2022전기 졸업과제 중간 보고서

분과 : A

과제명 : 수요예측 알고리즘 경량화 및 서비스 구현

팀명 : 어쩔컴퓨터

팀원

- **서준호(201724489)**
- **윤정현(201724516)**
- **김민태(201724426)**

지도교수

- **박 진 선 교수님**

목차

1. 과제 목표 및 요구 조건

1-1. 과제 목표

수요 예측 인공지능을 구현하고 경량화를 통한 서비스 배포

1-2. 요구 조건

- **어떤 분야의 수요를 예측하는 서비스를 만들지 결정**
- **정해진 데이터 셋에 맞는 수요 예측 알고리즘을 조사 및 구현**
- **알고리즘 경량화**
- **Back-end 및 Front-end 구현**

2. 과제 설계

2-1. 데이터 선정

[그림 1] AI와 주식. [1]

2010년 후반기부터 가장 빠르게 발전한 IT 분야 중 하나는 AI라고 할 수 있다. AI는 현재 모든 분야에서 각광받는 기술이다. 그 중에서도 금융, 특히 주식 시장은 다른 어느 영역에서 보다 AI 기술을 빠르게 받아드리고 있다. 주식 시장에서 인공지능을 활용하려는 가장 큰 이유는 깊게 생각할 필요도 없이 바로 미래의 주가에 대한 예측일 것이다.

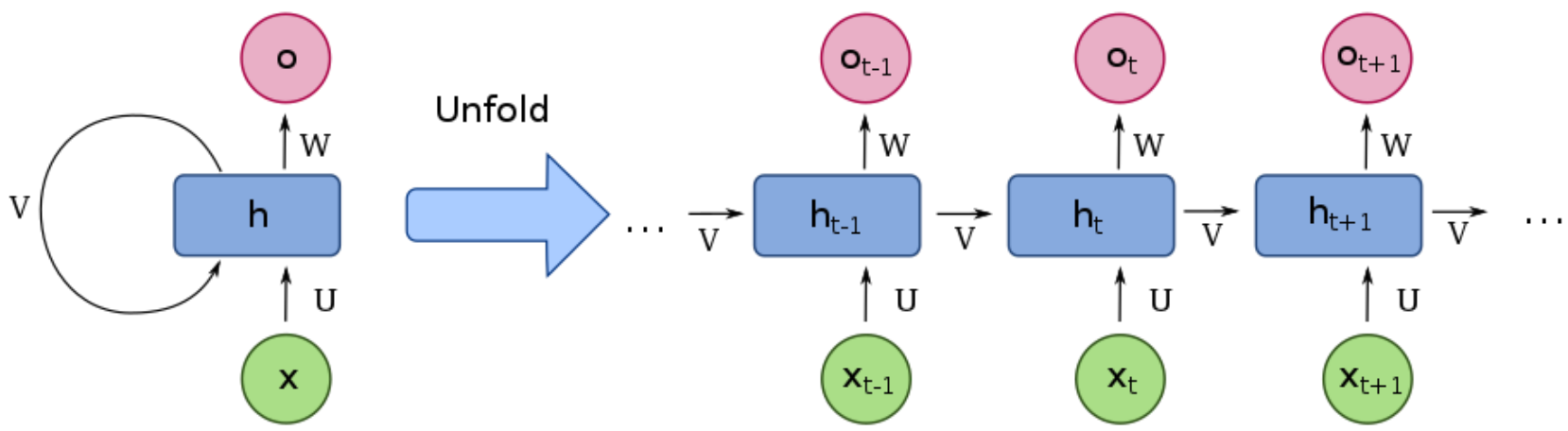
주식은 그 자체로 수요에 대한 데이터라고 말할 수 있다. 주식 가격을 결정하는 가장 기본적인 요인은 수요와 공급이고, 그 균형점에서 주식 가격이 형성된다.

따라서, 주식 가격 데이터를 AI에게 학습시켜 미래의 주식 가격을 예측하는 과제를 진행하고자 하였다.

2-2. 알고리즘 조사

2-2-1. Sequential Model

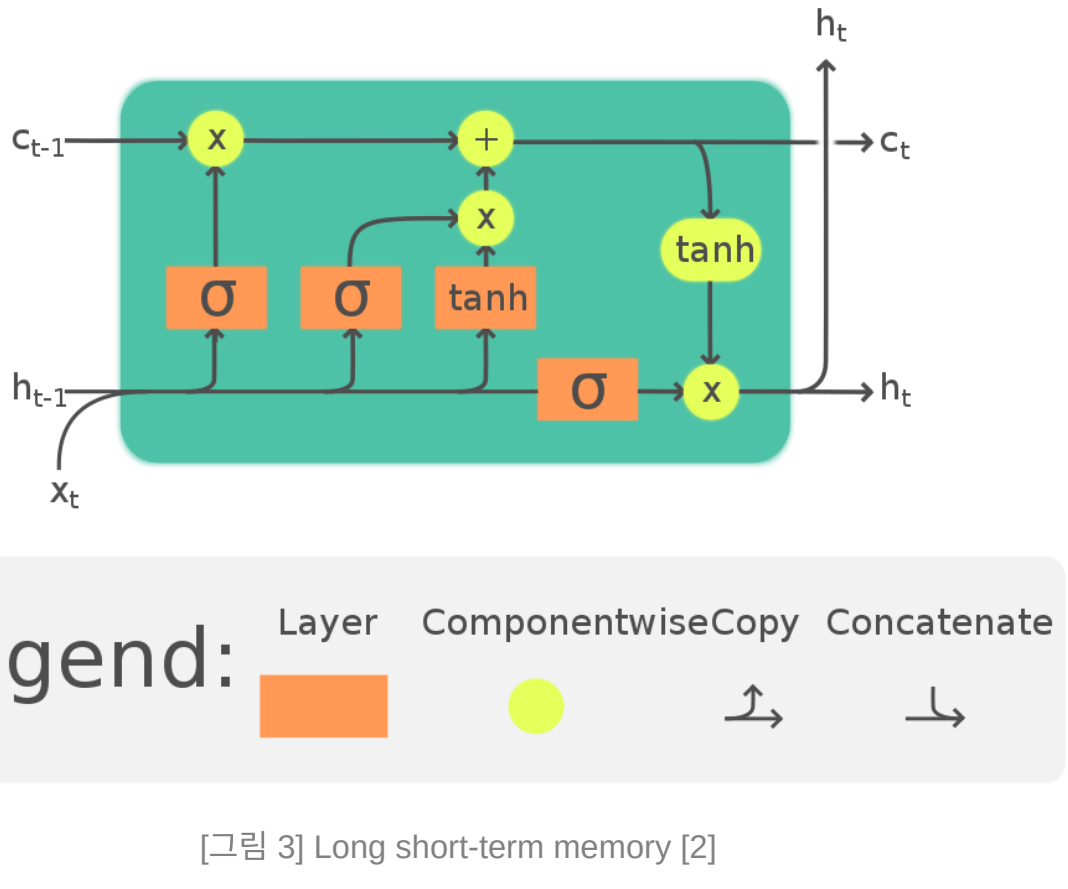
주식 데이터는 시계열 데이터이다. 현재 사용되는 딥러닝 모델 중, 시계열 데이터를 다루는 Sequence model에는 대표적으로 RNN, GRU, LSTM이 있다. 이 중에서 LSTM을 선택하였다.



[그림 2] Recurrent nueral network [2]

그 이유는 RNN에는 Long-term dependencies가 없다는 문제가 있기 때문이다. 즉, 특정 데이터의 위치가 출력과 멀어지면 멀어질 수록 예측에 반영되기가 힘들다. 주식 데이터는 상당히 시간 간격이 긴 정보들을 포함하고 있기 때문에, 꽤 먼 과거의 데이터도 어느정도는 현재의 가격과 연관이 있다.

따라서 이를 해결하기 위해 Long Short-Term Memory(LSTM) 모델을 사용하기로 하였다.

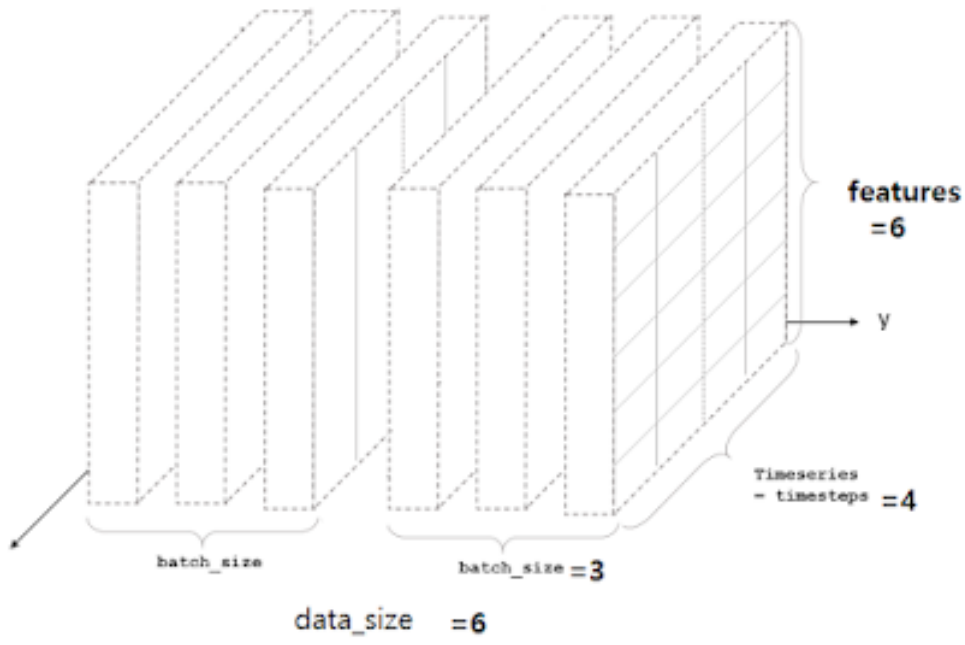


[그림 3] Long short-term memory [2]

LSTM과 RNN의 가장 큰 차이점 중 하나는 Cell State의 유무이다. RNN과 달리 LSTM은 Cell State라는 또 다른 상태가 있고, 매 time step t 마다 forget gate를 통해 이전의 cell state에서 얼마나 보존할 지 결정한다.

2-2-2. Convolutional neural network

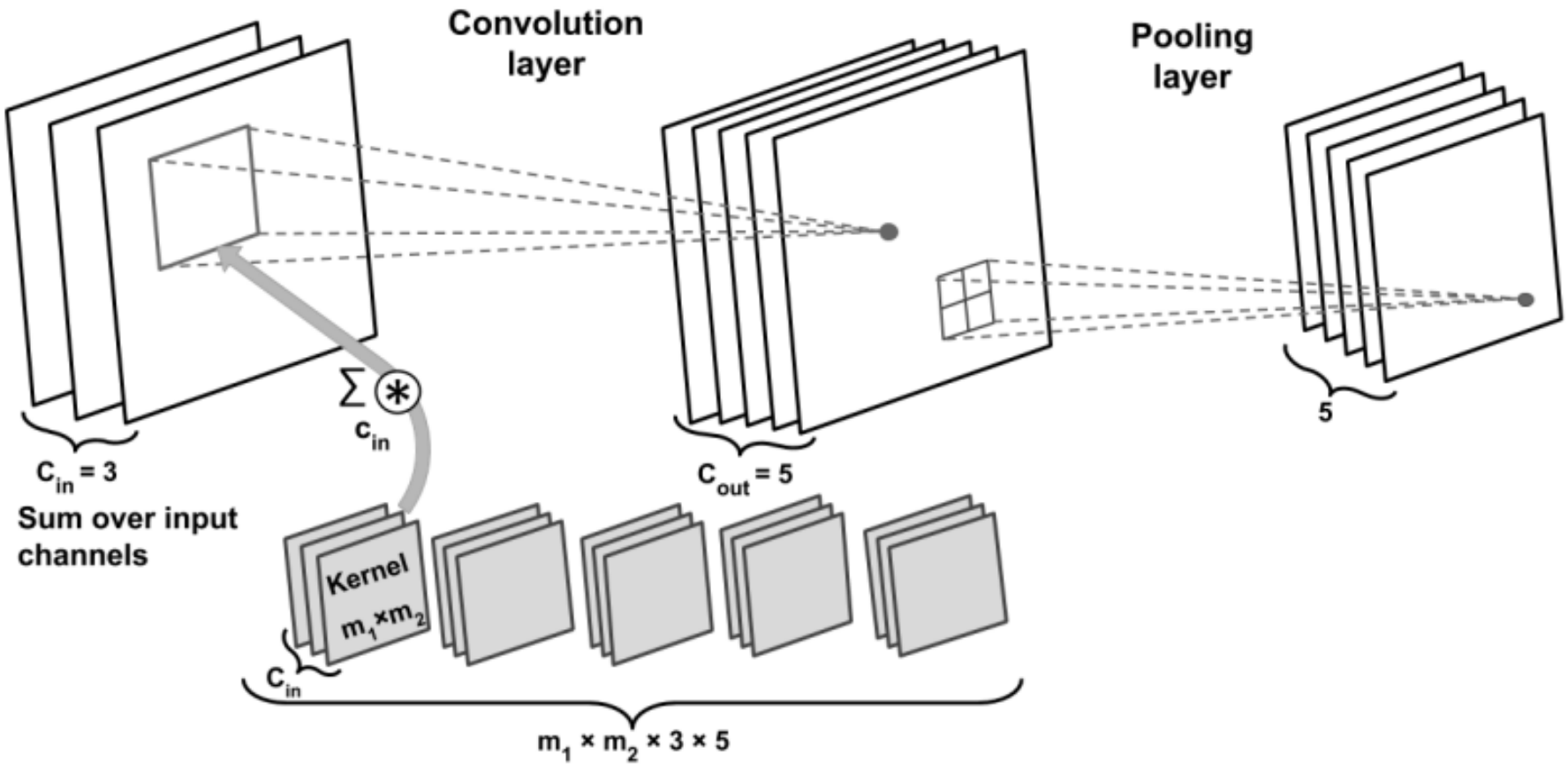
현재 모델의 입력으로 들어가는 x_t 는 시계열 데이터이다. 주식 데이터에서 사용할 정보의 종류 개수를 $feature$라고 하면 $t \times feature$ 크기의 행렬 형태로 들어오게 된다.



[그림 4] LSTM 입력 데이터의 예. [3]

이 데이터를 바로 LSTM에 넣어서 학습시키는 것 보다, 유용한 정보만 뽑아서 학습을 시키면 정확도가 더 올라갈 것이라는 생각을 하였다.

그래서 우리는 모델의 윗층에 Convolutional Layer를 추가하여 이 문제를 해결하고자 하였다. [3]

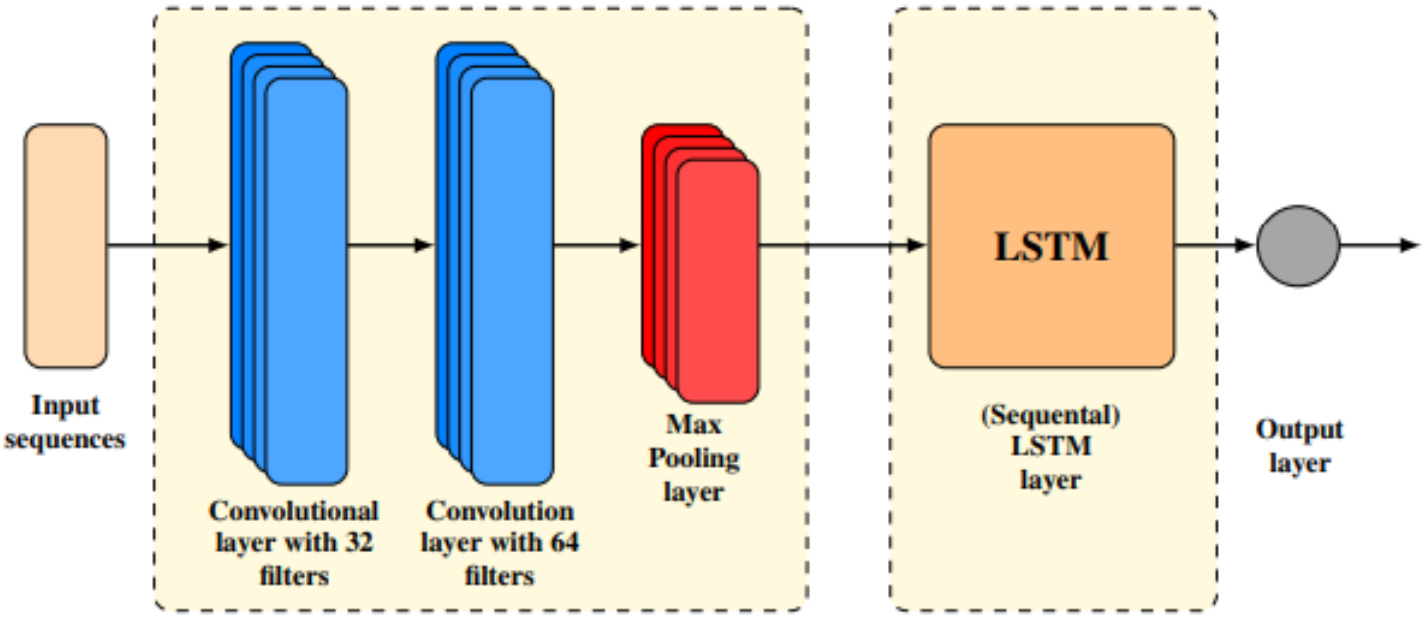


[그림 5] Convolutional neural network의 구조

Convolutional layer에서는 raw data와 filter(kernel)을 convolution 연산하여 데이터들의 상관관계를 뽑아낸다. 과제에서는 주식 데이터들의 시간을 기준으로 정보를 추출하기 위해 raw data를 작은 window size로 묶은 후, 행렬의 형태로 만든다. 이렇게 처리한 데이터는 각 feature들이 합성된 형태의 matrix로 나타내어진다. 이를 여러번 반복하여 raw data보다 일반적으로 더 유용한 데이터들만이 추출된다.

Convolutional layer를 통과한 데이터는 pooling layer를 통해 feature들을 더 낮은 차원의 행렬로 만들어준다. 이 과정을 통해 데이터들을 조금 더 밀집시킬 수 있고 이는 outlier data의 영향을 더 적게 하여 모델의 성능을 끌어올릴 수 있다.

최종적으로 CNN과 LSTM을 합친 모델은 다음과 같은 형태로 나타낼 수 있다.



[그림 6] CNN-LSTM의 모델 예제. [4]

2-3. 알고리즘 구현

구현에 사용된 라이브러리는 다음과 같다.

- **pandas**

  데이터 분석과 계산을 위한 오픈소스 툴이다.

- **tensorflow**

  구글에서 만든, 딥 러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공하는 라이브러리이다.

- **FinanceDataReader**

  금융 데이터를 읽어올 수 있는 오픈소스 라이브러리이다.

- **numpy**

  파이썬에서 각종 과학적인 계산을 할 수 있도록 도와주는 라이브러리이다.

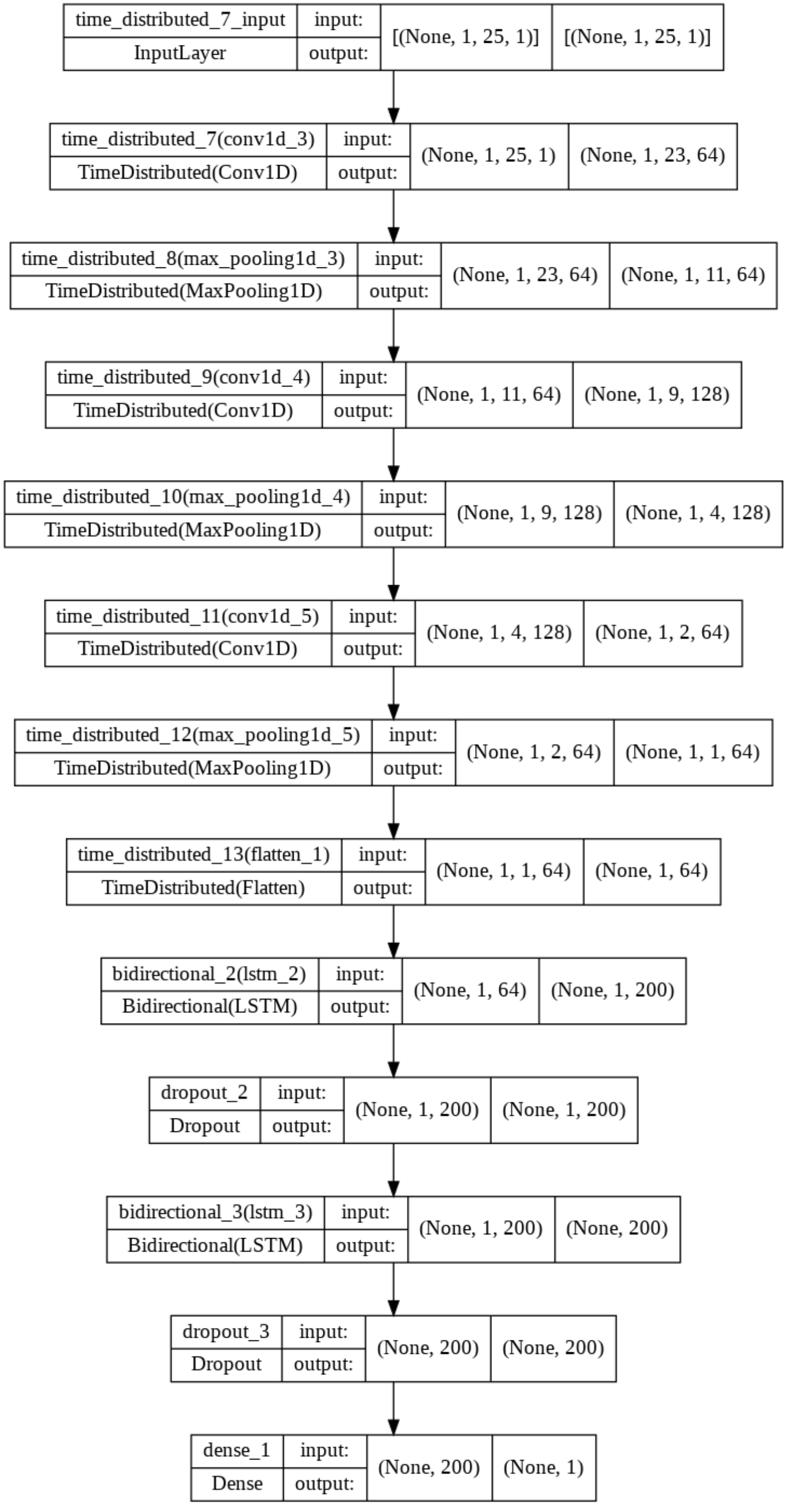
- **scikit-learn**

  데이터 분석 및 전처리를 위한 라이브러리이다.

- **metplotlib**

  데이터 시각화를 위한 라이브러리이다.

tensorflow를 사용하여 현재 구현된 모델의 요약본을 [그림 6]과 같이 나타내었다.


time_distributed_7_input | input: | [(None, 1, 25, 1)] | [(None, 1, 25, 1)]
InputLayer | output:
time_distributed_7(conv1d_3) | input: | (None, 1, 25, 1) | (None, 1, 23, 64)
TimeDistributed(Conv1D) | output:
time_distributed_8(max_pooling1d_3) | input: | (None, 1, 23, 64) | (None, 1, 11, 64)
TimeDistributed(MaxPooling1D) | output:
time_distributed_9(conv1d_4) | input: | (None, 1, 11, 64) | (None, 1, 9, 128)
TimeDistributed(Conv1D) | output:
time_distributed_10(max_pooling1d_4) | input: | (None, 1, 9, 128) | (None, 1, 4, 128)
TimeDistributed(MaxPooling1D) | output:
time_distributed_11(conv1d_5) | input: | (None, 1, 4, 128) | (None, 1, 2, 64)
TimeDistributed(Conv1D) | output:
time_distributed_12(max_pooling1d_5) | input: | (None, 1, 2, 64) | (None, 1, 1, 64)
TimeDistributed(MaxPooling1D) | output:
time_distributed_13(flatten_1) | input: | (None, 1, 1, 64) | (None, 1, 64)
TimeDistributed(Flatten) | output:
bidirectional_2(lstm_2) | input: | (None, 1, 64) | (None, 1, 200)
Bidirectional(LSTM) | output:
dropout_2 | input: | (None, 1, 200) | (None, 1, 200)
Dropout | output:
bidirectional_3(lstm_3) | input: | (None, 1, 200) | (None, 200)
Bidirectional(LSTM) | output:
dropout_3 | input: | (None, 200) | (None, 200)
Dropout | output:
dense_1 | input: | (None, 200) | (None, 1)
Dense | output:

[그림 7] 현재까지 진행된 모델 구현 도식

위 모델에 사용된 hyper parameter의 정보는 다음과 같다.

- `window size` : 25
- `batch size` : 32
- `epoch` : 40

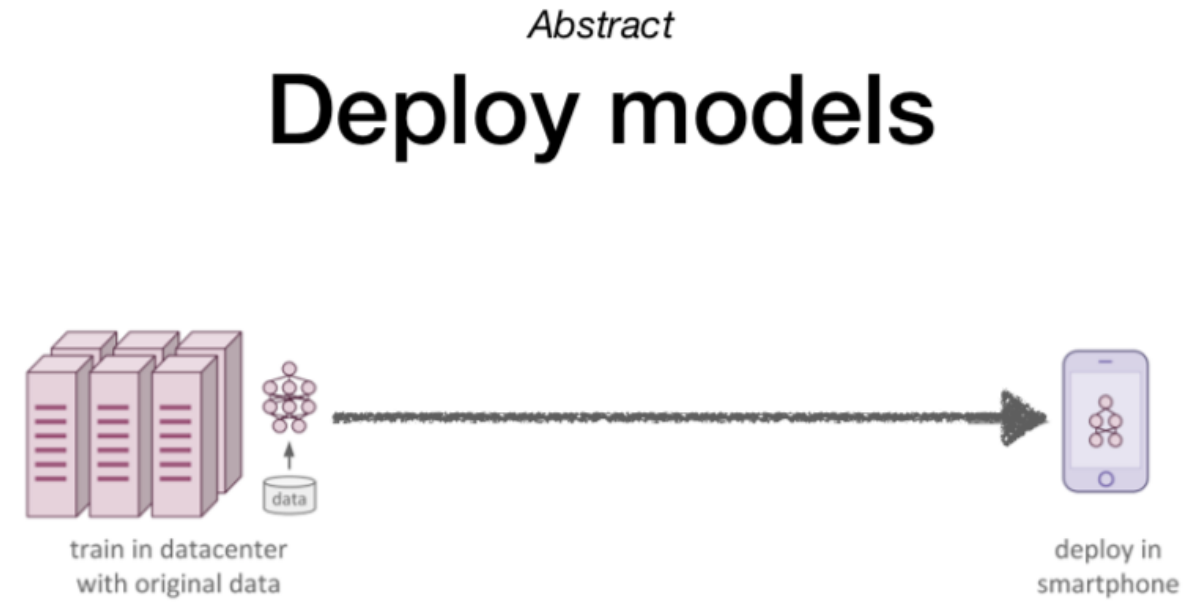
[그림 7]의 Input layer에서 input size가 `(None, 1, 25, 1)` 인 것을 확인할 수 있다. 이는 주식 데이터의 여러 feature 중에서 종가만 사용하기 때문이다.

이를 개선하기 위해 **tf.keras.layers.Conv1D**를 사용하여 만든 Convolutional layer를 **tf.keras.layers.Conv2D** 를 사용하여 만들고자 한다.

2-4. 경량화

경량화의 가장 큰 목적 중 하나는 바로 모델 배포이다. 우리가 주가 예측을 위한 딥러닝 모델을 개발하였다고 가정해보자. 이 모델은 다량의 주식 데이터를 통해 최대한 정확한 예측을 하도록 설계되었을 것이다.

하지만 이 모델을 배포한다고 생각했을 때, 모델이 배포한 모바일 장치는 이 복잡한 모델이 작동할 만큼 좋은 하드웨어가 아닐 것이다. [.]



[그림 8] 기존의 AI 배포 모델. [5]

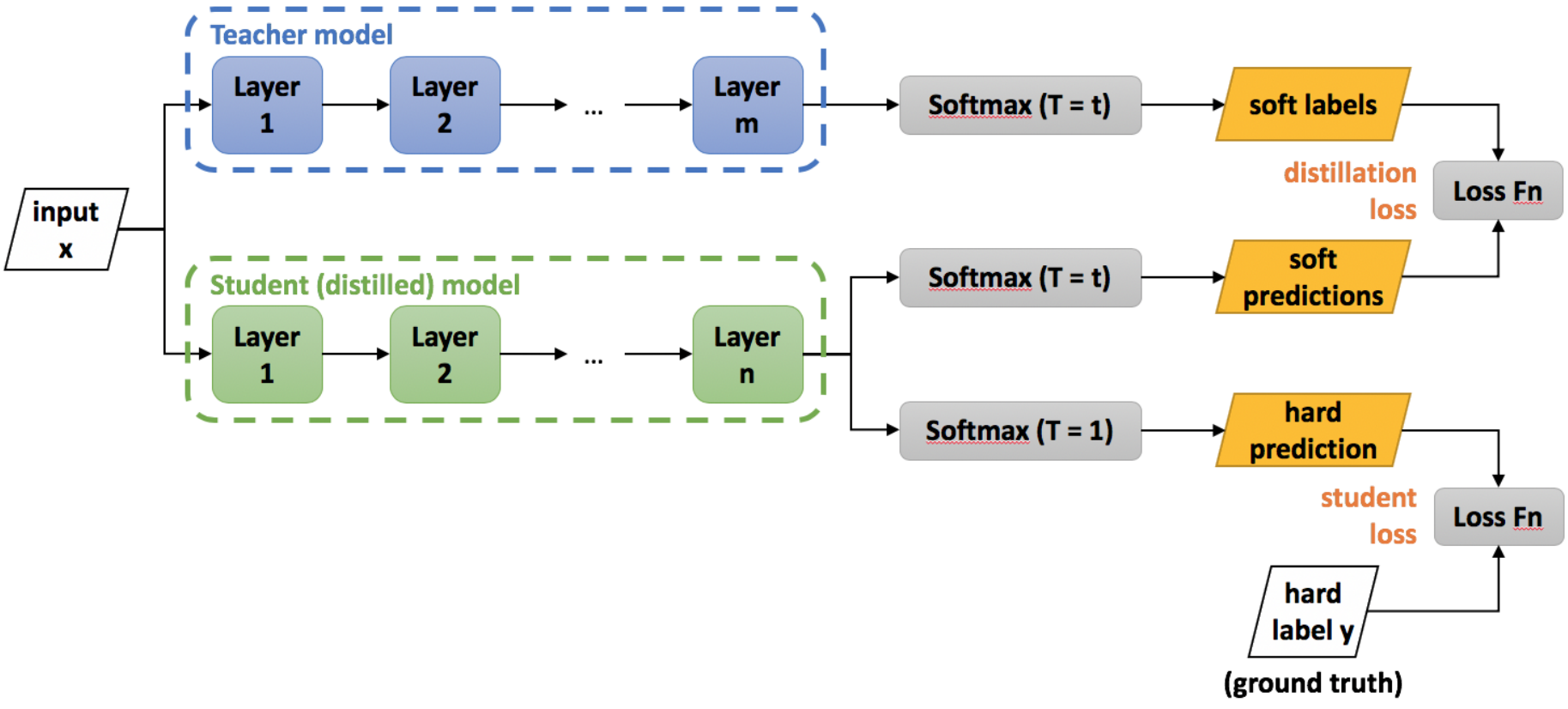
만약 다음 두 모델이 있다고 가정해보자.

- 모델 1 : 예측 정확도 99%, 예측 소요 시간 3시간
- 모델 2 : 예측 정확도 90%, 예측 소요 시간 3분

이 두 모델 중 하나를 사용하여 사용자들에게 서비스를 제공해야 한다면, 일반적인 사용자들을 타겟으로 할 때, 모델 1 보다 모델 2가 더 적합한 선택일 것이다.

하지만, 만약 이 두 모델을 동시에 활용할 수 있는 방법이 있지 않을까? 라는 생각을 할 수 있다. 이를 개념화한 것이 Knowledge Distillation 이다.

2-4-1. Knowledge Distillation



[그림 9] Knowledge Distillation의 예시. [6]

Knowledge distillation의 목적은 미리 잘 학습된 큰 네트워크인 **Teacher network**의 정보를, 실제 배포하고자 하는 모델인 **Student network**에 전달하는 것이다.

Teacher network의 결과를 Student network에 전달하고, Studnet network는 전달받은 정보를 모방하도록 학습시킨다. 이때, Loss Function의 parameter를 조절하여 Teacher network로부터의 정보를 얼마나 받아드릴건지 결정할 수 있다.

Loss Function의 수식은 다음과 같다.

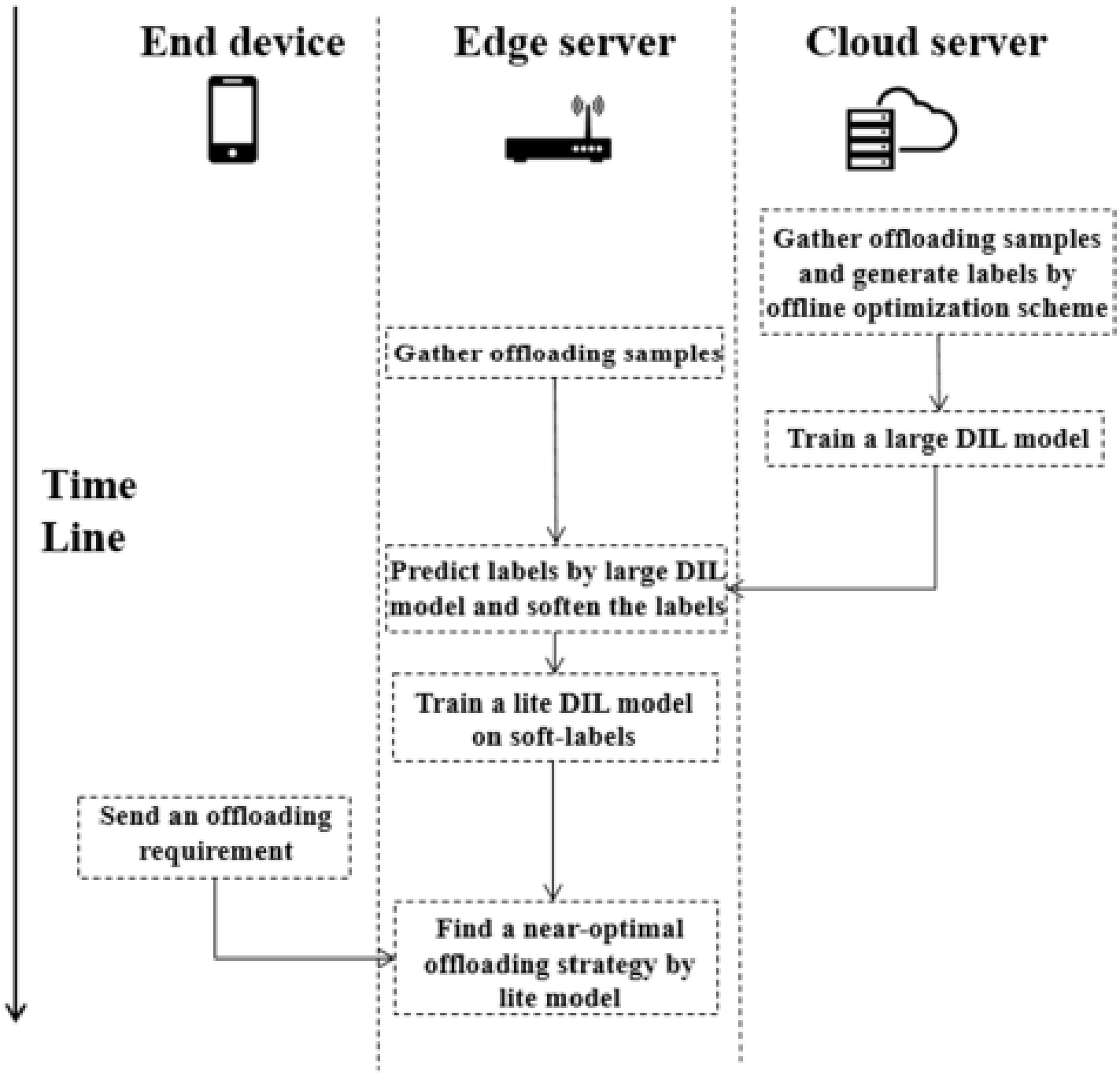
$$\mathcal{L}(x;W) = \alpha * \mathcal{H}(y, \sigma(z_s;T=1)) + \beta * \mathcal{H}(\sigma(z_t;T=\tau), \sigma(z_s,T=\tau))$$

- x : Input
- W: Student network의 parameter
- y : 정답 데이터
- $\mathcal{H}$: Cross-entropy loss function
- σ : 활성화 함수
- T : σ의 Temperature 계수
- α : hyper-parameter for studnet loss
- β : hyper-parameter for distillation loss

$\beta = 1 - \alpha$를 사용한다. [7] Hyper-parameter인 α를 조절하여 Student Model과 Teacher 모델간의 balance를 조절할 수 있다. Hinton et al., 2015 에 따르면 α를 β보다 훨씬 작게 설정하는 것이 최적의 결과를 만들었다고 한다.

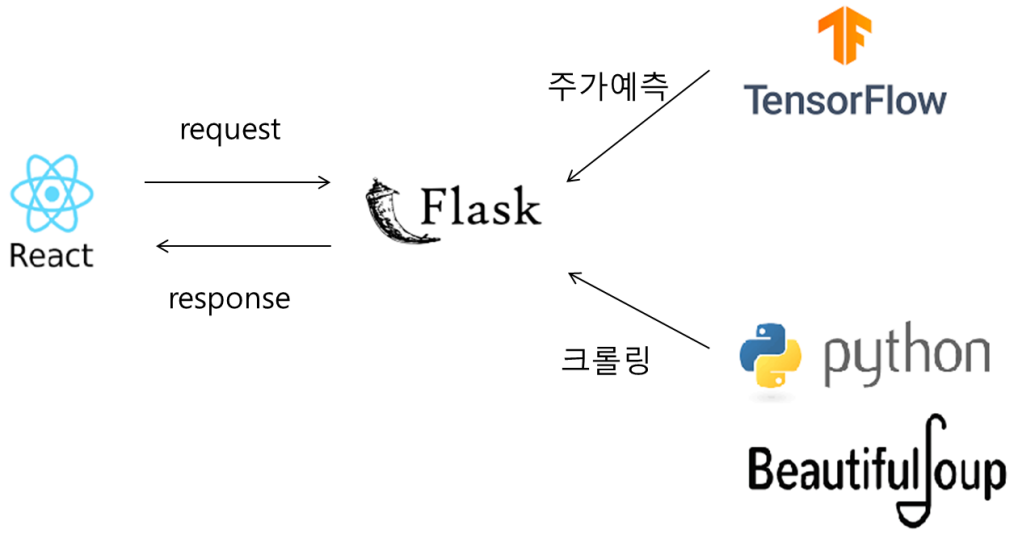
σ는 보통 $SoftMax$ 를 사용한다. 이때, T라는 parameter를 통해 output으로 나오는 값들의 scale을 조절할 수 있다.

다음은 Knowledge Distillation을 통해 만들 수 있는 서비스 구조의 예시이다.



[그림 10] Knowledge Distillation을 사용하는 서비스의 flowchart 예시 [8]

2-5. Back-end



[그림 11] 현재까지의 back-end 구조 도식

- Flask를 이용한 API server 구축
- FinanceDataReader를 이용하여 주가 및 주요지수 불러오기
- 관련기사 및 기업이슈 불러오기
- tensorflow를 이용한 주가예측 제공

Flask

Micro Web Framework, 간단한 API 서버를 만드는 데에 특화 되어있는 **Python Web Framework**이다. Django는 다량의 기능을 미리 제공하는 반면 Flask는 기본적인 기능만 제공하고, 특별한 도구나 라이브러리가 필요 없기 때문에 '마이크로 웹 프레임워크'라고도 한다.

```
import flask from Flask
app=Flask(__name__)

@app.route('/')
def fnc():
  return 'Hello'
```

- `app = Flask(__name__)` : flask application을 생성하는 코드
- `__name__` : file이 실행될때 실행되는 모듈이 담기는 변수 즉, 여기서는 main.py가 담긴다.
- `@app.route` ()안의 주소에 접속하면 그 바로 아랫줄에 있는 함수를 호출하는 flask의 decorator 이다.

딥러닝 모델을 API서버에 추가하여 예측정보를 제공하여 시각화할 예정이다. 따라서, flask에서 Rest API를 구현하기 위해 Response를 `json` 형태로 작성해야 한다.

이때, `pandas.DataFrame` 형식의 모델 예측값을 1차원 list로 변환해주고 jsonify를 이용하여 json형태로 반환하는 작업을 해주었다.

사용 라이브러리

- **Beautifulsoup**

웹 페이지의 정보를 쉽게 스크랩할 수 있도록 기능을 제공하는 라이브러리.

- **requests**

HTTP 요청을 보낼 수 있도록 기능을 제공하는 라이브러리.

2-6. front-end

[그림 12] 현재까지의 front-end 구조 도식

- React.js 를 기반으로 구현
- 코스피 종목 종류 확인할 수 있는 페이지 구현
- 해당 종목 가격, 관련기사 및 기업이슈 제공하는 페이지 구현
- 딥러닝 모델을 이용하여 가격예측 하는 페이지 구현

사용 라이브러리

- **Styled-Components**

기존 돔을 만드는 방식인 css, scss 파일을 밖에 두고, 태그나 id, class이름으로 가져와 쓰지 않고, 동일한 컴포넌트에서 컴포넌트 이름을 쓰듯 스타일을 지정하는 것을 styled-components라고 부른다.css 파일을 밖에 두지 않고, 컴포넌트 내부에 넣기 때문에, css가 전역으로 중첩되지 않도록 만들어주는 장점이 있다.

- **React-Hook(useState, useEffect,…)**

React 에서 기존에 사용하던 Class를 이용한 코드를 작성할 필요 없이,

state와, 여러 React 기능을 사용할 수 있도록 만든 라이브러리

hook을 사용해 함수형 컴포넌트에서도 state와 생명주기를 다룰 수 있기에 **클래스형 컴포넌트에서만 가능하던 상태관리를 더 손쉽게 할 수 있어 필요하다**

- **React Query(사용예정)**

서버의 값을 클라이언트에 가져오거나, 캐싱, 값 업데이트, 에러핸들링 등 비동기 과정을 더욱 편하게 하는데 사용.

- ~~**Recoil(사용예정)**~~

Recoil은 atoms(공유 상태) => selectors(순수 함수) => React Component로 흘러가는 data-flow graph 구조이다.
Atom은 Recoil의 단위 데이터이며 컴포넌트가 구독할 수 있는 상태의 단위이다.

atom이 업데이트되면 해당 atom을 구독하고 있는 컴포넌트가 업데이트된 값을 기반으로 리렌더링 된다.

앞으로의 목표

- 기존의 사이트들에 대한 디자인 및 정보 수집
- 웹 사이트 상세 디자인 및 구현
- 추가 기능 모색 및 구현
- 백엔드 서버와의 링크 및 정보 처리

3. 과제 계획

개발 일정

Aa 프로세스	일정	진행상황
데이터 종류 선정	@2022년 5월 23일 → 2022년 7월 12일	완료
경량화할 수요 예측 알고리즘 조사	@2022년 5월 23일 → 2022년 8월 9일	완료
알고리즘 구현	@2022년 7월 12일 → 2022년 8월 9일	진행중
중간 보고서 작성	@2022년 7월 28일 → 2022년 8월 1일	진행중
백앤드 개발	@2022년 7월 12일 → 2022년 8월 31일	진행중
프론트 앤드 개발	@2022년 7월 12일 → 2022년 8월 31일	진행중
알고리즘 경량화 작업	@2022년 8월 8일 → 2022년 8월 31일	시작전
서비스 테스트	@2022년 9월 1일 → 2022년 9월 30일	시작전
최종 보고서 작성	@2022년 9월 1일 → 2022년 9월 30일	시작전

타임라인

4. 과제 진행 내용

4-1. 구성원별 진행 현황

이름	진행 현황
서준호	**알고리즘 조사 및 구현 -** 주식 데이터를 가져오고, 모델에 맞게 데이터를 전처리 - 기본적인 CNN + LSTM 모델 구현 완료 - 예측 값들이 Real data에 비해 뒤쳐지는 경향이 있어, hyper-parameter들을 조정하고, 다양한 layer를 쌓아보며 개선 작업 진행
윤정현	**프론트 개발** - 종목 상세정보 페이지 디자인 - 종목 종류 보여주는 메인페이지 디자인 **백엔드 개발** - 관련기사, 기업정보 크롤링 - 라이브러리를 이용하여 종목별 가격 가져오기
김민태	**프론트 개발** - 웹서비스 형태의 사이트 디자인 구현 - 백엔드와 프론트엔드의 프록시 서버를 통한 연결

4-2. 현재까지의 상세 진행 내용

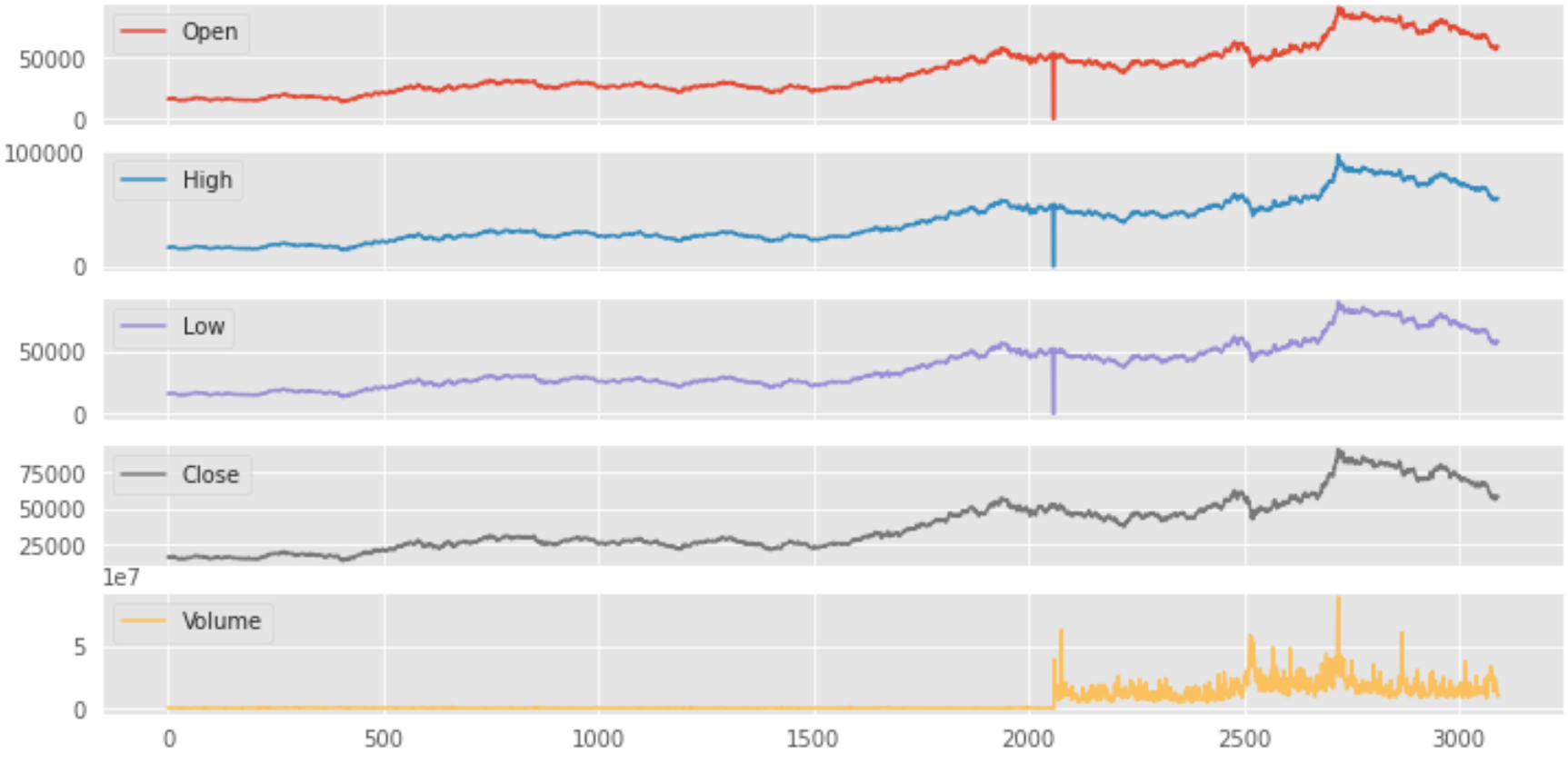
4-2-1. 데이터 전처리

`FinanceDataReader` 라이브러리를 통해 주식 정보를 한번에 불러올 수 있다.

```
import FinanceDataReader as fdr

data = fdr.DataReader('005930', '2010-01-01')
data.head()
```

Date	Open	High	Low	Close	Volume	Change
@2010년 1월 4일 오전 12:00	16060	16180	16000	16180	239271	0\.012515644555694 65
@2010년 1월 5일 오전 12:00	16520	16580	16300	16440	559219	0\.016069221260815 78
@2010년 1월 6일 오전 12:00	16580	16820	16520	16820	459755	0\.023114355231143 47
@2010년 1월 7일 오전 12:00	16820	16820	16260	16260	443237	-0\.033293697978596 94
@2010년 1월 8일 오전 12:00	16400	16420	16120	16420	295798	0\.009840098400984 099

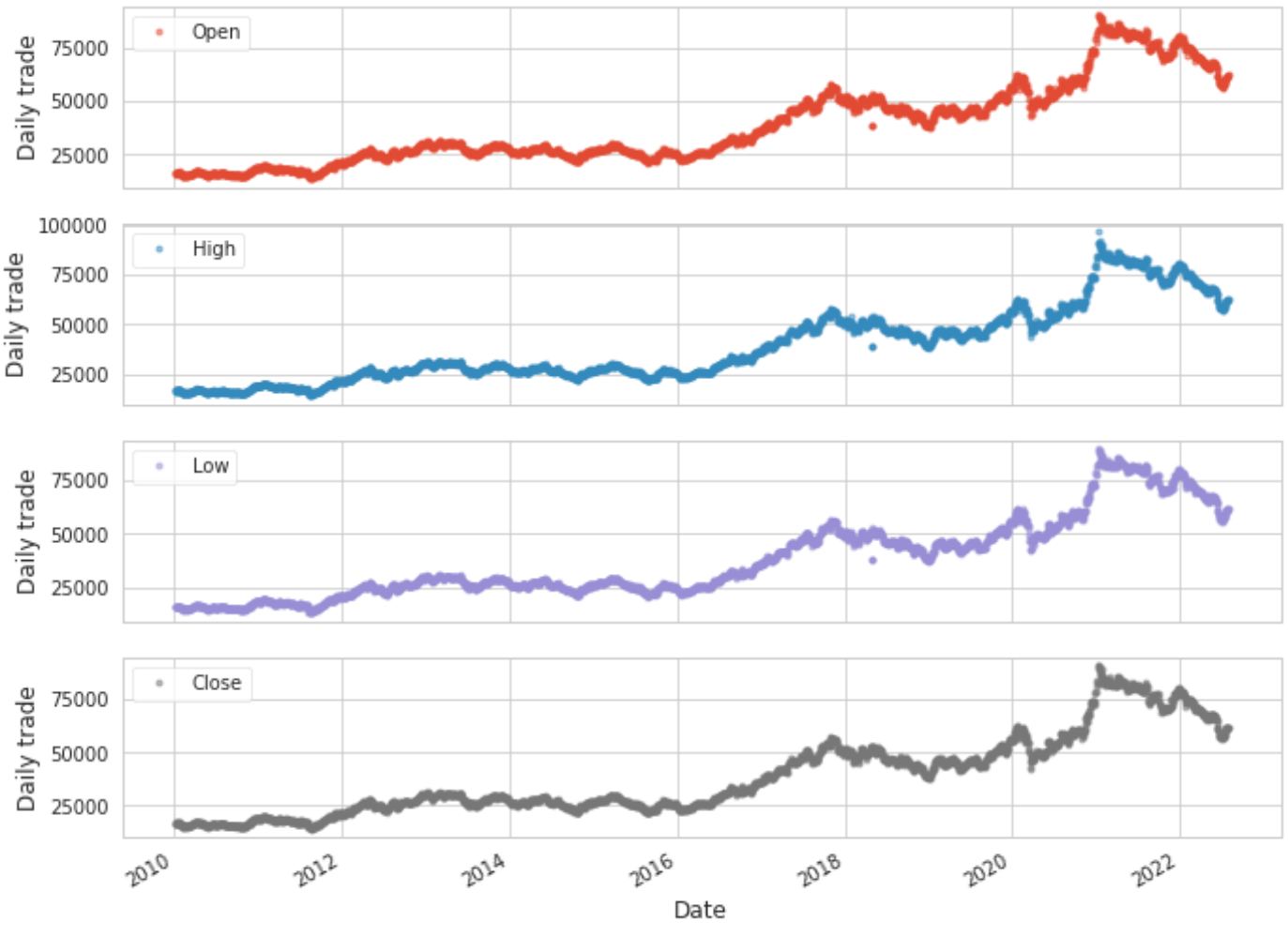
여기서 `Change` 는 사용하지 않는다.



[그림 13] 주식 데이터의 feature 별 분포 그래프

데이터 분포를 보면 `0`으로 뚝 떨어지는 것 있다. 이 데이터들은 outlier data로 데이터의 밀집도를 해친다. 따라서 이를 없애기 위한 작업을 진행한다.

```
def erase_zero(data):
  data.replace(0, np.NaN, inplace=True)
  data.fillna(data.mean(), inplace=True)
  return data
```

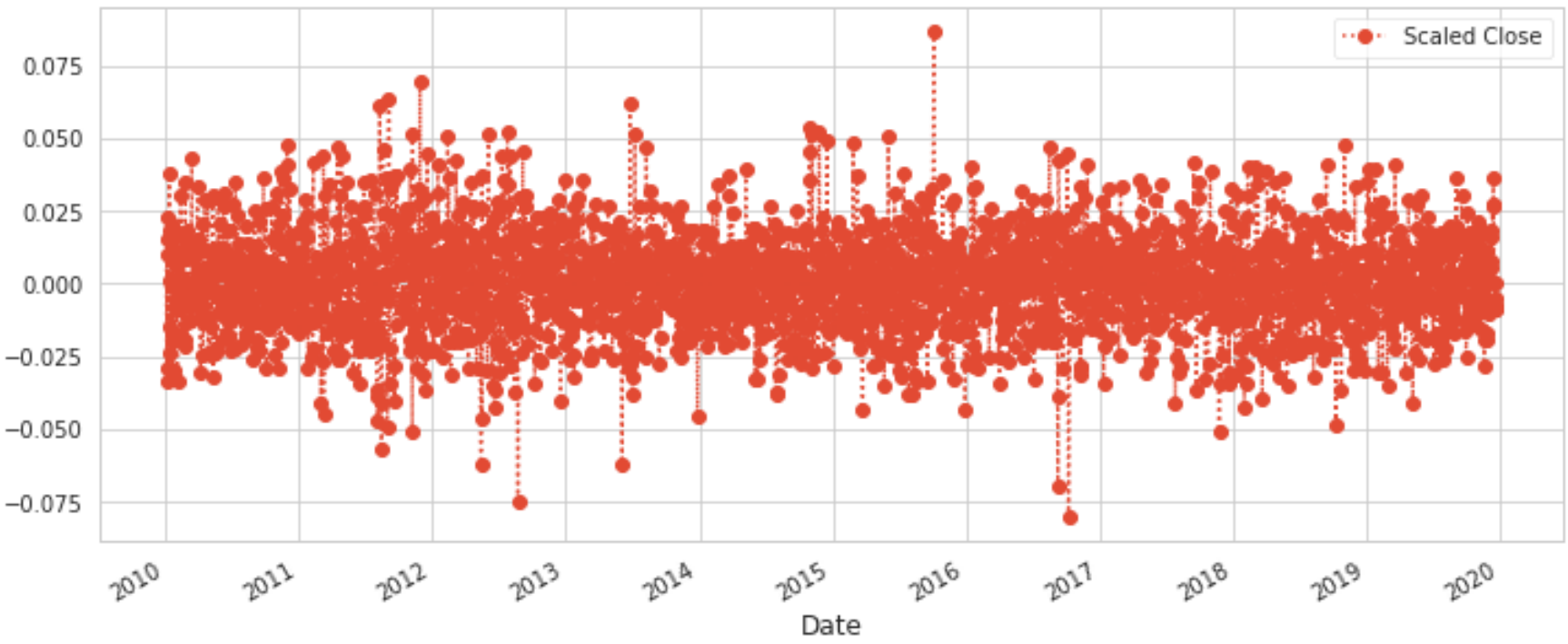


[그림 14] 수정 후 데이터 feature 별 그래프

데이터들이 전체적으로 큰 값들이기 때문에 오차가 크게 발생할 수 있다. 따라서 데이터의 스케일을 조정해주는 작업을 한다.

그리고, [그림 7]과 같이 데이터의 입력을 맞추기 위해 데이터를 나누어 준다.

```
def create_window_set(df, column, window_size):
  X = []
  Y = []
  for i in range(1 , len(df) - window_size -1 , 1):
    first = df.iloc[i,column]
    temp = []
    temp2 = []

    for j in range(window_size):
        temp.append((df.iloc[i + j, column] - first) / first)
    temp2.append((df.iloc[i + window_size, column] - first) / first)

  X.append(np.array(temp).reshape(window_size, 1))
  Y.append(np.array(temp2).reshape(1, 1))

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=False)

train_X = np.array(x_train)
test_X = np.array(x_test)
train_Y = np.array(y_train)
test_Y = np.array(y_test)

train_X = train_X.reshape(train_X.shape[0],1,window_size,1)
test_X = test_X.reshape(test_X.shape[0],1,window_size,1)

return train_X, train_Y, test_X, test_Y
```



[그림 15] Scale 처리 한 후, 종가(Close)에 대한 그래프

4-2-2. 알고리즘 구현

[그림 7]의 알고리즘을 구현한 코드이다.

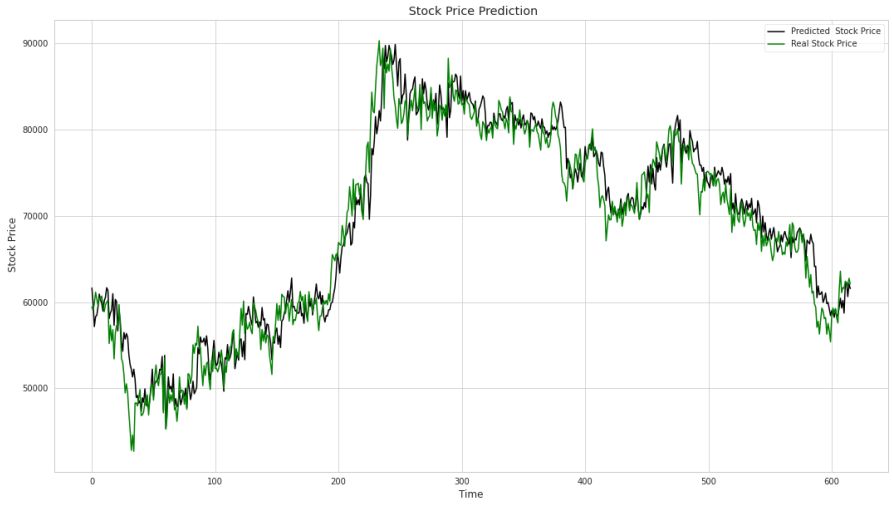
```
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Bidirectional, TimeDistributed
from tensorflow.keras.layers import MaxPooling1D, Flatten
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.metrics import Accuracy
from tensorflow.keras.metrics import RootMeanSquaredError

def build_model(window_size):
  model = tf.keras.Sequential()

  # CNN layers
  model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu', input_shape=(None, window_size, 1))))
  model.add(TimeDistributed(MaxPooling1D(2)))
  model.add(TimeDistributed(Conv1D(128, kernel_size=3, activation='relu')))
  model.add(TimeDistributed(MaxPooling1D(2)))
  model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu')))
  model.add(TimeDistributed(MaxPooling1D(2)))
  model.add(TimeDistributed(Flatten()))

  # LSTM layers
  model.add(Bidirectional(LSTM(100, return_sequences=True)))
  model.add(Dropout(0.5))
  model.add(Bidirectional(LSTM(100, return_sequences=False)))
  model.add(Dropout(0.5))

  #Final layers
  model.add(Dense(1, activation='linear'))
  model.compile(optimizer='adam', loss='mse', metrics=['mse', 'mae'])
  return model
```

4-2-3. 실행 결과

4-2-3. 에서 구현한 모델에 4-2-2.의 데이터를 넣어 학습시켜보았다. 이때, hyper-parameter를 다르게 설정하여 총 4개의 결과를 얻었다.

[그림 16] window size : 25, batch : 64, epoch : 40

[그림 17] window size : 50, batch : 64, epoch : 40



[그림 18] window size : 25, batch : 32, epoch : 40

[그림 19] window size : 25, batch : 128, epoch : 40

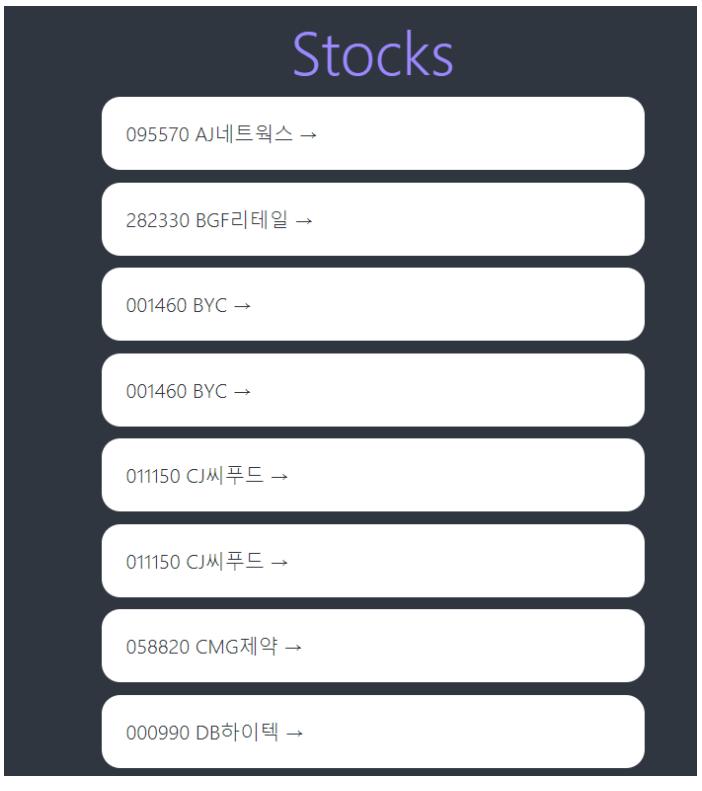
다양한 hyper-parameter들을 설정하면, 어떤 모델이 가장 주가를 빠르게 반영할 수 있는지 비교할 수 있다.

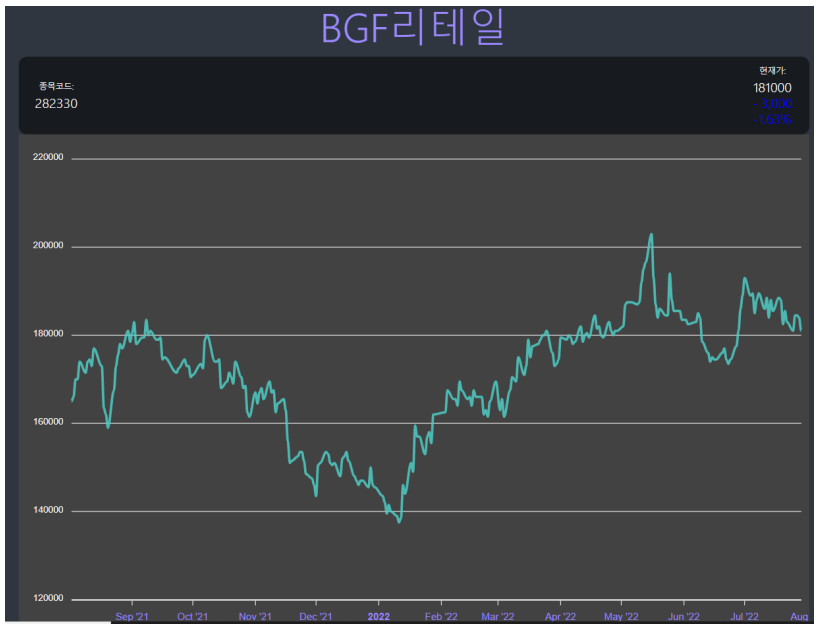
[그림 16]과 [그림 17]는 `window size`를 조정한 결과를 나타낸다 . `window size` 가 큰 경우, 그래프가 전체적으로 우측으로 밀리는 경향을 보였다. 이는 미래의 가격을 예측하는 속도가 상대적으로 느리다는 것을 의미하므로 모델의 성능이 좋지 않음을 것을 의미한다.

`window size`가 커질 수록, 주식 데이터를 길게 보는 것과 같기 때문에 다음과 같은 결과가 나왔다고 추측된다. 예를 들어, 주식 차트를 분석할 때, 하루를 기준으로 보는 것과 일주일 기준으로 보는 것은 서로 다른 목적을 가진다. 주식 차트의 스케일을 작게 하면 할 수록, 짧은 미래의 주가를 예측할 수 있고, 스케일이 커지면 커질 수록 주가의 큰 흐름을 파악하기 좋을 것이다.

이 특성이 학습에 반영이 되었을 것이다. 이를 통해, 내일의 주가를 예측하기 위해서는 작은 `window size`를 가지는 것이 유리하고, 먼 미래의 주가를 예측하기 위해서는 `window size`를 크게 가지는 것이 유리함을 알 수 있었다.

4-2-4. page 디자인 및 웹 크롤링





[그림 21] 상세정보 페이지

[그림 20] 메인 페이지

```
useEffect(() => {
    // 종목 가격 불러오기 1.날짜 2.시가 3.고가 4.저가 5.종가
    fetch(`/api/info/${stockId}`)
      .then((res) => res.json())
      .then((data) => {
        setstockData(data.data);
      });
  }, []);
```

메인페이지에서 종목명을 누르면 상세정보 페이지가 렌더링되면서 `/api/info/` 종목코드로 `request`를 보낸다.

```
@app.route('/api/info/<co>')
def chart(co):
    df = stock.get_market_ohlcv_by_date(last_year, today, co)
    js = df.to_json(orient='table', force_ascii=False ,date_format='iso')
    #print(df)
    #print(js)
    return js
```

Flask서버는 요청을 받으면 데이터프레임 형식의 주가 정보를 `json`형태로 변환 후 response 해준다.

	시가	고가	저가	종가	거래량
날짜					
2021-08-02	163000	165000	161500	165000	27305
2021-08-03	163000	166500	162000	166000	18501
2021-08-04	164000	171000	164000	170000	46309
2021-08-05	170000	171000	168500	170000	25632
2021-08-06	173000	177500	172000	174000	62355
...	...	...	...	...	...
2022-07-25	182500	184000	179500	181000	19952
2022-07-26	181000	184500	179000	184500	24944
2022-07-27	184500	186500	181000	184500	15580
2022-07-28	183000	187500	182500	184000	23340

[그림 22] 주가 요청에 따른 response 메세지

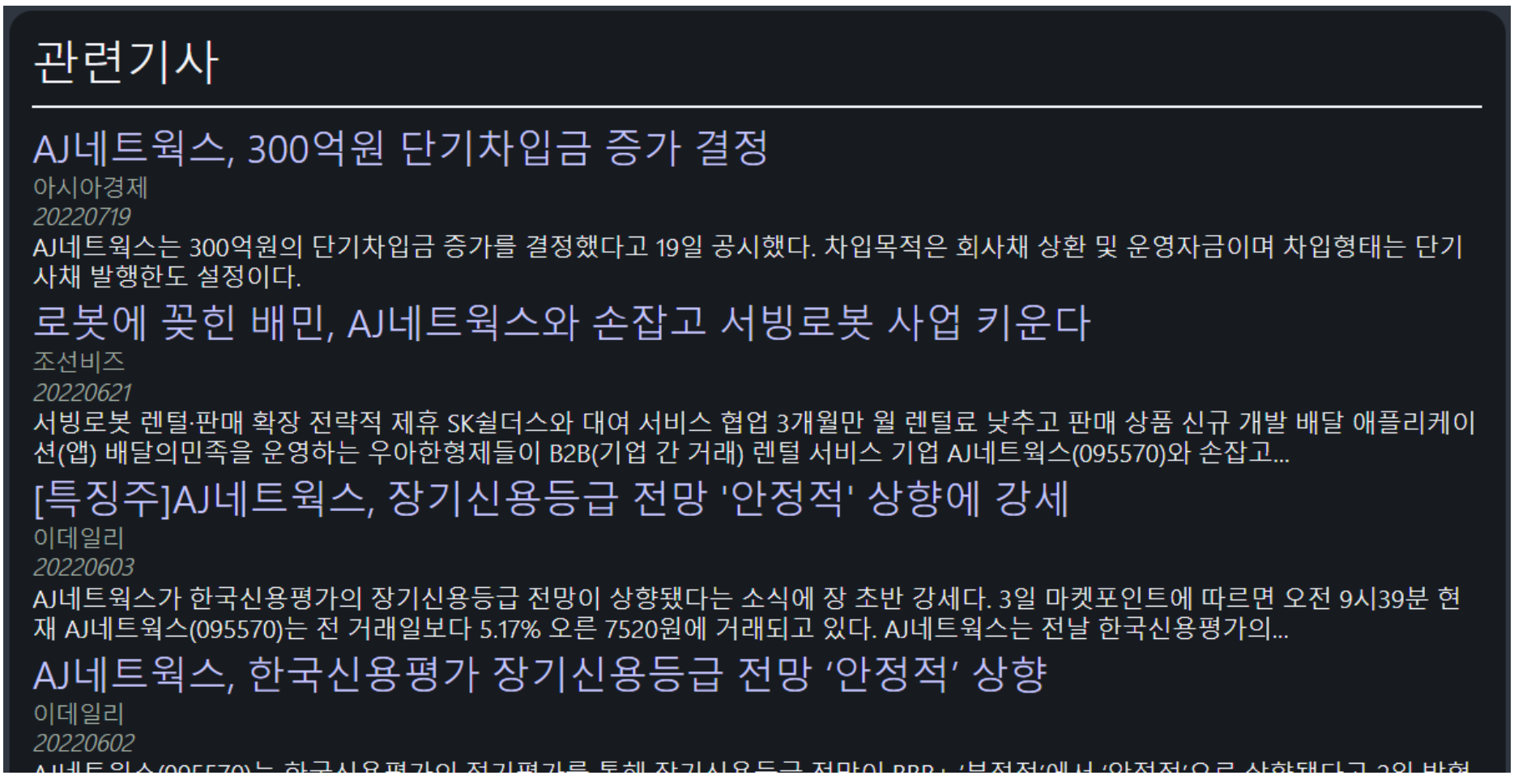
```
<div>
            <ApexChart
              type="line"
              series={[
                {
                  name: "sales",
                  data: stockData?.map((price) => price.종가),
                },
```

```
            ]}
            options={{
              theme: {
                mode: "dark",
              },
              chart: {
                toolbar: {
                  show: false,
                },
                height: 500,
              },
              grid: {
                show: true,
              },
              stroke: {
                curve: "smooth",
                width: 3,
              },
              xaxis: {
                type: "datetime",
                categories: stockData?.map((price) => price.날짜),
                labels: {
                  style: {
                    colors: "#9c88ff",
                  },
                },
              },
            }}
          />
        </div>
```

주가를 시각화 하기 위해 `ApexChart`라는 라이브러리를 사용하였다

선 그래프, 거품형, 타임라인, 히트맵까지 다양한 스타일들이 있고, 반응형에 커스텀 기능까지 구현되어 있다.

데이터는 `FinanceDataReader`를 사용하였고 종가를 기준으로 하였다.



[그림 23] 기사 정보를 제공하는 페이지

```
@app.route('/api/article/<co>')
def article(co):
    tot_list = []

    for p in range(1):
        # 뉴스 기사 모인 페이지
        url = 'https://m.stock.naver.com/domestic/stock/' + str(
            co) + '/news/title'  # https://m.stock.naver.com/domestic/stock/003550/total
        # F12누르면 나오는 네트워크상에서 찾아온 경로
        # https://m.stock.naver.com/api/news/stock/005930?pageSize=20&page=1&searchMethod=title_entity_id.basic
        url = "https://m.stock.naver.com/api/news/stock/" + str(
            co) + "?pageSize=5&searchMethod=title_entity_id.basic&page=1"
        res = requests.get(url)

        news_list = json.loads(res.text)
        # 페이지에서 가져온 전체 뉴스기사를 for문으로 분리
        # print(news_list[0])
        for i, news in enumerate(news_list):
            # 신문사 id
            a = news['items'][0]['officeId']
            # 기사 id
            b = news['items'][0]['articleId']
            list = []
            list.append(news['items'][0]['officeName'])  # 신문사
            list.append(news['items'][0]['datetime'][:8])  # 날짜
            list.append(news['items'][0]['title'].replace('&quot;', '\"'))  # 제목
            list.append(news['items'][0]['imageOriginLink'])  # 이미지
            list.append(news['items'][0]['body'].replace('&quot;', '\"'))  # 기사 내용
            list.append('https://m.stock.naver.com/domestic/stock/005930/news/view/' + str(a) + '/' + str(b))  #
기사 url
            tot_list.append(list)

    news_df = pd.DataFrame(data=tot_list, columns=['offname', 'rdate', 'title', 'imgsrc', 'content', 'url'])
    news_df['title'] = news_df['title'].str.replace('&amp;', '&')
    news_df['content'] = news_df['content'].str.replace('&amp;', '&')
    json_str = news_df.to_json(orient="records",force_ascii=False)
    print(json_str)
    return json_str
```

React에서 `/api/article/`종목번호로 `request`를 보내면 flask서버에서 크롤링을 수행한다.

크롤링에는 `beautifulsoup`라이브러리를 사용하였고 url은 네이버 증권을 이용하였다.

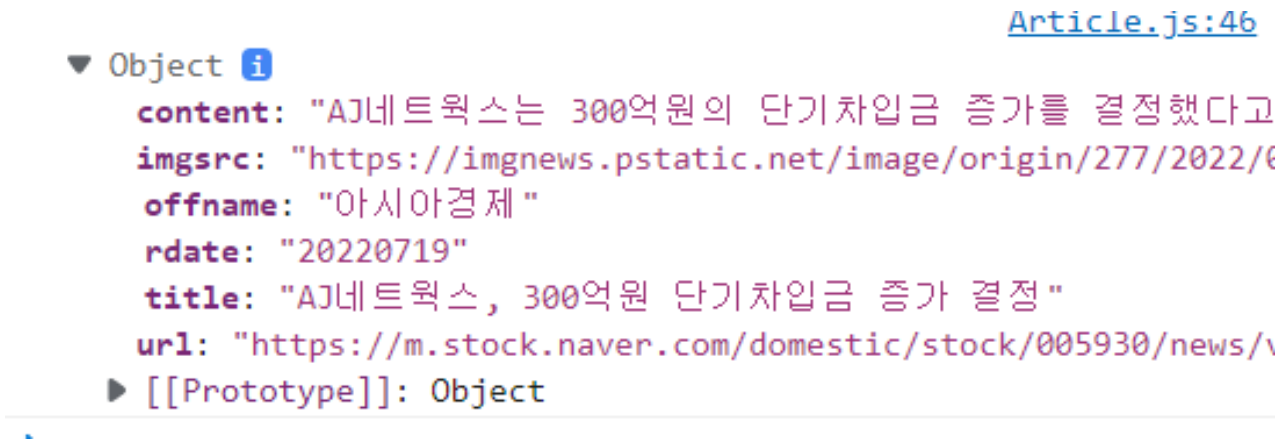
해당 url을 가져오기 위해 개발자도구(f12)를 사용하였고,

```
https://m.stock.naver.com/api/news/stock/005930?pageSize=20&page=1&searchMethod=title_entity_id.basic
```

해당 URI 에 `GET` 방식으로 Parameter 들을 서버에 요청하여 정보를 얻고 있다.

1. `officeName`: 신문사
2. `datetime`: 날짜
3. `title`: 제목
4. `imageOriginLink`: 이미지링크
5. `body`: 기사내용

해당 정보를 `json` 형태로 반환해주었다.



[그림 24] 크롤링을 통해 웹에서 reponse를 받아온 화면

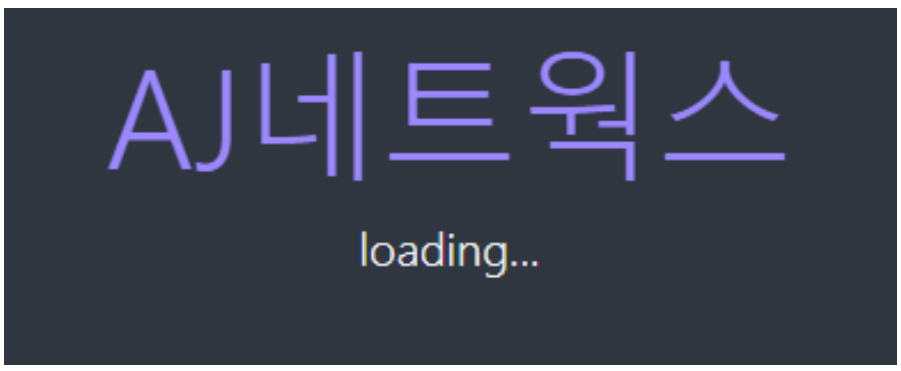
4-2-5. 프록시 서버 에러 수정 및 백엔드 서버 연결

```
Invalid options object. Dev Server has been initialized using an options object that does not match the API sche
ma.

- options.allowedHosts[0] should be a non-empty string.
```

프록시 서버를 이용해 백엔드 정보를 프론트로 가져올 때 위 문구의 에러가 계속해서 뜨거나 리엑트가 실행이 되어도 주식을 선택하여 클릭 시 로딩창에서 넘어가지 않고 백엔드 데이터를 불러오지 못하는 등의 문제를 보였다.

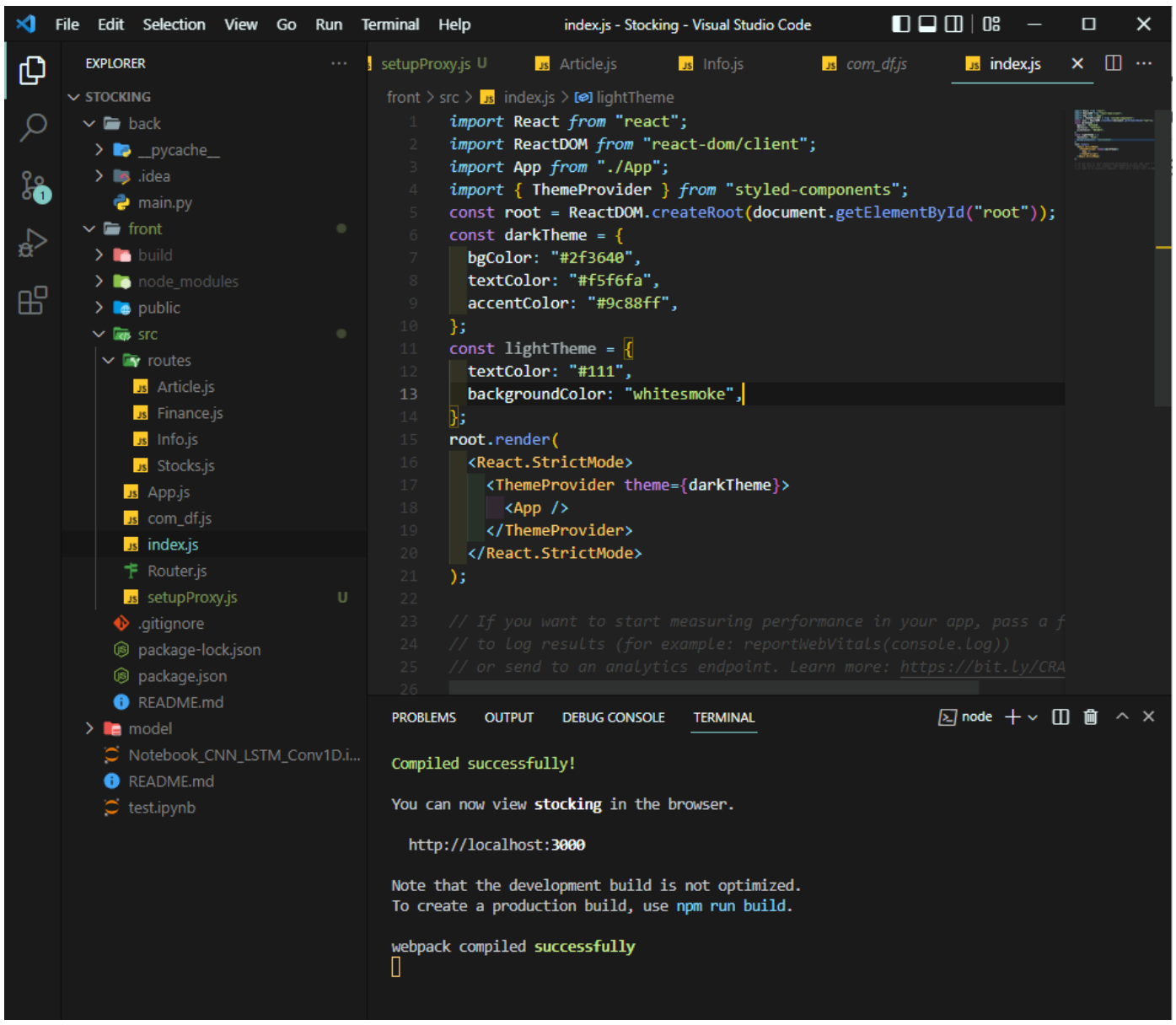
문제를 해결하기 위해 서칭을 해보고 난 후 모든 request 에 `/api` 를 붙임으로써 이 에러를 해결하여 리엑트에서 주식 정보 그래프 및 관련기사를 불러오는데 성공하였다.

처음 파일을 돌렸을때 백엔드 데이터를 프런트로 가져오는데 지속적으로 오류가 생겼다.

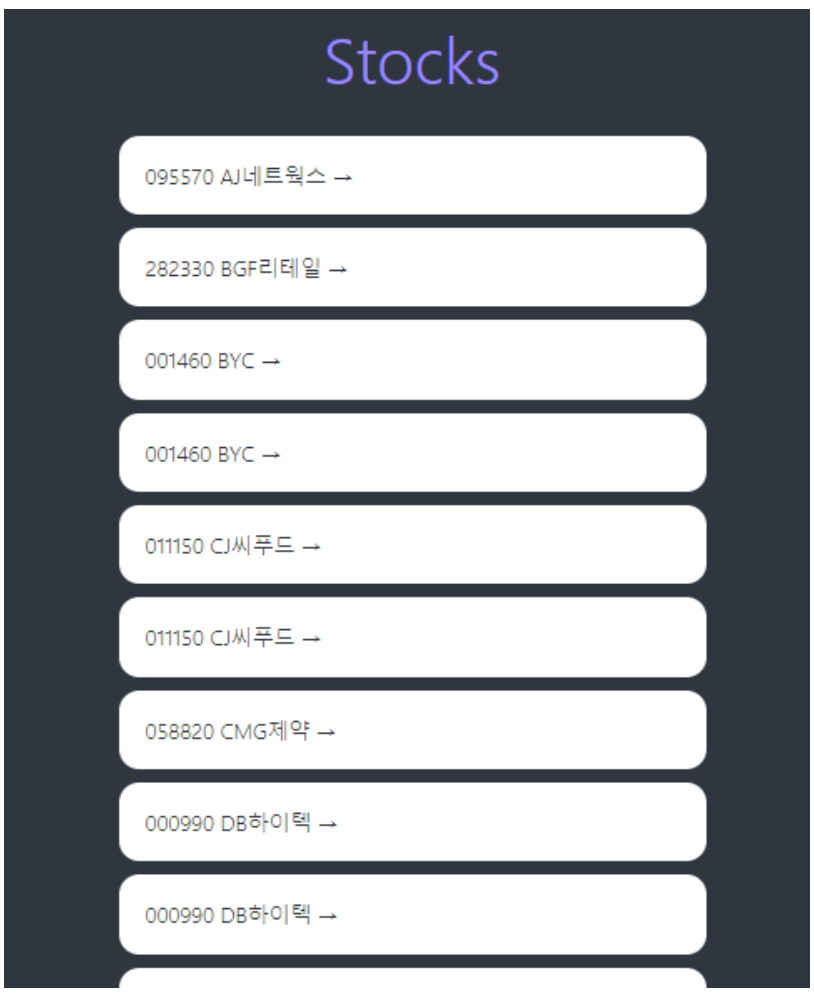


[그림 25] 로딩창에서 넘어가지 않는 에러

위와같이 계속해서 로딩화면에서 넘어가지 않거나 터미널에서 맨위의 문구와 같은 에러 메시지가 뜨고 아예 리엑트가 실행되지 않는등의 문제가 있었다. 그래서 구글링을 해본 결과 프록시 서버의 문제로 밝혀져서 모든 request에 `/api`를 붙여주고 난 후 다시 터미널을 돌리니까 정상적으로 작동하였다.



[그림 26] 오류 수정 작업 화면



[그림 27] 정상적으로 작동하는 페이지

이처럼 데이터를 가져올때 해당 사이트에서 바로 자신의 pc로 가져오는 것이 아니라 임시저장소를 거쳐오기 때문에 프록시 서버의 request에 `/api`를 추가시켜 주어서 가져오는 데이터를 명시해 주었더니 오류가 해결되는 것을 볼 수 있었다.

5. Reference

[1] http://www.aitimes.com/news/articleView.html?idxno=136379

[2] https://en.wikipedia.org/

[3] https://swlock.blogspot.com/2019/04/keras-lstm-understanding-input-and.html

[4] https://doi.org/10.1007/s00521-020-04867-x

[5] 딥러닝 모델 지식의 증류기법, Knowledge Distillation | Seongsu (baeseongsu.github.io)

[6] https://intellabs.github.io/distiller/knowledge_distillation.html

[7] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. Distilling the Knowledge in a Neural Network. https://arxiv.org/abs/1503.02531

[8] Haowei Chen, Liekang Zeng, Shuai Yu, and Xu Chen. Knowledge Distillation for Mobile Edge Computation Offloading Microsoft Word - ZTE (arxiv.org)