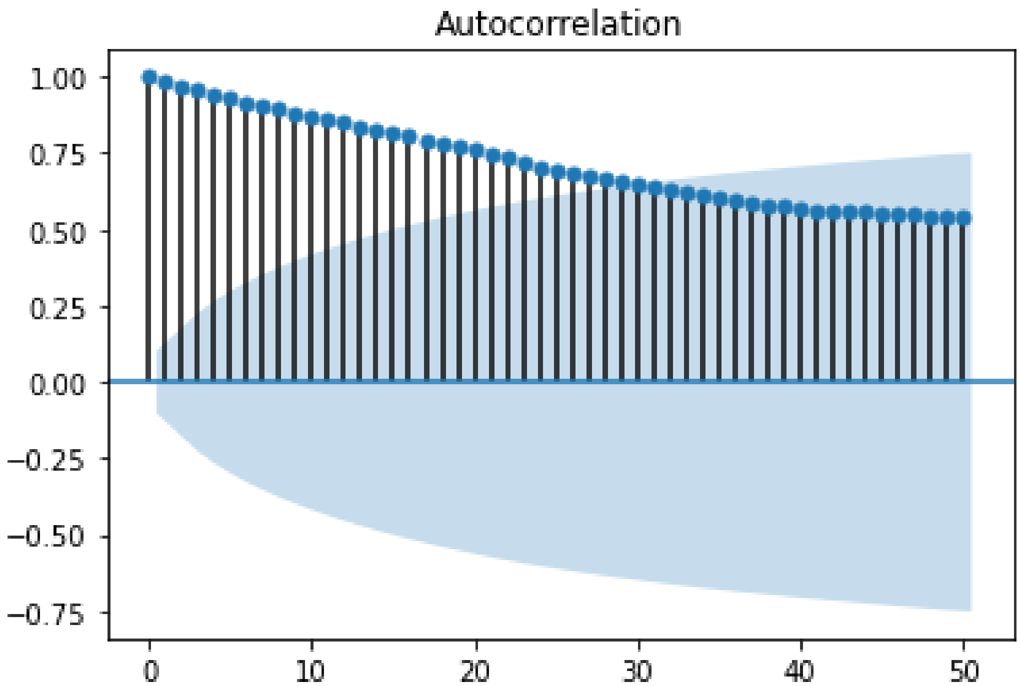
```
df['Price'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28fa353650>
```




```
#Plotting the ACF
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

```
sm.graphics.tsa.plot_acf(df.values.squeeze(), lags=50)
plt.show()
```



The time series plot for the given dataset doesn't follow the rules of stationarity. That is 1) The means values of Price is at different values of time (date) are not constant. 2) The variances of Price on different time periods(dates) are not constant. 3) The covariances in prices at time t and t-k depends on both t and k. Also, a gradual decrease in ACF, rather than a sharp decrease/ cut off to 0 is another sign of nonstationarity. Hence, the given time series is non-stationary.

Problem 2 (2 point)

- Run Augmented Dickey Fuller Test. Analyze whether the time-series is stationary, based on ADF results (1 point)

Hint: Use the `print_adf_results` function below to print the results of the ADF function cleanly after obtaining results from the library function. Pass the results from library function to `print_adf_results` function

```
def print_adf_results(adf_result):
    print('ADF Statistic: %f' % adf_result[0])
    print('p-value: %f' % adf_result[1])
    print('Critical Values:')
    for key, value in adf_result[4].items():
        print('\t%s: %.3f' % (key, value))
```

- If not stationary, apply appropriate transformations. Run the ADF test again to show stationarity after transformation (1 Point)

Hint: `diff` and `dropna`. Assign the DataFrame after transformation to a new DataFrame with name `transformed_df`

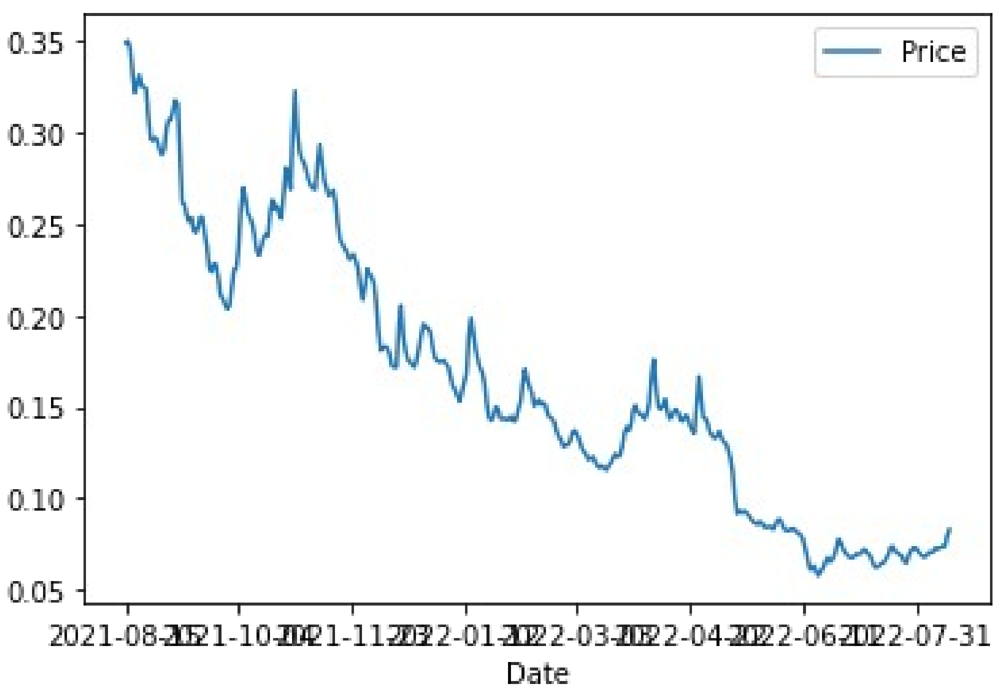
Here the null and alternate hypothesis are as follows: H0 = Time series is non-stationary. Ha = Time series is stationary.

```
from statsmodels.tsa.stattools import adfuller
adf_result = adfuller(df)
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
print('Critical Values:')
for key, value in adf_result[4].items():
  print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -1.558935
p-value: 0.504182
Critical Values:
	1%: -3.449
	5%: -2.870
	10%: -2.571
```

```
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28fa2aecd0>
```



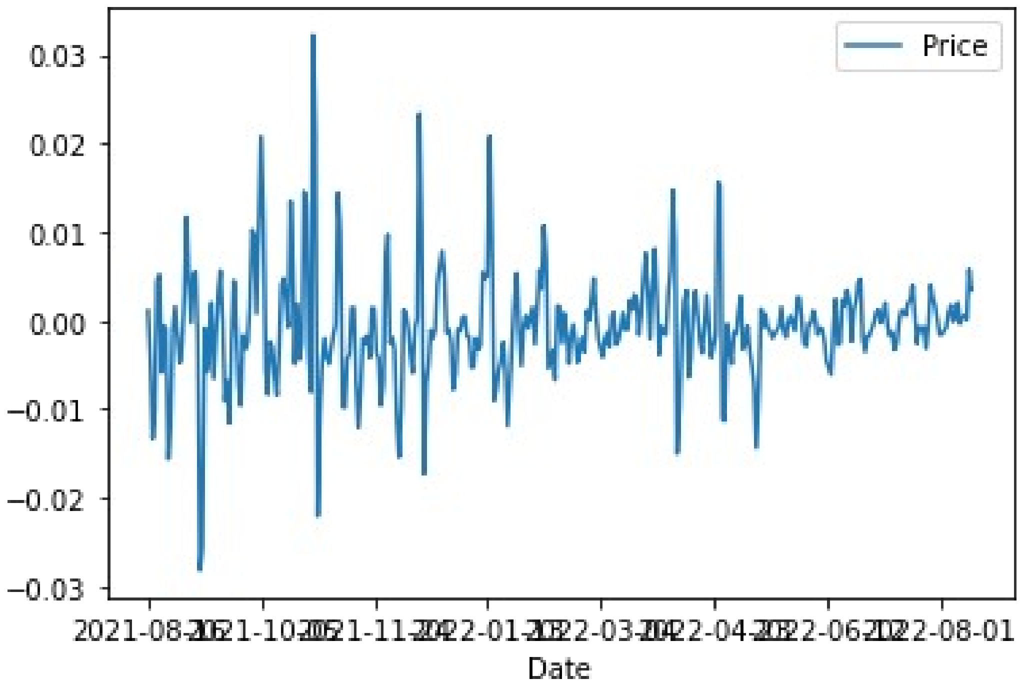
The ADF statistics for the original dataframe are as give above. A p-value = 0.5 is obtained which signifies fail to reject H0. Even the plot shows a non-stationary/varying wave. Thus, we can conclude that the time series is non-stationary.

```
transformed_df = df.diff().dropna()
trans_adf = adfuller(transformed_df)
print_adf_results(trans_adf)
```

```
ADF Statistic: -5.593446
p-value: 0.000001
Critical Values:
	1%: -3.449
	5%: -2.870
	10%: -2.571
```

```
transformed_df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28fa238410>
```



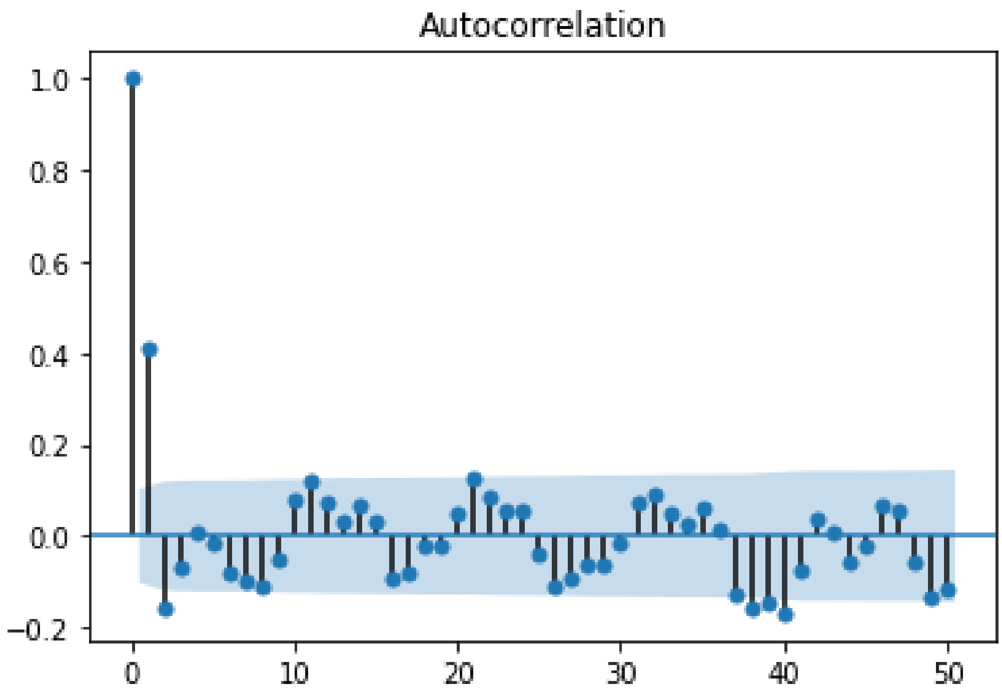
After applying the transformation of a single order difference and dropping all NA values, we obtain a new dataframe called transformed_df. The ADF statistics and the p-value(0.000001) indicate rejection of Ho and conclude that the new time series is stationary.

Problem 3 (1 point)

- Plot both ACF and PACF plot. From these select optimal parameters for the ARIMA(p,q) model

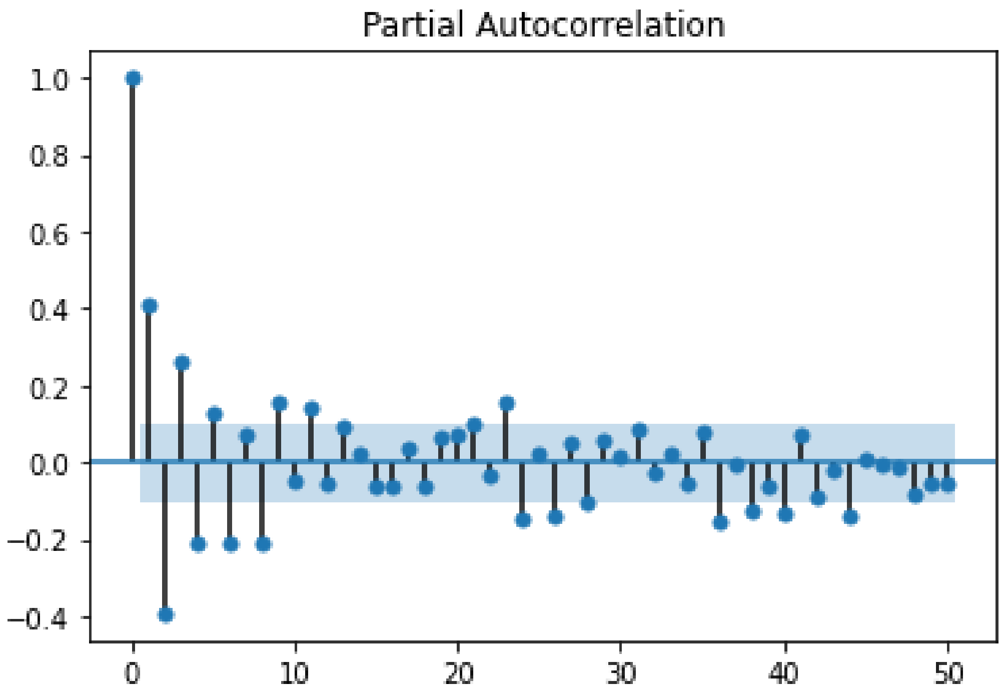
Hint: Negative values that are significantly outside the Confidence interval are considered significant too.
Hint: `p+q = 3`

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
sm.graphics.tsa.plot_acf(transformed_df, lags=50)
plt.show()
```



As seen from the ACF plot, there are 2 values that are significantly outside the confidence interval compared to all of the other values.Also the auto-correlation function decreases exponentially after 2 instances. Hence value of p can be chosen to be 2.

```
sm.graphics.tsa.plot_pacf(transformed_df, lags=50)
plt.show()
```



As seen from the above PACF plot, the partial auto-correlation is >1.96/sqrt(n) for the order 1. Thus, the value for q shd be 1.

Problem 4 (2 point)

- Write a function to forecast values using only AR(p) model (2 Points)
  Only use `pandas` functions and Linear Regression from `sklearn`. LR documentation

Hint: Create p new columns in a new DataFrame that is a copy of `transformed_df`
Each new column has lagged value of Price. `Price_t-i` (From `Price_t-1` upto `Price_t-p`)
Look at the `shift` function in pandas to create these new columns Link

```
### Adding columns for lagged values
arima_df = transformed_df.copy()

## AR terms
p = 2
# Creating p new columns, for p lagged values
for i in range(1,p+1):
    arima_df[f'Price_t-{i}'] = arima_df['Price'].shift(i)

arima_df.dropna(inplace=True)

arima_df
```

```
               Price  Price_t-1  Price_t-2
Date
2021-08-18 -0.013363  -0.004630   0.001116
```

```
2021-08-19 -0.010222  -0.013363  -0.004630
2021-08-20  0.004498  -0.010222  -0.013363
2021-08-21  0.005169   0.004498  -0.010222
2021-08-22 -0.005841   0.005169   0.004498
...              ...        ...        ...
2022-08-11  0.000380  -0.000374   0.001993
2022-08-12  0.000585   0.000380  -0.000374
2022-08-13  0.000107   0.000585   0.000380
2022-08-14  0.005767   0.000107   0.000585
2022-08-15  0.003446   0.005767   0.000107

[363 rows x 3 columns]
```

Hint:

- **Seperate into `X_train` and `y_train` for linear regression**

- We know that AR(p) is linear regression with p lagged values, and we have created p new columns with the p lagged values

- `X_train` is training input that consists of the columns `Price_t-1` upto `Price_t-p` (p columns in total)
- `y_train` is the training output (truth values) of the Price, i.e the `Price` column (Only 1 column)

```
X_train = arima_df[['Price_t-1', 'Price_t-2']].values
y_train = arima_df['Price'].values
```

- Set up the Linear Regression between `X_train` and `y_train` LR documentation

Name the `LinearRegression()` object `lr`

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
lr.coef_
```

```
array([ 0.57292128, -0.39148166])
```

```
# Adding new column with predictions using the LR coefficients. The LR Coefficients are Alpha values or AR coefficients
arima_df['AR_Prediction'] = X_train.dot(lr.coef_.T) + lr.intercept_
```

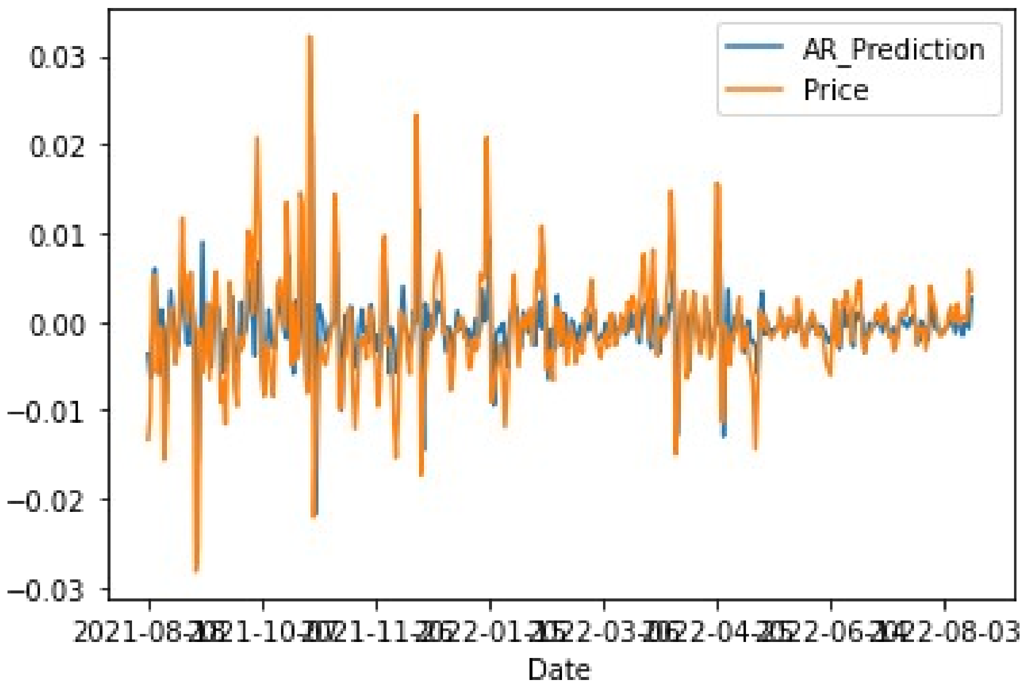
```
arima_df
```

```
               Price  Price_t-1  Price_t-2  AR_Prediction
Date
2021-08-18 -0.013363  -0.004630   0.001116      -0.003682
2021-08-19 -0.010222  -0.013363  -0.004630      -0.006436
```

```
2021-08-20  0.004498  -0.010222  -0.013363      -0.001218
2021-08-21  0.005169   0.004498  -0.010222       0.005986
2021-08-22 -0.005841   0.005169   0.004498       0.000608
...              ...        ...        ...            ...
2022-08-11  0.000380  -0.000374   0.001993      -0.001587
2022-08-12  0.000585   0.000380  -0.000374      -0.000228
2022-08-13  0.000107   0.000585   0.000380      -0.000406
2022-08-14  0.005767   0.000107   0.000585      -0.000760
2022-08-15  0.003446   0.005767   0.000107       0.002669

[363 rows x 4 columns]
```

```
arima_df.plot(y=['AR_Prediction', 'Price'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28fa097990>
```



Once you get predicitons like this using AR you would have to, undifference the predictions (which are differenced), but we will not deal with that here. For some hints on how to undifference the data to get actual predictions look here

Problem 5 (1 Point)

Phew! Just handling AR(2) manually required us to difference, apply regression, undifference. Let's make all of this much easier with a simple library function

- **Use the ARIMA function using parameters picked to forecast values (1 point)**

Hint: Look at ARIMA function the `statstmodels`. Pass the `p,d,q` inferred from the previous tasks

We **DO NOT** need to pass the `transformed_df` to the ARIMA function.
Pass the orirginal `df` as input to ARIMA function, with the `d` value inferred when Transforming the df to make it stationary
The ARIMA function automatically performs the differencing based on the `d` value passed
Store the `.fit()` results in an object named `res`

```
import statsmodels.api as sm
model = sm.tsa.arima.ARIMA(df, order=(2,1,1))
res = model.fit()
res.summary()
```

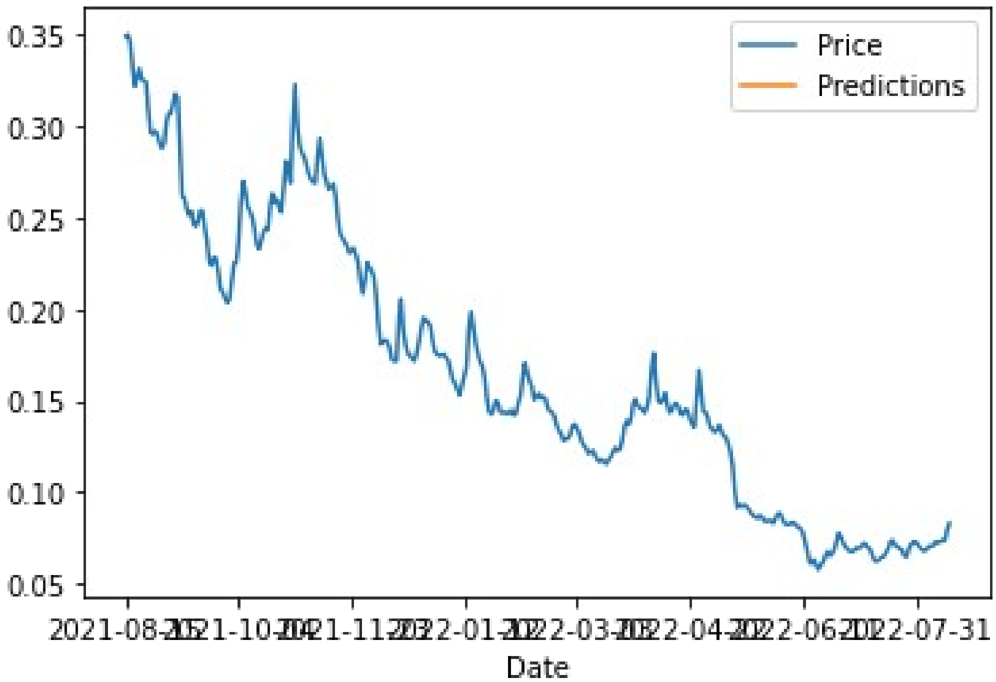
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/
tsa_model.py:527: ValueWarning: No frequency information was provided,
so inferred frequency D will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.
py:527: ValueWarning: No frequency information was provided, so
inferred frequency D will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.
py:527: ValueWarning: No frequency information was provided, so
inferred frequency D will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:568:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
  ConvergenceWarning)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results

======================================================================
========
Dep. Variable:                  Price   No. Observations:
366
Model:                 ARIMA(2, 1, 1)   Log Likelihood
1440.153
Date:                Fri, 07 Oct 2022   AIC                          -
2872.305
Time:                        14:14:00   BIC                          -
2856.706
Sample:                    08-15-2021   HQIC                         -
2866.106
                         - 08-15-2022

Covariance Type:                  opg

======================================================================
========
```

```
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------
--------
ar.L1         -0.0540      0.048     -1.137      0.256      -0.147
0.039
ar.L2         -0.1093      0.052     -2.115      0.034      -0.211
-0.008
ma.L1          0.9396      0.021     43.763      0.000       0.898
0.982
sigma2      2.201e-05   7.48e-07     29.428      0.000    2.05e-05
2.35e-05
========================================================================
=============
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
3047.97
Prob(Q):                              0.91   Prob(JB):
0.00
Heteroskedasticity (H):               0.15   Skew:
1.73
Prob(H) (two-sided):                  0.00   Kurtosis:
16.73
========================================================================
=============

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

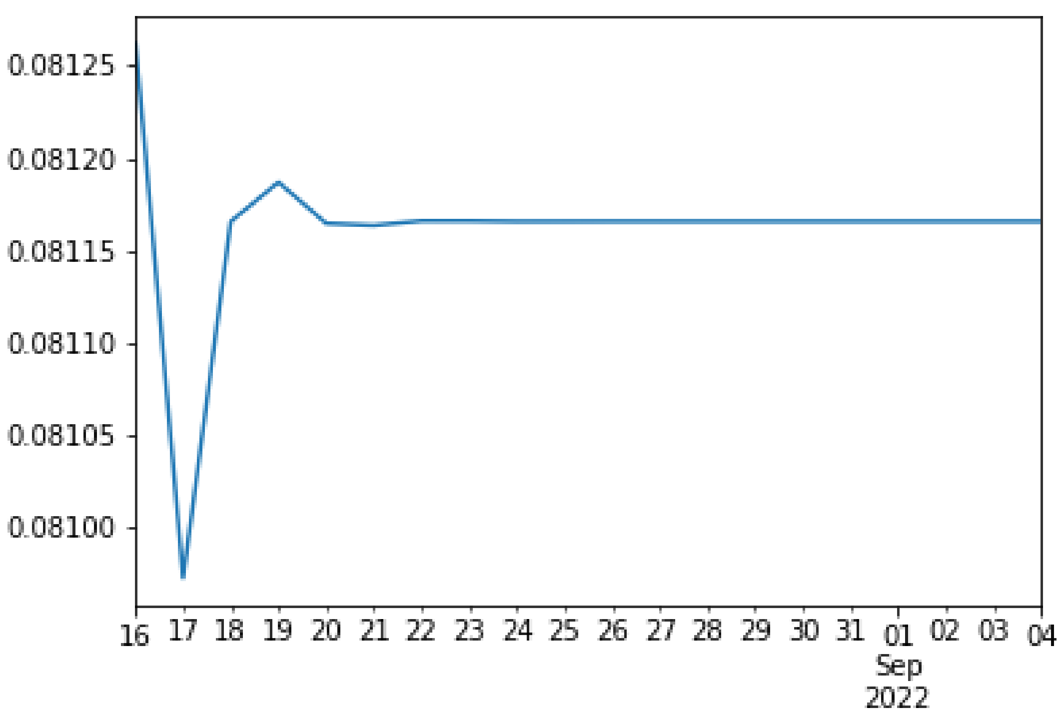
```
## TODO: Use ARIMA function

# # Making predictions and plotting
df['Predictions'] = res.predict(0, len(df)-1)
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28fad6c8d0>
```




```
# Forecast for 20 future dates after training data ends
res.forecast(20).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28f9f73d50>
```



Problem 6 (1 point)

- Evaluate the ARIMA model using Ljung Box test. Based on p-value infer if the Model shows lack of fit

Hint: Pass the `res.resid` (Residuals of the ARIMA model) as input the Ljung-Box Text. Pass `lags=[10]`. Set `return_df=True` For inference, refer back to the Null and Alternate Hypotheses of Ljung-Box test. (If p value high, Null Hypothesis is significant)

```
#import statsmodels.api as sm
sm.stats.acorr_ljungbox(res.resid, lags=[10], return_df=True)
```

```
     lb_stat  lb_pvalue
10  0.929177   0.999877
```