개별 연구

최종 보고서

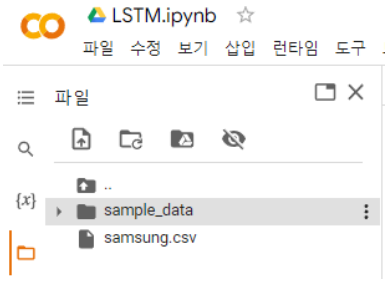
학 과	컴퓨터공학과
학 번	2017112292
이 름	김준하

1. 코드 구현에 사용된 모델과 데이터셋

- 1차 코드 구현
  - 모델: ARIMA, SARIMA 모델
  - 데이터셋: 최근 5년간의 삼성전자 주가 데이터(첨부된 파일 'samsung_1차' 파일을 'samsung'으로 이름 변경 후 사용)
- 3차 코드 구현
  - 모델: LSTM 모델
  - 데이터셋: 최근 5년간의 삼성전자 주가 데이터(첨부된 파일 'samsung_3차' 파일을 'samsung'으로 이름 변경 후 사용)

2. 개발 환경

- Google Colab
  - 다음과 같이 주식 데이터 파일(samsung.csv)을 업로드하여 실행



3. 모델링 과정

- ARIMA, SARIMA 모델의 모델링 과정(1차 구현)

I. 데이터 전처리

i. 정상성(stationarity) 확인

위 그림은 원본 데이터를 시계열 분해한 모습이다.

원본 데이터만 보아도 시점에 관계없이 평균과 분산이 일정하지 않으므로 정상성을 만족하지 않을 것이라고 예상할 수 있다. 즉, 차분이 필요할 것으로 예상된다.

ii. 데이터가 stationary하지 않다면 전처리(transformation, differencing) 과정을 통해 stationary하게 바꾸어준다.

```
            Close
Date
2017-10-19  52980
2017-10-20  53840
2017-10-23  54300
2017-10-24  54040
2017-10-25  53900
...           ...
2021-10-12  69000
2021-10-13  68800
2021-10-14  69400
2021-10-15  70100
2021-10-18  70200

[982 rows x 1 columns]
```

```
Date
2017-10-20     860.0
2017-10-23     460.0
2017-10-24    -260.0
2017-10-25    -140.0
2017-10-26   -1500.0
              ...
2021-10-12   -2500.0
2021-10-13    -200.0
2021-10-14     600.0
2021-10-15     700.0
2021-10-18     100.0
Name: Close, Length: 981, dtype: float64
```

왼쪽 데이터가 원본 데이터, 오른쪽 데이터가 차분을 진행한 데이터의 모습이다.

II. 시범적으로 시행해 볼 만한 모델 찾기

여러 방법 중 한 가지로 'Graphical method'가 있는데, 방법은 다음과 같다.

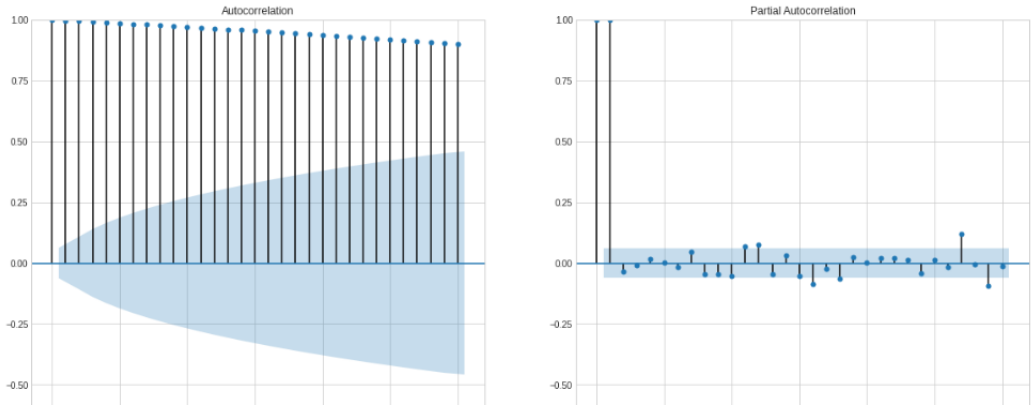
i. 데이터를 사용하여 ACF, PACF plot을 생성하고 그 패턴으로부터 어떠한 모델을 사용할지 선택.

모델	ACF	Partial ACF
MA(q)	q시차 이후 0으로 급감	지수적으로 감소, 소멸하는 sine함수 형태

AR(p)	지수적으로 감소, 소멸하는 sine함수 형태	p시차 이후 0으로 급감
ARMA(p,q)	시차 (q-p)이후 급감	시차 (q-p)이후 급감

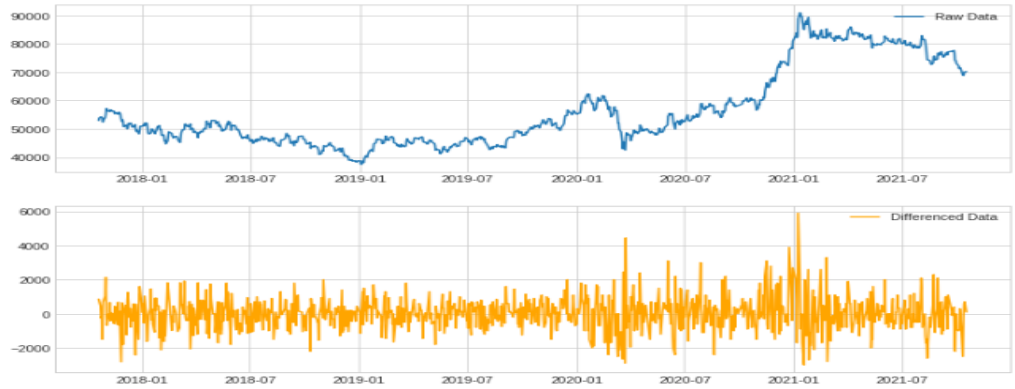
ii. ACF, PACF plot이 다음과 같은 형태일 때, 각각 MA, AR, ARMA 모델이 적합하다고 알려져 있음.
패턴을 파악하는 과정이 주관적일 수 있음.

(시차는 lag를 의미)



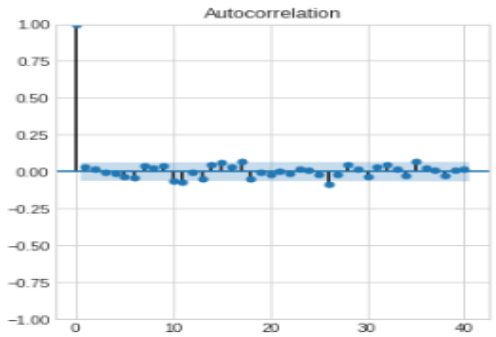
원본 데이터를 사용하여 그린 ACF, PACF 그래프의 모습이다. ACF가 점진적으로 작아지는 것으로 보아 stationary하지 않다는 점을 알 수 있다.

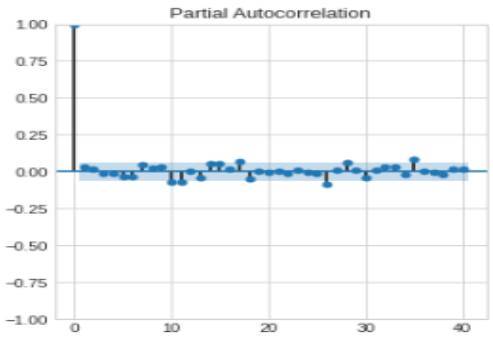
여기에서 1차 차분을 진행하여 다음과 같은 데이터를 얻었다.



차분한 데이터는 보기에는 정상성이 있는 것으로 예상할 수 있다. 이를 정확히 확인하기 위해 ACF, PACF 그래프를 그려 보았다.

다음은 차분한 데이터를 이용하여 ACF, PACF 그래프를 그린 모습이다.





차분한 데이터의 ACF, PACF 그래프를 보면 정상성이 존재한다고 볼 수 있다. 이를 통해 graphical method를 사용하여 어떠한 모델을 사용할 것인지 정할 수 있다.

차분한 데이터에 graphical method를 적용하면 시점 1 이후에 급감하므로 ARMA(p,d,q)에서 q-p=1이 된다. 여기에서 1차 차분을 적용했으므로 ARIMA(1,1,2) 모델을 선정해 보았다.

III. parameter 추정

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	982
Model:	ARIMA(1, 1, 2)	**Log Likelihood**	-8116.739
Date:	Thu, 27 Oct 2022	**AIC**	16241.478
Time:	02:32:56	**BIC**	16261.033
Sample:	0 - 982	**HQIC**	16248.917
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5182	0.867	-0.598	0.550	-2.217	1.180
ma.L1	0.5501	0.868	0.634	0.526	-1.150	2.250
ma.L2	0.0356	0.032	1.116	0.264	-0.027	0.098
sigma2	9.024e+05	2.68e+04	33.622	0.000	8.5e+05	9.55e+05

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	307.91
Prob(Q):	0.96	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.80	**Skew:**	0.42
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.61

ARIMA(1,1,2) 모델을 사용하여 훈련을 진행하고 얻은 결과이다. 이는 단순히 graphical method를 사용하여 p, q를 추정한 것이다.

IV. 모델이 괜찮은지 확인

총 4개의 모델을 만들어 보았다.

① ARIMA 모델1 (statsmodel.tsa.arima.model.ARIMA)

② SARIMA 모델1 (statsmodel.tsa.statespace.sarimax.SARIMAX)

③ ARIMA 모델2 (pmdarima.arima.auto_arima)

④ SARIMA 모델2 (pmdarima.arima.auto_arima)

'ARIMA 모델1'에서는 p와 q는 0~3의 범위, d는 1~2의 범위를 주어 최적의 조합을 탐색했다.

'SARIMA 모델1'에서는 SARIMA(0,1,0)(0,1,0,12) 부터 SARIMA(2,1,2)(2,1,2,12) 까지의 조합을 탐색한 결과, SARIMA 모델은 탐색 시간이 오래 걸려서 이후에는 탐색의 범위를 줄였다.

그 결과 다음과 같이 각각 ARIMA(2,1,2) 모델과 SARIMA(1,1,0)(0,1,2,12)가 AIC값이 가장 낮은 최적의 조합임을 확인할 수 있다.

SARIMAX Results

Dep. Variable:	y	No. Observations:	982
Model:	ARIMA(2, 1, 2)	Log Likelihood	-8109.215
Date:	Thu, 27 Oct 2022	AIC	16228.431
Time:	06:29:51	BIC	16252.874
Sample:	0 - 982	HQIC	16237.729

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.1737	0.044	26.958	0.000	1.088	1.259
ar.L2	-0.9114	0.044	-20.483	0.000	-0.999	-0.824
ma.L1	-1.1379	0.045	-25.271	0.000	-1.226	-1.050
ma.L2	0.9063	0.047	19.208	0.000	0.814	0.999
sigma2	9.024e+05	2.8e+04	32.178	0.000	8.47e+05	9.57e+05

Ljung-Box (L1) (Q):	0.24	Jarque-Bera (JB):	304.31
Prob(Q):	0.62	Prob(JB):	0.00
Heteroskedasticity (H):	1.80	Skew:	0.43
Prob(H) (two-sided):	0.00	Kurtosis:	5.59

SARIMAX Results

Dep. Variable:	y	No. Observations:	982
Model:	SARIMAX(1, 1, 0)x(0, 1, [1, 2], 12)	Log Likelihood	-8154.245
Date:	Thu, 27 Oct 2022	AIC	16316.490
Time:	06:39:47	BIC	16335.995
Sample:	0 - 982	HQIC	16323.915

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0440	0.025	1.753	0.080	-0.005	0.093
ma.S.L12	-0.6993	0.013	-55.632	0.000	-0.724	-0.675
ma.S.L24	-0.2021	0.008	-26.146	0.000	-0.217	-0.187
sigma2	1.059e+06	3.2e+04	33.110	0.000	9.96e+05	1.12e+06

Ljung-Box (L1) (Q):	0.04	Jarque-Bera (JB):	746.38
Prob(Q):	0.85	Prob(JB):	0.00
Heteroskedasticity (H):	1.13	Skew:	-0.37
Prob(H) (two-sided):	0.27	Kurtosis:	7.23

'ARIMA 모델2'와 'SARIMA 모델2'는 앞선 모델들과 비교해서 사용하는 함수만 다르고, 동일한 방식으로 여러 조합들 중에서 최적의 조합을 선택하여 모델을 만든다. 그런데 동일한 조합의 모델이라고 해도 약간의 차이가 있는 것을 볼 수 있었는데, 이는 다른 함수를 사용하기 때문이라고 생각된다.

다음은 순서대로 'ARIMA 모델2'와 'SARIMA 모델2'의 분석 결과인데, ARIMA 모델2는 ARIMA(2,1,2) 모델, SARIMA 모델2는 SARIMA(0,1,0)(0,1,1,12) 모델이 최적의 조합임을 알 수 있다.

SARIMAX Results

Dep. Variable:	y	No. Observations:	982
Model:	SARIMAX(2, 1, 2)	Log Likelihood	-8109.090
Date:	Thu, 27 Oct 2022	AIC	16230.181
Time:	06:43:08	BIC	16259.512
Sample:	0 - 982	HQIC	16241.339

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	16.6461	24.149	0.689	0.491	-30.684	63.976
ar.L1	1.1739	0.043	26.988	0.000	1.089	1.259
ar.L2	-0.9118	0.044	-20.497	0.000	-0.999	-0.825
ma.L1	-1.1382	0.045	-25.316	0.000	-1.226	-1.050
ma.L2	0.9066	0.047	19.235	0.000	0.814	0.999
sigma2	9.022e+05	2.87e+04	31.435	0.000	8.46e+05	9.58e+05

Ljung-Box (L1) (Q):	0.24	Jarque-Bera (JB):	304.35
Prob(Q):	0.63	Prob(JB):	0.00
Heteroskedasticity (H):	1.80	Skew:	0.43
Prob(H) (two-sided):	0.00	Kurtosis:	5.59

SARIMAX Results

Dep. Variable:	y	No. Observations:	982
Model:	SARIMAX(0, 1, 0)x(0, 1, [1], 12)	Log Likelihood	-8179.188
Date:	Thu, 27 Oct 2022	AIC	16362.376
Time:	06:46:26	BIC	16372.129
Sample:	0 - 982	HQIC	16366.089

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
ma.S.L12	-0.7448	0.009	-78.933	0.000	-0.763	-0.726
sigma2	1.085e+06	3.22e+04	33.714	0.000	1.02e+06	1.15e+06

Ljung-Box (L1) (Q):	0.38	Jarque-Bera (JB):	4232.31
Prob(Q):	0.54	Prob(JB):	0.00
Heteroskedasticity (H):	0.98	Skew:	-1.03
Prob(H) (two-sided):	0.86	Kurtosis:	13.03

- LSTM 모델의 모델링 과정

I. 설정

  - random seed를 설정하지 않으면 실행마다 결과가 달라지므로 설정 후 실행한다.

II. 데이터 전처리

- Volume이 0인 값들은 거래량이 0인 데이터로 생각된다. 이는 데이터로서 가치가 없으므로 사용하지 못하는 데이터로 바꾼다.
- na값은 의미 없는 데이터이므로 이에 해당하는 행들을 삭제한다.

III. 변수 scaling

- 데이터 간의 범위 차이가 큰 경우 학습에 부정적인 영향을 미칠 수 있고, 실제 운영에서도 학습에서 사용되지 않은 큰 데이터가 들어오면 모델이 강하게 발산할 여지가 존재하므로 정규화를 통하여 모델을 안정적으로 만들어주는 과정이 필요하다.

IV. shifting을 통해 window 생성

- 이전의 데이터들을 사용하여 다음의 데이터 값을 예측하기 위해 사용되는 sliding window를 생성한다.

V. 훈련셋과 테스트셋 생성

- scaling이 완료된 window로 이루어진 특징 리스트와 라벨 리스트에서 8:2의 비율로 훈련셋과 테스트셋을 구성한다.

VI. LSTM 모델 생성

- 여러 층의 레이어를 선형으로 연결하여 구성하기 위하여 Sequential 모델을 생성하고, 활성화 함수로 tanh를 사용하는 LSTM셀과 활성화 함수로 선형 함수를 사용하는 Dense 층을 하나씩 추가하였다.
- 모델은 손실함수로 평균제곱오차(mse), 옵티마이저로 adam, 측정항목함수(metrics)로 평균절대오차(mae)로 학습방식을 설정하였다.

```
#손실함수로는 평균제곱오차를, 옵티마이저로는 adam을 사용한다.
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 128)               66560

 dense_6 (Dense)             (None, 1)                 129

=================================================================
Total params: 66,689
Trainable params: 66,689
Non-trainable params: 0
_________________________________________________________________
```

VII. 훈련셋을 사용한 모델 훈련

- Early stopping을 사용하여 validation loss가 5번 연속으로 증가하면 모델 훈련을 멈추도록 하였다.
  총 100번을 반복하여 훈련하고 훈련이 진행되는 batch의 크기는 16으로 설정하였다.

```
#특정 조건에 도달하면 종료하기 위해 EarlyStopping을 사용한다.
early_stop = EarlyStopping(monitor='val_loss', patience=5)

model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=100, batch_size=16,
          callbacks=[early_stop])
```

```
Epoch 1/100
59/59 [==============================] - 4s 35ms/step - loss: 0.0161 - mae: 0.0717 - val_loss: 0.0013 - val_mae: 0.0286
```

VIII. 테스트셋을 사용한 테스팅

4. 분석

I. ARIMA, SARIMA 모델에 대한 분석1

오른쪽과 같은 모델에 대해 분석해 보고자 한다.

SARIMAX Results

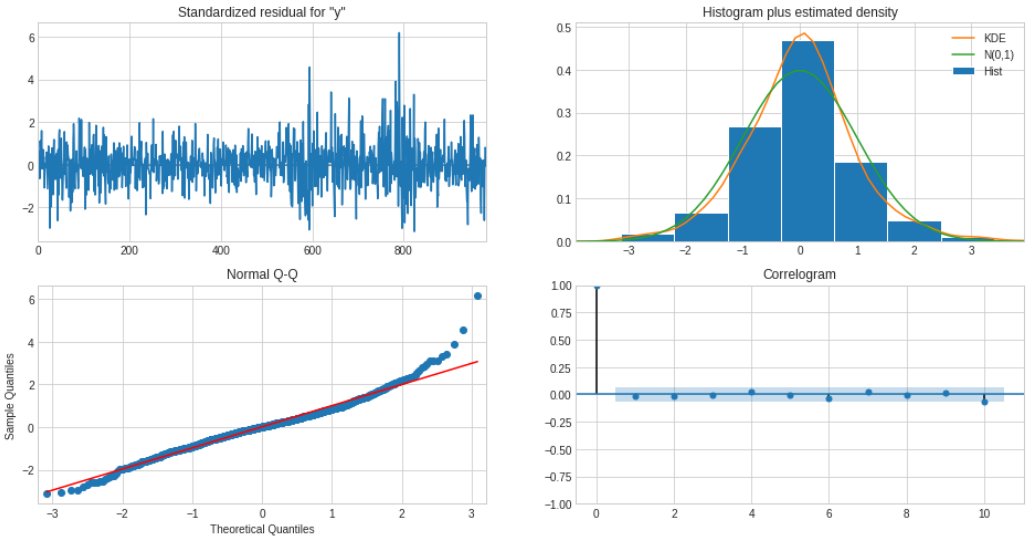
Dep. Variable:	y	**No. Observations:**	982
Model:	ARIMA(1, 1, 2)	**Log Likelihood**	-8116.739
Date:	Thu, 27 Oct 2022	**AIC**	16241.478
Time:	02:32:56	**BIC**	16261.033
Sample:	0	**HQIC**	16248.917
	- 982		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5182	0.867	-0.598	0.550	-2.217	1.180
ma.L1	0.5501	0.868	0.634	0.526	-1.150	2.250
ma.L2	0.0356	0.032	1.116	0.264	-0.027	0.098
sigma2	9.024e+05	2.68e+04	33.622	0.000	8.5e+05	9.55e+05

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	307.91
Prob(Q):	0.96	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.80	**Skew:**	0.42
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.61

- 가장 최적의 모델은 AIC 값이 가장 낮은 모델을 말한다.
- Ljung-Box(Q) 값은 residual이 백색 잡음인지에 대한 통계량으로, Prob(Q) 값이 0.05보다 작으면 자기 상관성이 존재한다. 0.05보다 크면 자기 상관성이 존재하지 않는다(백색 잡음이다).
- Jarque-Bera(JB) 값은 residual이 정규성을 띠는지에 대한 통계량으로, Prob(JB) 값이 0.05보다 작으면 정규성을 따르지 않는다. 0.05보다 크면 정규성을 따른다.
- Heteroskedasticity(H) 값은 residual이 이분산을 띠는지에 대한 통계량이다.
- skew(비대칭도)는 0에 가까울수록 residual이 정규 분포를 따른다.
- Kurtosis(첨도)는 3에 가까울수록 residual이 정규 분포를 따른다.

II. ARIMA, SARIMA 모델에 대한 분석2



- Standardized residual은 residual을 시계열로 나타낸 것이다. 백색 잡음이므로 시계열이 평균 0을 중심으로 무작위하게 움직인다.
- Correlogram은 residual의 ACF를 나타낸 것이다. 위에 주어진 Correlogram은 허용 범위 안에 위치하므로 자기 상관성이 존재하지 않음을 알 수 있다.
- Histogram plus estimated density는 residual의 히스토그램으로, 정규 분포 N(0,1)과 밀도를 추정한 그래프를 겹쳐서 보여준다.
- Normal Q-Q는 정규성을 만족하면 빨간 일직선 위에 점들이 분포한다. 위에 주어진 그래프에서는 대부분 정규성을 만족하지만 양쪽 끝 부분에서 약간 벗어난다.

III. pmdarima.arima.auto_arima 함수에 대한 분석

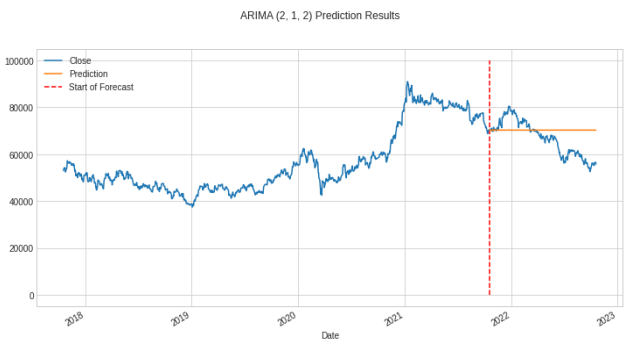
```
auto_arima_model = auto_arima(train_data, start_p=1, start_q=1,max_p=1, max_q=1, m=12,
                              seasonal=True, d=1, D=1, max_P=1, max_Q=2,trace=True,
                              error_action='ignore', suppress_warnings=True, stepwise=False)
```

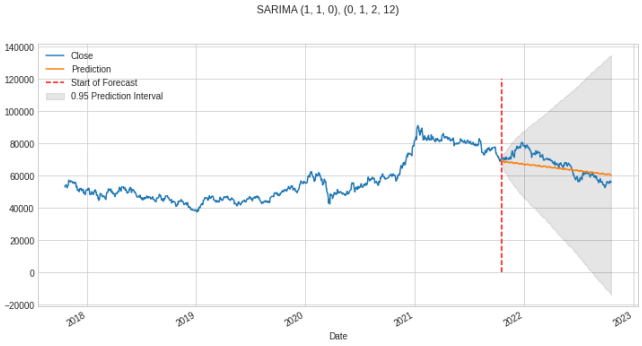
- d: 차분의 차수 (기본값 none)
- D: 계절성 차분의 차수. (기본값 none)
- start_p, max_p: AR(p)에서 p의 범위. (기본값 2~5)
- start_q ,max_q: MA(q)에서 q의 범위. (기본값 2~5)
- m: 계절적 차분이 필요할 때 쓸 수 있는 모수. 4이면 분기별, 12이면 월별, 1이면 계절적 특징을 띠지 않는 데이터를 의미한다. (기본값 1)
- seasonal: 계절성. SARIMA (기본값 True)
- stepwise: 최적의 모수를 찾기 위해 사용하는 힌드만-칸다카르 알고리즘의 사용 여부. False이면 모든 모수 조합으로 모형을 적합한다. (기본값 True)
- trace: 각 stepwise로 모델을 적합할 때마다 결과를 프린트. (기본값 False)
- start_P, max_P: SARIMA에서 P의 범위. (기본값 1~2)
- start_Q, max_Q: SARIMA에서 Q의 범위. (기본값 1~2)
- error_action: 에러가 발생했을 때 처리 방법. (기본값 'warn')
- suppress_warnings: 모델 내부에서 발생하는 많은 warning들을 무시. (기본값 True)

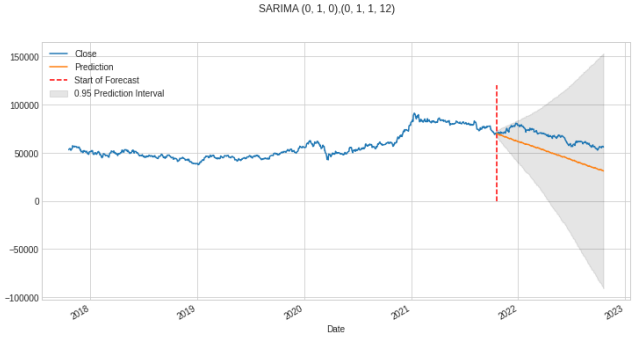
5. 결과

I. ARIMA, SARIMA 모델

- 다음은 ARIMA와 SARIMA 모델들의 순서대로 실제값과 예측한 결과를 나타낸 그림이다. 예측한 지점은 훈련 데이터의 마지막 부분(전체 데이터의 80% 지점)이다.







그래프 상으로는 두 번째 모델인 SARIMA 모델2가 가장 좋은 예측을 보이는 것을 확인할 수 있다.

II. LSTM 모델

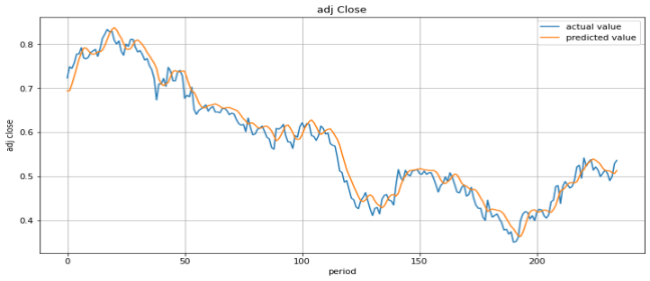
- 모델에 테스트셋을 적용하여 예측을 적용한 후 실제값과 함께 출력한 모습이다.

```
pred = model.predict(x_test)

plt.figure(figsize=(12, 6))
plt.title('adj Close')
plt.ylabel('adj close')
plt.xlabel('period')
plt.plot(y_test, label='actual value')
plt.plot(pred, label='predicted value')
plt.grid()
plt.legend(loc='best')

plt.show()
```

```
8/8 [==============================] - 0s 17ms/step
```



- 예측 결과값을 사용하여 평균 절대 백분율 오차(MAPE)를 구한 결과이다.

```
# 평균 절대 백분율 오차를 계산한다.

print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```

```
0.025162243278213375
```

III. LSTM에서 층을 하나 추가한 결과

- 왼쪽은 기존대로 하나의 LSTM셀 레이어만 사용한 결과이고, 오른쪽은 하나의 레이어를 추가로 사용한 결과이다. MAPE를 계산한 결과를 보면 두 결과의 차이가 거의 나지 않음을 알 수 있다. 하나의 레이어를 추가하는 것은 큰 차이를 보이지 않았다.

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.025162243278213 75

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.025624423965300226

IV. LSTM의 활성화 함수를 ReLU로 사용한 결과

- 왼쪽은 기존대로 LSTM의 활성화 함수로 tanh를 사용한 결과이고, 오른쪽은 활성화 함수로 ReLU를 사용한 결과이다. 오른쪽은 결과는 MAPE가 기존의 결과보다 높으므로 비교적 덜 정확한 모델이라고 볼 수 있다.

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.025162243278213 75

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.029576010967483628

V. window의 크기를 다르게 사용한 결과

- 아래 결과는 기존대로 window의 크기를 40으로 사용한 결과이다.

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.025162243278213 75

- 아래 결과는 왼쪽 위에서부터 순서대로 window의 크기가 5, 10, 20, 50, 60일 때의 결과이다. window의 크기가 너무 작아도 성능이 좋지 못하고 너무 커도 좋지 못하다는 것을 알 수 있다. 결과들 중에서는 window의 크기가 50인 경우에 가장 적절하여 좋은 성능을 가진다는 것을 알 수 있다.

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.06750810836273029

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.048170421562024286

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.04569349168550582

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.024368936960163647

```
# 평균 절대 백분율 오차를 계산한다.
print( np.sum(abs(y_test-pred)/y_test) / len(x_test) )
```
0.030254315779466558