PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

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

Designed by Vishruth Veerendranath, Dept. of CSE - vishruth@pesu.pes.edu

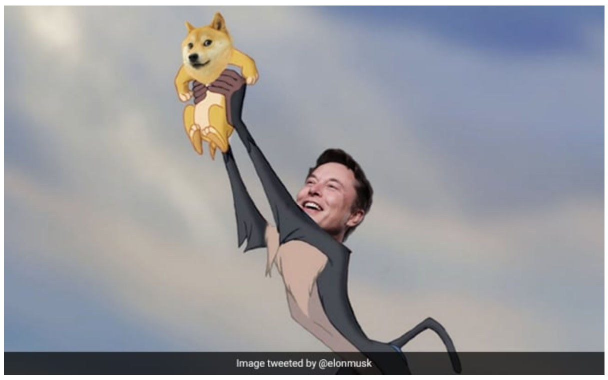
AR and MA models

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 (https://www.forbes.com/sites/lawrencewintermeyer/2022/05/25/from-hero-to-zero-how-terra-was-toppled-in-cryptos-darkest-hour/?sh=5a7e83bf389e) and here (https://c.ndtvimg.com/2021-02/4lo9ita_elon-musk-dogecoin-meme_625x300_04_February_21.jpg) 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 (https://github.com/Data-Analytics-UE20CS312/Unit-3-Worksheets/blob/master/3b%20-%20AR%20and%20MA%20models/doge.csv)

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

In [ ]:

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

df.head()
```

Out[ ]:

	Date	Price
0	2021-08-15	0.348722
1	2021-08-16	0.349838
2	2021-08-17	0.345208
3	2021-08-18	0.331845
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 (https://conda.io/projects/conda/en/latest/user-guide/getting-started.html)
- Instead, you can also use a python venv and install `ipykernel` manually (We instead suggest using conda instead for easy setup) Reference (https://docs.python.org/3/tutorial/venv.html)
- Install the `statsmodels` package either in your Conda environment or Python venv. Refer to the installation guide (https://www.statsmodels.org/dev/install.html)

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: 3 point
- Problem 5: 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 (https://www.statsmodels.org/dev/index.html)

Problem 0 (0.5 point)

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

In [ ]:

```
df
```

Out[ ]:

```
           Date     Price
  0  2021-08-15  0.348722
  1  2021-08-16  0.349838
  2  2021-08-17  0.345208
  3  2021-08-18  0.331845
  4  2021-08-19  0.321622
...         ...       ...
361  2022-08-11  0.072978
362  2022-08-12  0.073563
363  2022-08-13  0.073670
364  2022-08-14  0.079437
365  2022-08-15  0.082882

366 rows × 2 columns
```

```
In [ ]:
df.set_index('Date', inplace=True)
df
```

```
Out[ ]:
```

	Price
Date	
2021-08-15	0.348722
2021-08-16	0.349838
2021-08-17	0.345208
2021-08-18	0.331845
2021-08-19	0.321622
...	...
2022-08-11	0.072978
2022-08-12	0.073563
2022-08-13	0.073670
2022-08-14	0.079437
2022-08-15	0.082882

366 rows × 1 columns

Problem 1 (1.5 point)

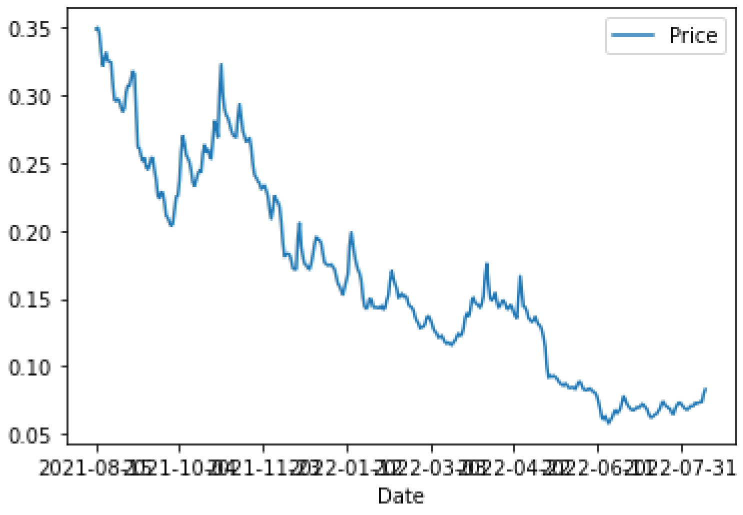
- Plot the time-series. Analyze the stationarity from the time-series. Provide reasoning for stationarity/non-stationarity based on visual inspection of time-series plot (0.5 point)

In [ ]:

```
df.plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feadd484100>
```

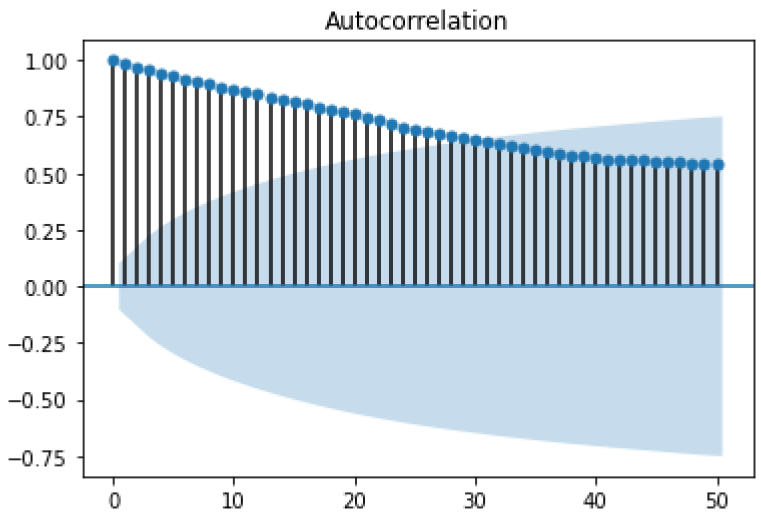


This time-series is non-stationary becasuse there is a **decreasing trend.** Seasonality might be possible too

- 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)

In [ ]:

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



We see that the ACF plot is decreasing very gradually and not exponentially, hence it 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

In [ ]:

```
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))
```

```
In [ ]:
```

```
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
```

p-value is very high 0.199. Hence, the process has unit root or is non-stationary

- 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`

```
In [ ]:
```

```
transformed_df = df.diff().dropna()
```

```
In [ ]:
```

```
trans_adf = adfuller(transformed_df)

print_adf_results(trans_adf)
```

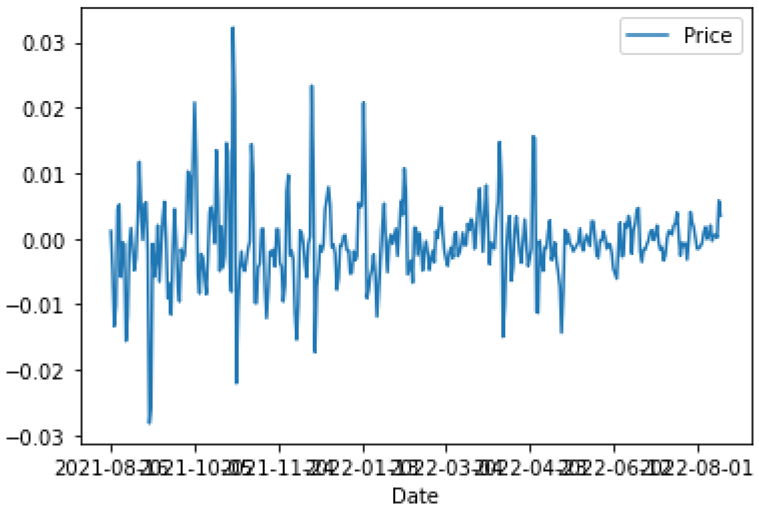
```
ADF Statistic: -5.593446
p-value: 0.000001
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
```

In [ ]:

```
transformed_df.plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feadce0e1c0>
```



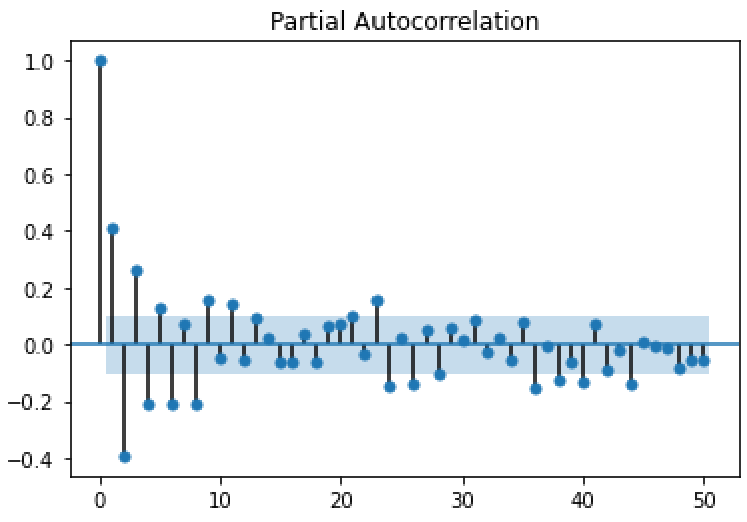
After differencing, the time-series is stationary

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`

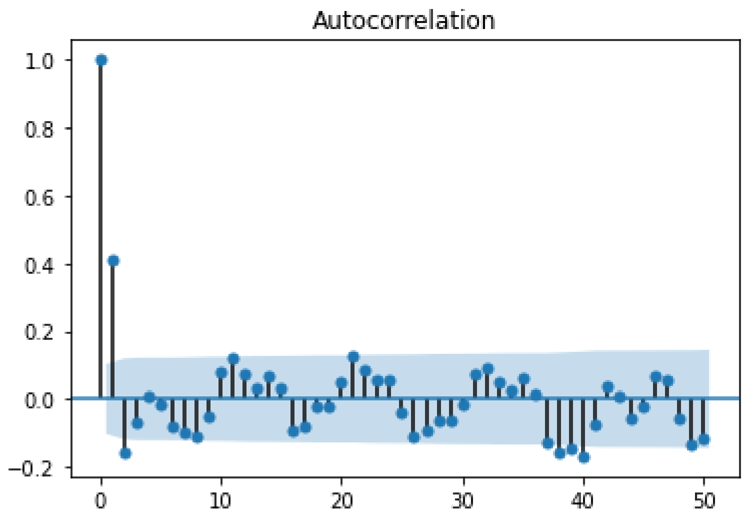
In [ ]:

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



In [ ]:

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



ARIMA(2,1) or ARIMA(2,2) is optimal. We can use either one.

As p+q = 3 is the Hint given. We will go with **ARIMA(2,1)**

Problem 4 (3 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 (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

Hint1: 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 (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html)

In [ ]:

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

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

arima_df.dropna(inplace=True)
```

In [ ]:

```
arima_df
```

Out[ ]:

	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	0.001994
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 × 3 columns

- **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)

In [ ]:

```
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 (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

Name the `LinearRegression()` object `lr`

In [ ]:

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

Out[ ]:

```
LinearRegression()
```

In [ ]:

```
lr.coef_
```

Out[ ]:

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

In [ ]:

```
arima_df['AR_Prediction'] = X_train.dot(lr.coef_.T) + lr.intercept_
```

In [ ]:

```
arima_df
```

Out[ ]:

Date	Price	Price_t-1	Price_t-2	AR_Prediction
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	0.001994	-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

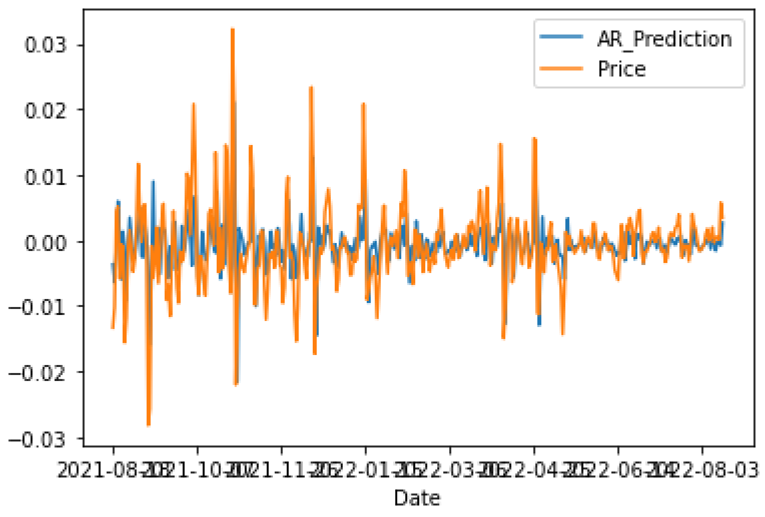
363 rows × 4 columns

In [ ]:

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

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feaddb5a5e0>
```



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 (https://stackoverflow.com/questions/49903037/pandas-reverse-of-diff)](https://stackoverflow.com/questions/49903037/pandas-reverse-of-diff)

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`

In [ ]:

```
import statsmodels.api as sm

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

```
/Users/vish/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ts
a/base/tsa_model.py:159: ValueWarning: No frequency information was
provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/vish/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ts
a/base/tsa_model.py:159: ValueWarning: No frequency information was
provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/vish/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ts
a/base/tsa_model.py:159: ValueWarning: No frequency information was
provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/vish/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ba
se/model.py:567: ConvergenceWarning: Maximum Likelihood optimization
failed to converge. Check mle_retvals
  warn("Maximum Likelihood optimization failed to converge. "
```

Out[ ]:

SARIMAX Results

Dep. Variable:	Price	**No. Observations:**	366
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	1440.333
Date:	Sat, 03 Sep 2022	**AIC**	-2872.667
Time:	12:50:26	**BIC**	-2857.067
Sample:	08-15-2021	**HQIC**	-2866.467
	- 08-15-2022		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-0.0660	0.045	-1.480	0.139	-0.153	0.021
ar.L2	-0.0977	0.052	-1.892	0.058	-0.199	0.004
ma.L1	0.9550	0.017	54.763	0.000	0.921	0.989
sigma2	2.176e-05	7.24e-07	30.042	0.000	2.03e-05	2.32e-05

Ljung-Box (Q):	58.42	**Jarque-Bera (JB):**	3283.10
Prob(Q):	0.03	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.15	**Skew:**	1.78
Prob(H) (two-sided):	0.00	**Kurtosis:**	17.26

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

In [ ]:

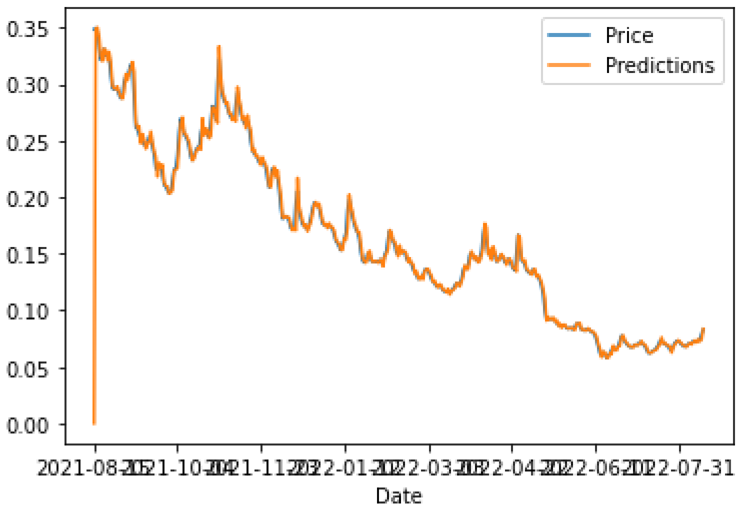
```
df['Predictions'] = res.predict(0, len(df)-1)
```

In [ ]:

```
df.plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feadcd29730>
```

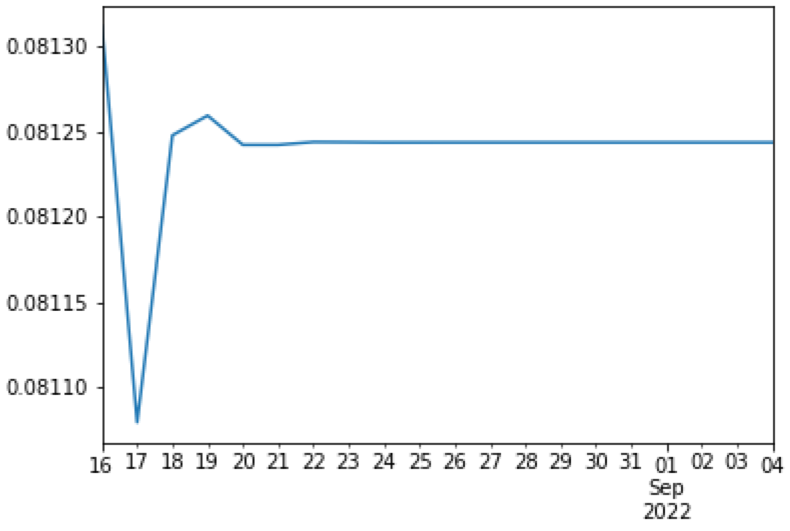


In [ ]:

```
res.forecast(20).plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feaddd0b9a0>
```

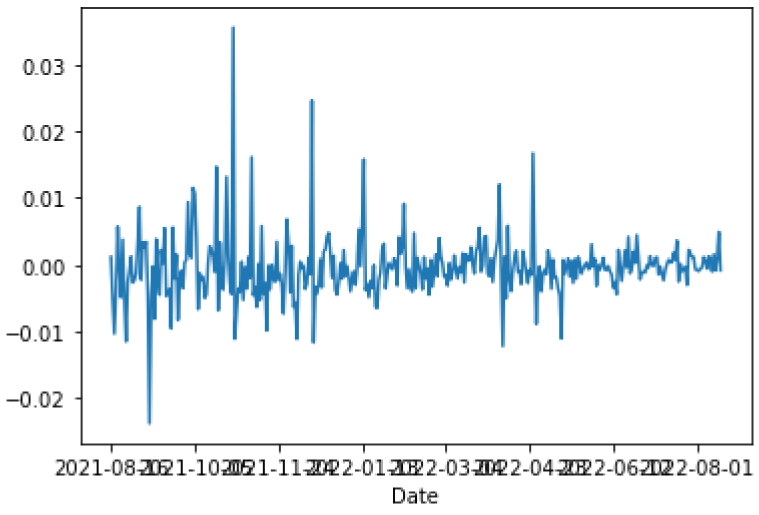


In [ ]:

```
res.resid[1:].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feaddd65f70>
```



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)

In [ ]:

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

Out[ ]:

	lb_stat	lb_pvalue
10	0.942831	0.999869

As the p-value is 0.999, the Null Hypothesis is very significant. Null Hypotheses states that model does not show lack of fit. Hence it is fitted well