نوید ابراهیمی ۹۹۵۲۱۰۰۱

```
[86] prices = {}
     returns = {}


     for crypto in cryptos:
         ticker = yf.Ticker(crypto)
         prices_1mo = ticker.history(start=start_date, end=end_date, interval=timeFrames[0]).Close
         returns_1mo = prices_1mo.pct_change().dropna()

         prices_1d = ticker.history(start=start_date, end=end_date, interval=timeFrames[1]).Close
         returns_1d = prices_1d.pct_change().dropna()

         if crypto not in prices:
             prices[crypto] = []
             returns[crypto] = []

         prices[crypto].append(prices_1mo)
         returns[crypto].append(returns_1mo)

         prices[crypto].append(prices_1d)
         returns[crypto].append(returns_1d)
```

در کد ۵ رمز ارز را بررسی کردم و ۲ تایم‌فریم یک روزه و یک ماهه را نیز بررسی کردم. در عکس اطلاعات هر ارز در timeframe دریافت کرده و ذخیره کردم.

```
for crypto in cryptos:
    plt.figure(figsize=(10,4))
    for i, tF in enumerate(timeFrames):
        plt.plot(prices[crypto][i], label=f"{tF}")
    plt.title(f"{crypto}", fontsize=20)
    plt.ylabel('Prices', fontsize=20)
    plt.legend()
    plt.show()
```

در این قسمت نمودارهای تمام cryptoها در timeframeهای مختلف را چاپ کردم.

ACF

```
p_values = {}

for crypto in cryptos:
  for i, tF in enumerate(timeFrames):
    result = adfuller(prices[crypto][i])
    print(f"crypto: {crypto}, result[1]: {result[1]}, tF: {tF}")
    p_value = result[1]
    if p_value <= 0.1:
      p_values[f"{crypto}, {tF}"] = p_value
```

```
crypto: BTC-USD, result[1]: 0.24637004959145098, tF: 1mo
crypto: BTC-USD, result[1]: 0.8847855278596932, tF: 1d
crypto: ETH-USD, result[1]: 0.004095915279189181, tF: 1mo
crypto: ETH-USD, result[1]: 0.4338634428412075, tF: 1d
crypto: USDT-USD, result[1]: 0.139401124798061, tF: 1mo
crypto: USDT-USD, result[1]: 0.0019003353971645154, tF: 1d
crypto: BNB-USD, result[1]: 0.664351676000936, tF: 1mo
crypto: BNB-USD, result[1]: 0.38330053826784727, tF: 1d
crypto: XRP-USD, result[1]: 0.3483824579098429, tF: 1mo
crypto: XRP-USD, result[1]: 0.37503245177282124, tF: 1d
```

در اینجا adf test را انجام دادم. در ادامه هر کدام که بالای ۹۰ درصد مانا بود را در لیست برای استفاده بعدی ذخیره کردم.

```
[90] min_p_value_crypto = min(p_values, key=p_values.get)
     print(f'The cryptocurrency with the smallest p-value is {min_p_value_crypto} with a p-value of {p_values[min_p_value_crypto]}')
```

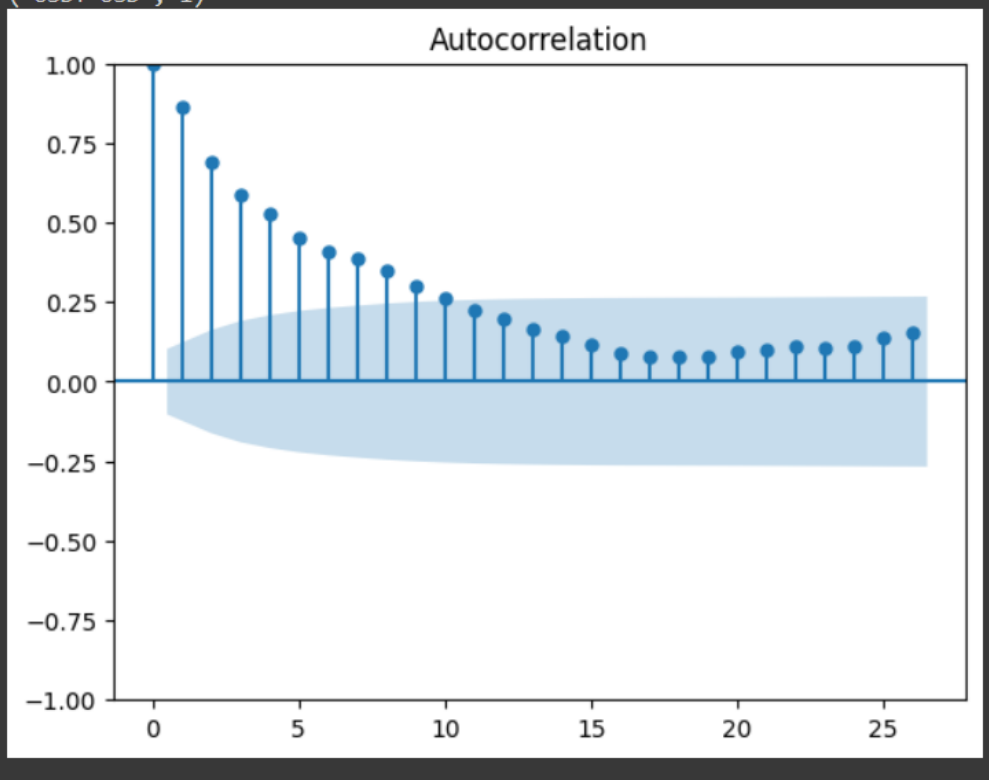
```
The cryptocurrency with the smallest p-value is USDT-USD, 1d with a p-value of 0.0019003353971645154
```

در اینجا نیز از بین تمام cryptoهای موجود، بهترین (ارزی که کمتری p value را دارد) را جدا کردم.

ACF

```
acf_plot = plot_acf(bestAmountOfPvalue)
crypto, timeFrames.index(tF)
```

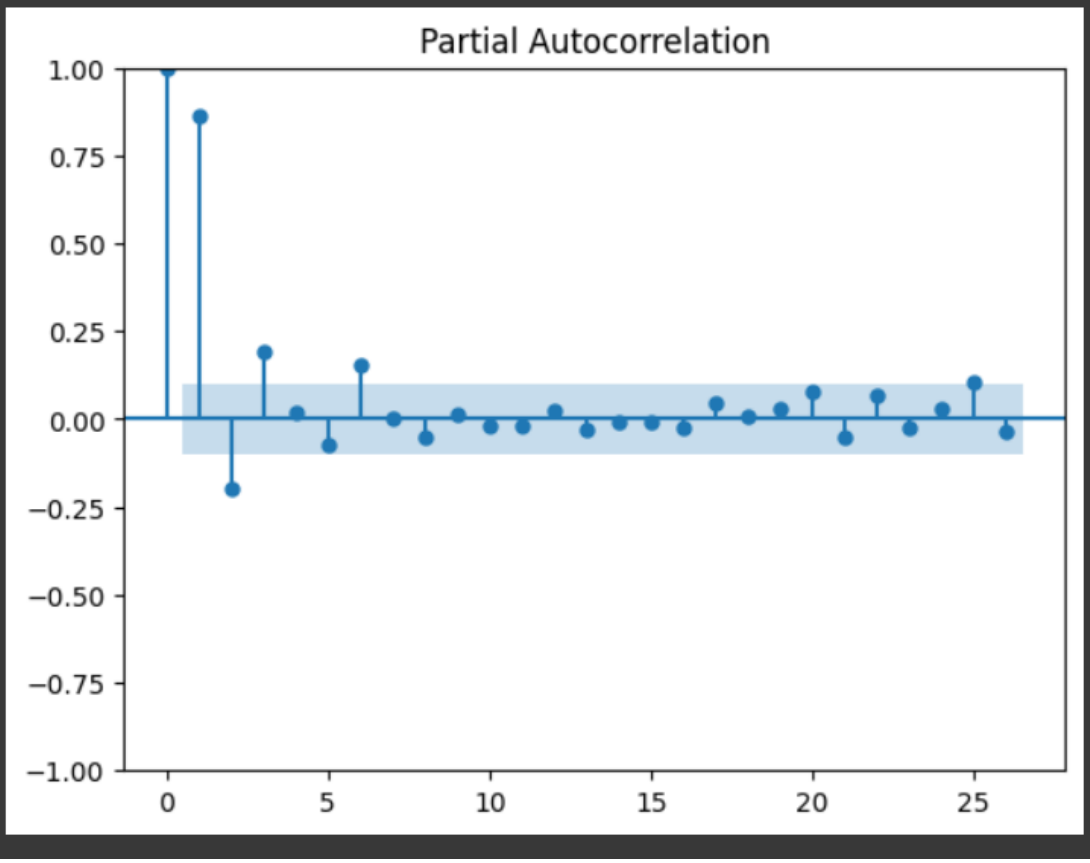
```
('USDT-USD', 1)
```



چاپ نمودار ACF

PACF

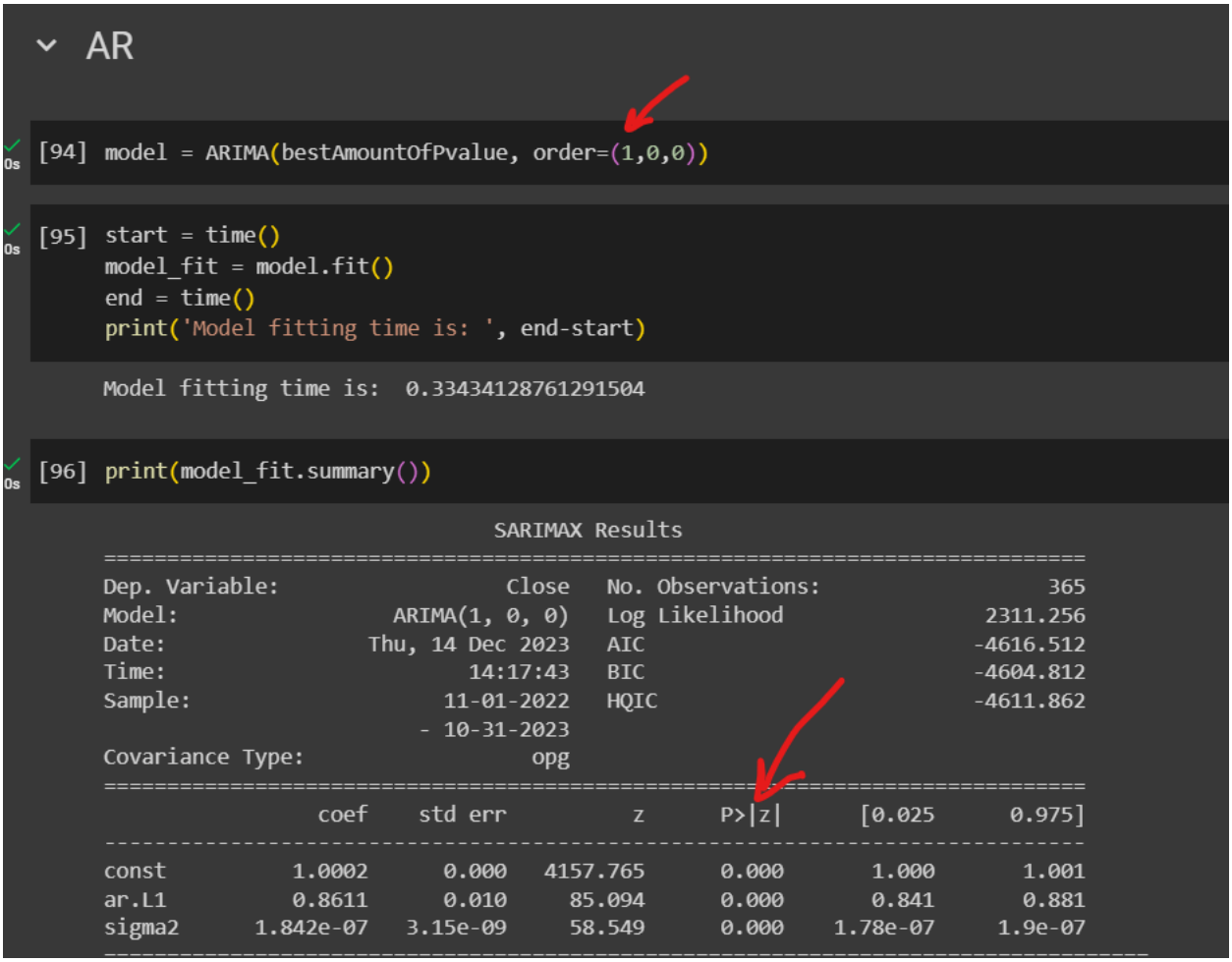
```
[93] pacf_plot = plot_pacf(bestAmountOfPvalue)
```



چاپ نمودار PACF

AR

```
[94] model = ARIMA(bestAmountOfPvalue, order=(1,0,0))

[95] start = time()
     model_fit = model.fit()
     end = time()
     print('Model fitting time is: ', end-start)

     Model fitting time is:  0.33434128761291504

[96] print(model_fit.summary())
```



SARIMAX Results

Dep. Variable:	Close	No. Observations:	365
Model:	ARIMA(1, 0, 0)	Log Likelihood	2311.256
Date:	Thu, 14 Dec 2023	AIC	-4616.512
Time:	14:17:43	BIC	-4604.812
Sample:	11-01-2022	HQIC	-4611.862
	- 10-31-2023		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.0002	0.000	4157.765	0.000	1.000	1.001
ar.L1	0.8611	0.010	85.094	0.000	0.841	0.881
sigma2	1.842e-07	3.15e-09	58.549	0.000	1.78e-07	1.9e-07

با توجه به نمودار PACF، بهترین مقدار برای p، مقدار ۱ بود. با ران کردن کد همانطور که مشخص است مقدار P>|z| که به عبارتی همان p value است زیر 0.1 و این نشانه خوبی برای ماست.

MA

```
[97] model = ARIMA(prices[crypto][timeFrames.index(tF)], order=(0,0,3))
```

```
[98] start = time()
     model_fit = model.fit()
     end = time()
     print('Model fitting time is: ', end-start)
```

```
Model fitting time is:  0.6195120811462402
```

```
[99] print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  365
Model:                 ARIMA(0, 0, 3)   Log Likelihood                2257.856
Date:                Thu, 14 Dec 2023   AIC                          -4505.713
Time:                        14:17:43   BIC                          -4486.213
Sample:                    11-01-2022   HQIC                         -4497.963
                         - 10-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0002      0.000   8956.993      0.000       1.000       1.000
ma.L1          1.0737      0.017     62.862      0.000       1.040       1.107
ma.L2          0.7826      0.019     42.102      0.000       0.746       0.819
ma.L3          0.5254      0.018     29.040      0.000       0.490       0.561
sigma2      2.463e-07   7.48e-09     32.949      0.000    2.32e-07    2.61e-07
```

به همین صورت با نگاه به نمودار ACF، مقدار ۳ مقدار مناسبی بود. با ران کردن کد، همانطور که مشخص است مقدار P>|z| که به عبارتی همان p value است زیر 0.1 و این نشانه خوبی برای ماست.

ARMA

```
[100] model = ARIMA(prices[crypto][timeFrames.index(tF)], order=(1,0,3))
```

```
[101] start = time()
      model_fit = model.fit()
      end = time()
      print('Model fitting time is: ', end-start)
```

```
Model fitting time is:  0.307051420211792
```

```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  365
Model:                 ARIMA(1, 0, 3)   Log Likelihood                2327.320
Date:                Thu, 14 Dec 2023   AIC                          -4642.640
Time:                        14:17:44   BIC                          -4619.241
Sample:                    11-01-2022   HQIC                         -4633.341
                         - 10-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0002      0.000   4285.399      0.000       1.000       1.001
ar.L1          0.8928      0.026     34.401      0.000       0.842       0.944
ma.L1          0.1858      0.036      5.135      0.000       0.115       0.257
ma.L2         -0.1820      0.050     -3.674      0.000      -0.279      -0.085
ma.L3         -0.1653      0.038     -4.326      0.000      -0.240      -0.090
sigma2      1.679e-07   5.31e-09     31.616      0.000    1.58e-07    1.78e-07
==============================================================================
```

همان نکات قبلی :)

Test

```
train_start = '2022-11-01'
train_end = '2023-09-30'
test_start = '2023-10-01'
test_end = '2023-11-01'
```

```
[104] train_end = datetime(2023,9,30, tzinfo=timezone.utc)
      test_end = datetime(2023,11,1, tzinfo=timezone.utc)

      train_data = prices[crypto][timeFrames.index(tF)].loc[:train_end][:]
      test_data = prices[crypto][timeFrames.index(tF)].loc[train_end + timedelta(days=1):test_end]
      print(train_end)
      print(test_end)
```

بازه بندی تقسیم دیتا به train و test را در این بخش انجام دادم. به طوری که ۱۱ ماه برای train و ۱ ماه برای test گذاشتم.

AR

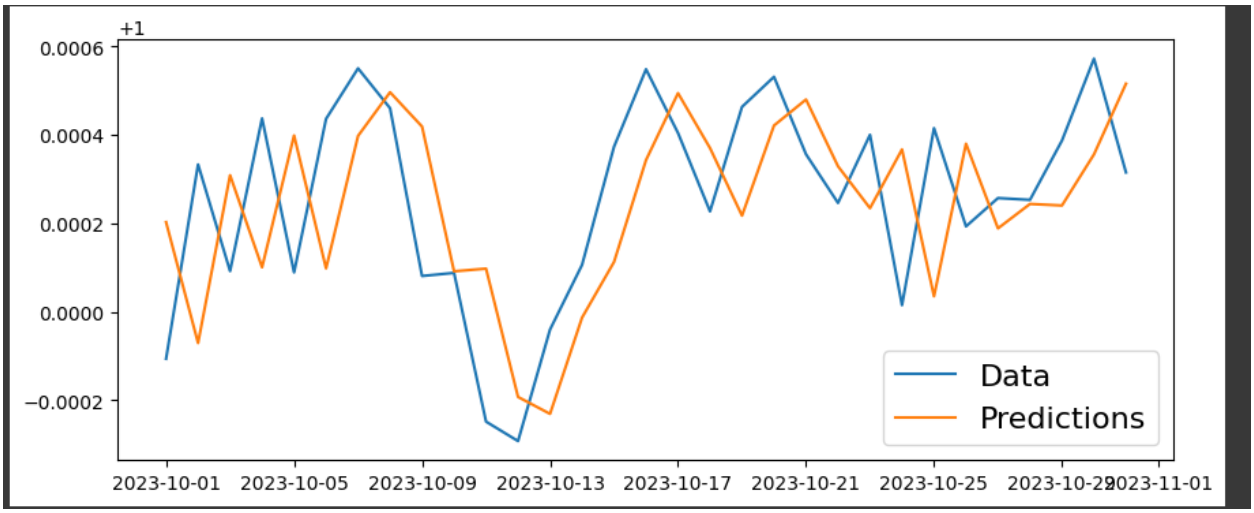
```
model = ARIMA(train_data, order=(1,0,0))
```

```
[106] start = time()
      model_fit = model.fit()
      end = time()
      print('Model fitting time is: ', end-start)

      Model fitting time is:  0.1866466999053955
```

```
[107] pred_start_date = test_data.index[0]
      pred_end_date = test_data.index[-1]
      test_data.index[-1]

      Timestamp('2023-10-31 00:00:00+0000', tz='UTC')
```

```
[108] predictions = model_fit.predict(start=pred_start_date, end=pred_end_date)
      residuals = test_data - predictions
```

حال با همان مقادیر order قبلی مدل را با ۱۱ ماه آموزش و برای ۱ ماه آینده عمل پیش‌بینی را انجام دادم.

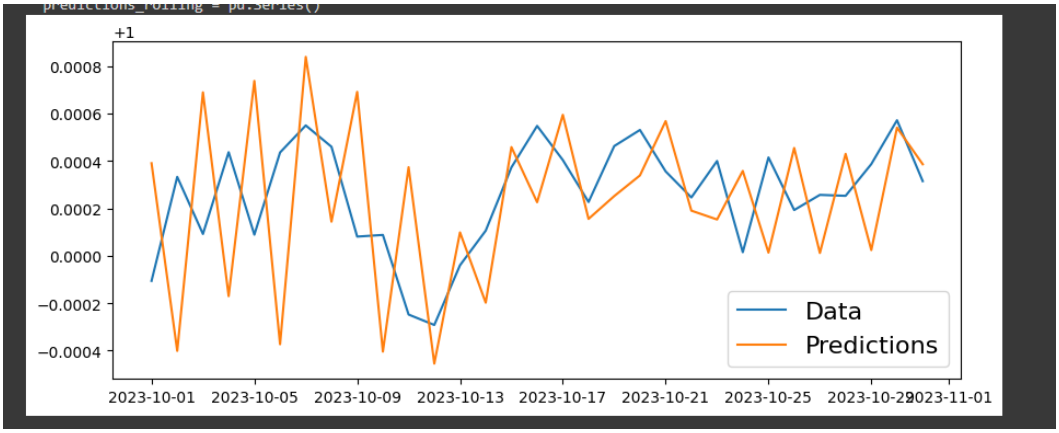


```
MAPE

[111] print('Mean absolute percent error (MAPE):', np.mean(abs(residuals/test_data)))

      Mean absolute percent error (MAPE): 0.00021509993688424687

MSE

[112] print('Mean Squared Error (MSE):', np.mean(residuals**2))

      Mean Squared Error (MSE): 6.072486496603657e-08
```

دو معیار محاسبه ارور خواسته شده در سوال

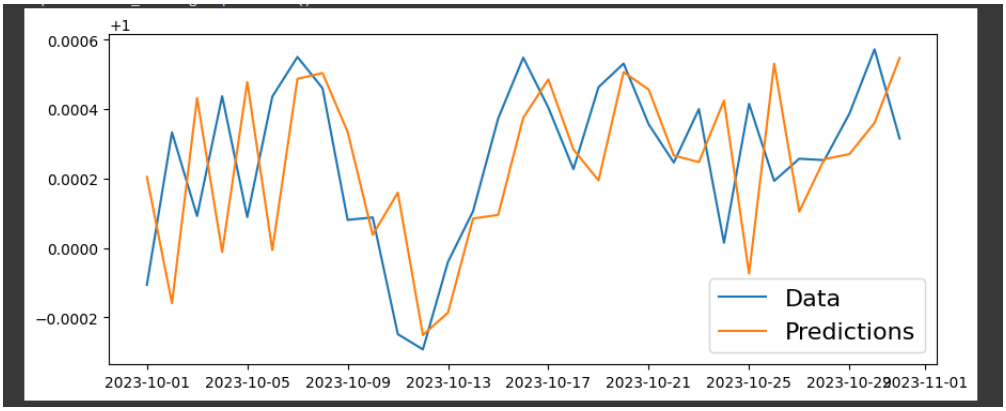
در ادامه همین کار را برای MA و ARMA نیز انجام دادم و صرفا خروجی نموداری را قرار میدهم:

MA:




```
MAPE

[118] print('Mean absolute percent error (MAPE):', np.mean(abs(residuals/test_data)))

      Mean absolute percent error (MAPE): 0.00022763869163227096

MSE

[119] print('Mean Squared Error (MSE):', np.mean(residuals**2))

      Mean Squared Error (MSE): 6.863598464636257e-08
```

:ARMA



MAPE

```
[125] print('Mean absolute percent error (MAPE):', np.mean(abs(residuals/test_data)))
```

```
Mean absolute percent error (MAPE): 0.00021602185511825452
```

MSE

```
[126] print('Mean Squared Error (MSE):', np.mean(residuals**2))
```

```
Mean Squared Error (MSE): 6.287599006872623e-08
```

Q3

```
[127] def ar_model(p):
        model = ARIMA(train_data, order=(p,0,0))
        model_fit_predict = model.fit()
        predictions = model_fit_predict.predict(start=pred_start_date, end=pred_end_date)
        residuals = test_data - predictions
        return residuals
```

```
[128] def ma_model(q):
        model = ARIMA(train_data, order=(0,0,q))
        model_fit_predict = model.fit()
        predictions = model_fit_predict.predict(start=pred_start_date, end=pred_end_date)
        residuals = test_data - predictions
        return residuals
```

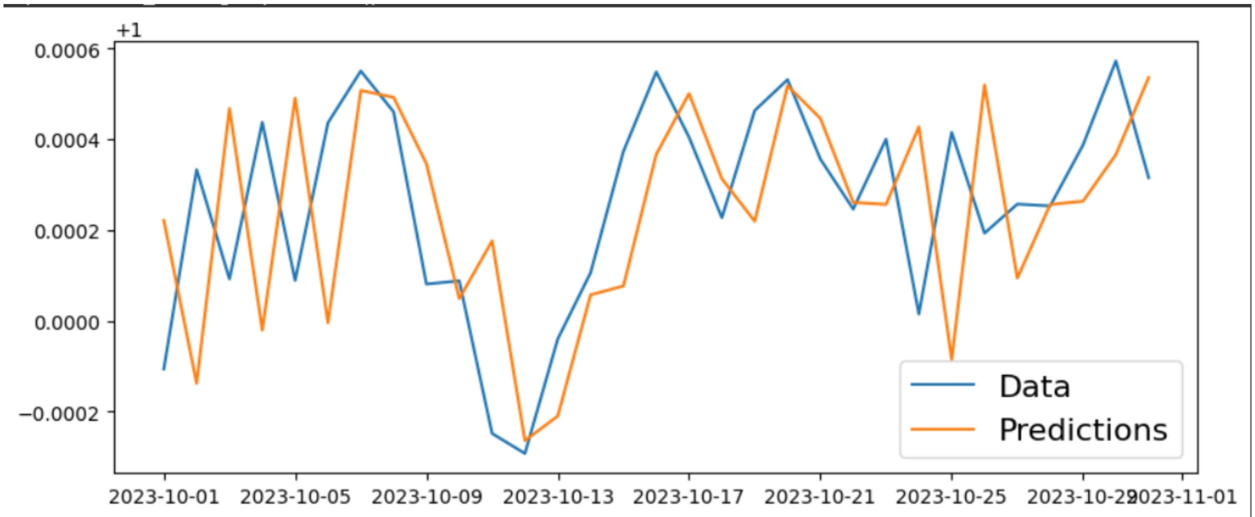
```
[129] def arma_model(p, q):
        model = ARIMA(train_data, order=(p,0,q))
        model_fit_predict = model.fit()
        predictions = model_fit_predict.predict(start=pred_start_date, end=pred_end_date)
        residuals = test_data - predictions
        return residuals
```

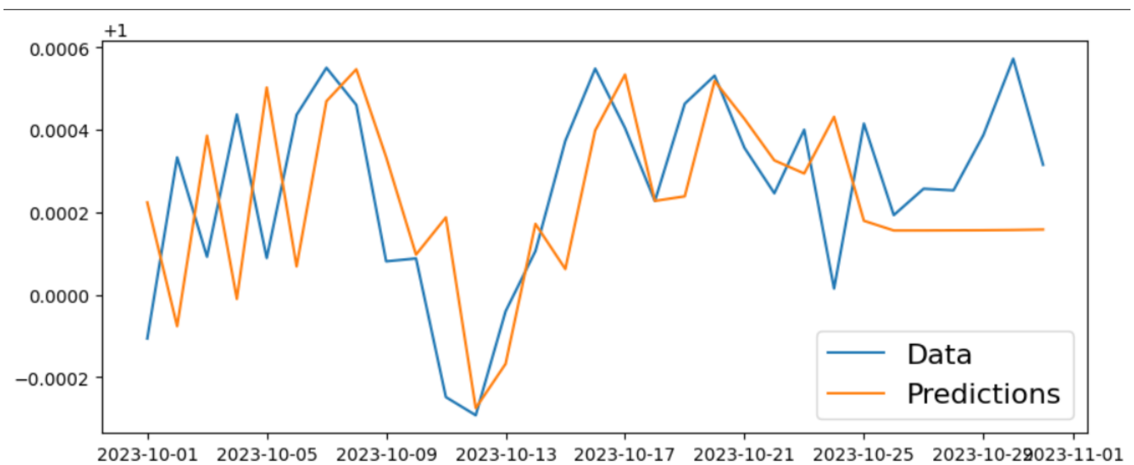
در این قسمت ۳ مدل را در function نوشتم.

```
[130] B_p = [0, 0, 0]
      B_q = [0, 0, 0]
      B_MAPE = [1, 1, 1]
      B_MSE = [1, 1, 1]
      B_res = []

      for p in range(1, 16):
        print(f"p: {p}")
        for q in range(1, 16):
          residuals = [ar_model(p), ma_model(q), arma_model(p, q)]
          for i, res in enumerate(residuals):
            MAPE = np.mean(abs(res/test_data))
            MSE = np.mean(res**2)
            if MAPE < B_MAPE[i] and MSE < B_MSE[i]:
              B_MAPE[i] = MAPE
              B_MSE[i] = MSE
              if i == 0:
                B_p[i] = p
                B_q[i] = 0
              elif i == 1:
                B_q[i] = q
                B_p[i] = 0
              else:
                B_p[i] = p
                B_q[i] = q
              print(f"Best p: {B_p}, Best q: {B_q}, B_MAPE[i]: {B_MAPE[i]}, B_MSE[i]: {B_MSE[i]}")
```

در اینجا همان خواسته سوال که brute force زدن روی مقادیر p و q بود را انجام دادم. تعریف لیست‌ها صرفا بخاطر کم کردن محاسبات تکراری‌ست. اینکار برای هر ۳ مدل تعریف شده در همین قسمت انجام می‌شود.

حال با مقادیر p و q به دست آمده مدل‌های را ران میکنیم.





همانطور که مشخص است نمودارها به شدت بهتر از حالت قبل به دست آمده‌اند و فیت‌تر شده‌اند.

```
for AR:
MSE: 6.033112318708502e-08
MAPE: 0.00021495818080776485

for MA:
MSE: 5.708712843912922e-08
MAPE: 0.00020583432504083069

for ARMA:
MSE: 5.1861016827461545e-08
MAPE: 0.00018531954550841 96
```

با مقایسه این اعداد با حالت قبل مشاهده می‌شود مقادیر خطا هم به نسبت کمتر شده‌اند و شاهد خطای کمتری هستیم.