실시간 주식 데이터 수집을 통한 추천시스템

양재 독수리
사남매

목차

팀 구성 및 업무분담

이규호

팀장

데이터 서버 구축
데이터 정제&가공
데이터 시각화
발표자료 준비

이소희

팀원

데이터 서버 구축
데이터 정제&가공
데이터 시각화
보고서 작성

권도양

팀원

추천 시스템을 위한
논리구조 수립
데이터 정제
데이터 수집
데이터 시각
시각자료 준비
보고서 작성

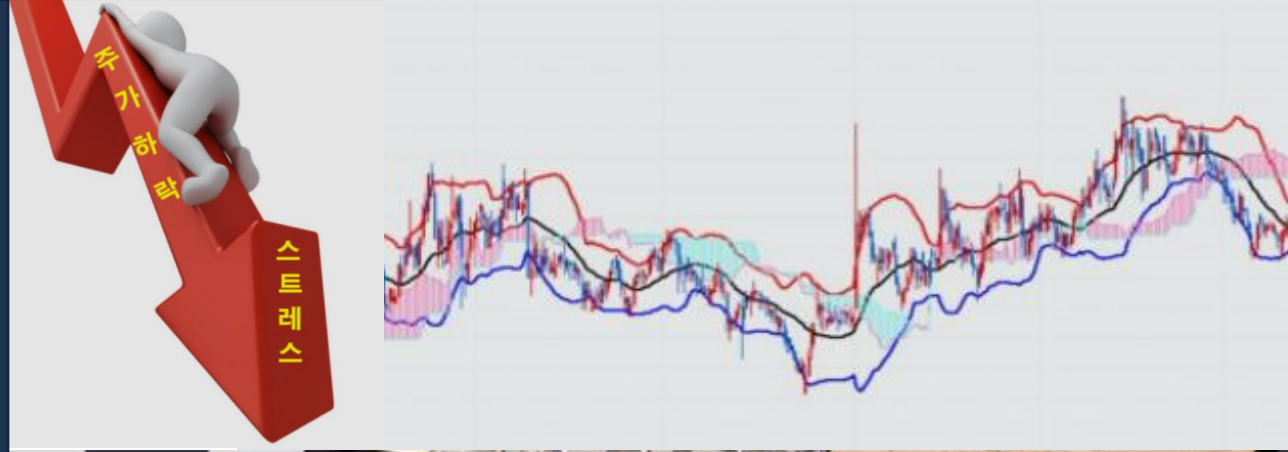
이상규

팀원

데이터 서버 구축
데이터 정제&가공
데이터 시각화
발표자료 준비

Part 1,

프로젝트 배경



Part 1, 프로젝트 배경

프로젝트 목표

비즈니스 아이디어

주식추천시스템

매수추천 시스템
매도추천 시스템
매수&매도 추천 시스템

매수추천시스템

요구사항

실시간 데이터 확보
딥러닝 모델 학습 가능여부
과거 데이터 축적 가능
(파이프라인)

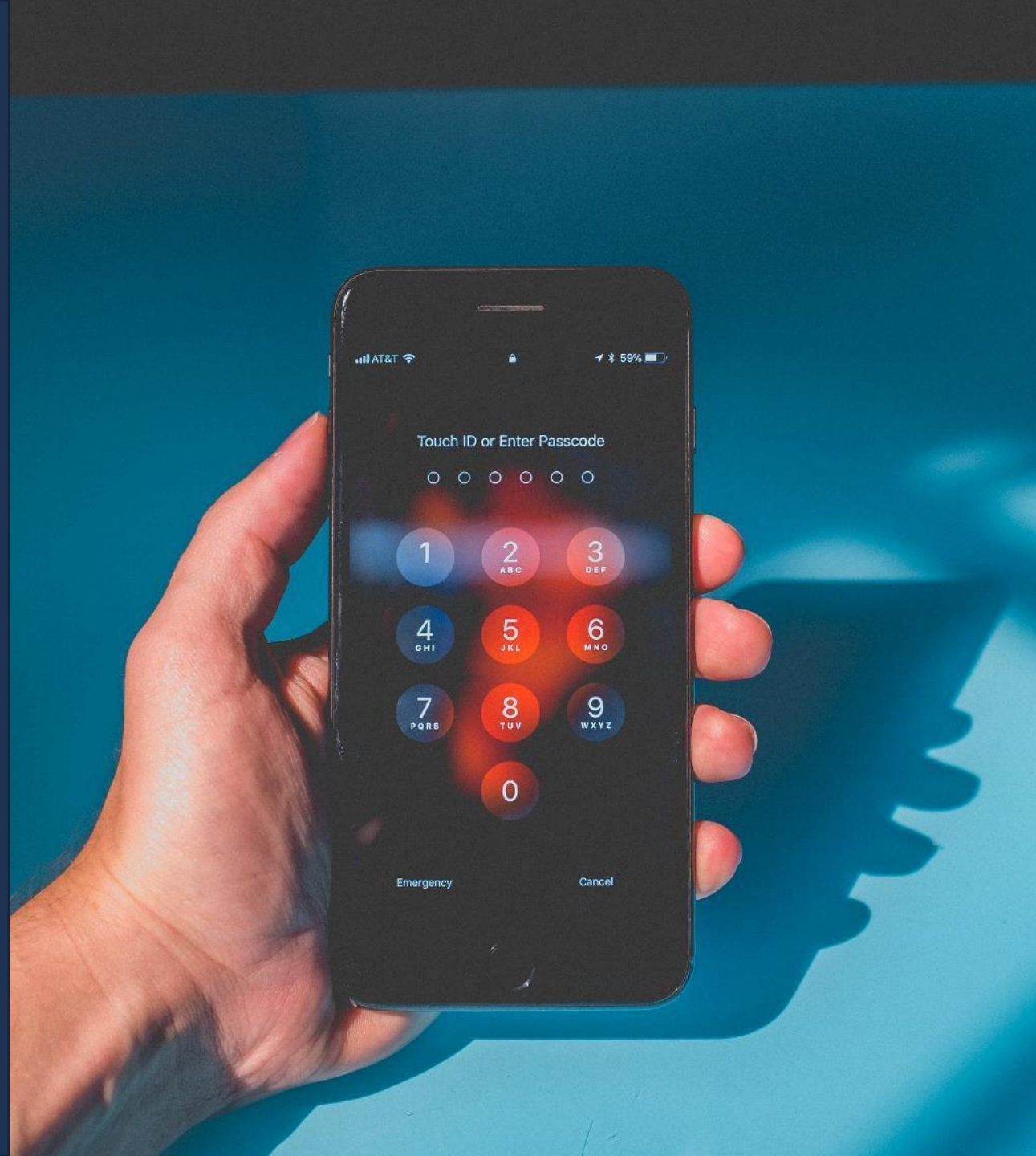
최종목표

여러 종목의 실시간 데이터의
학습을 통해

실시간으로 주식 매수를
추천하는 시스템

Part 2,

사용 프로그램



Part 2, 사용 프로그램

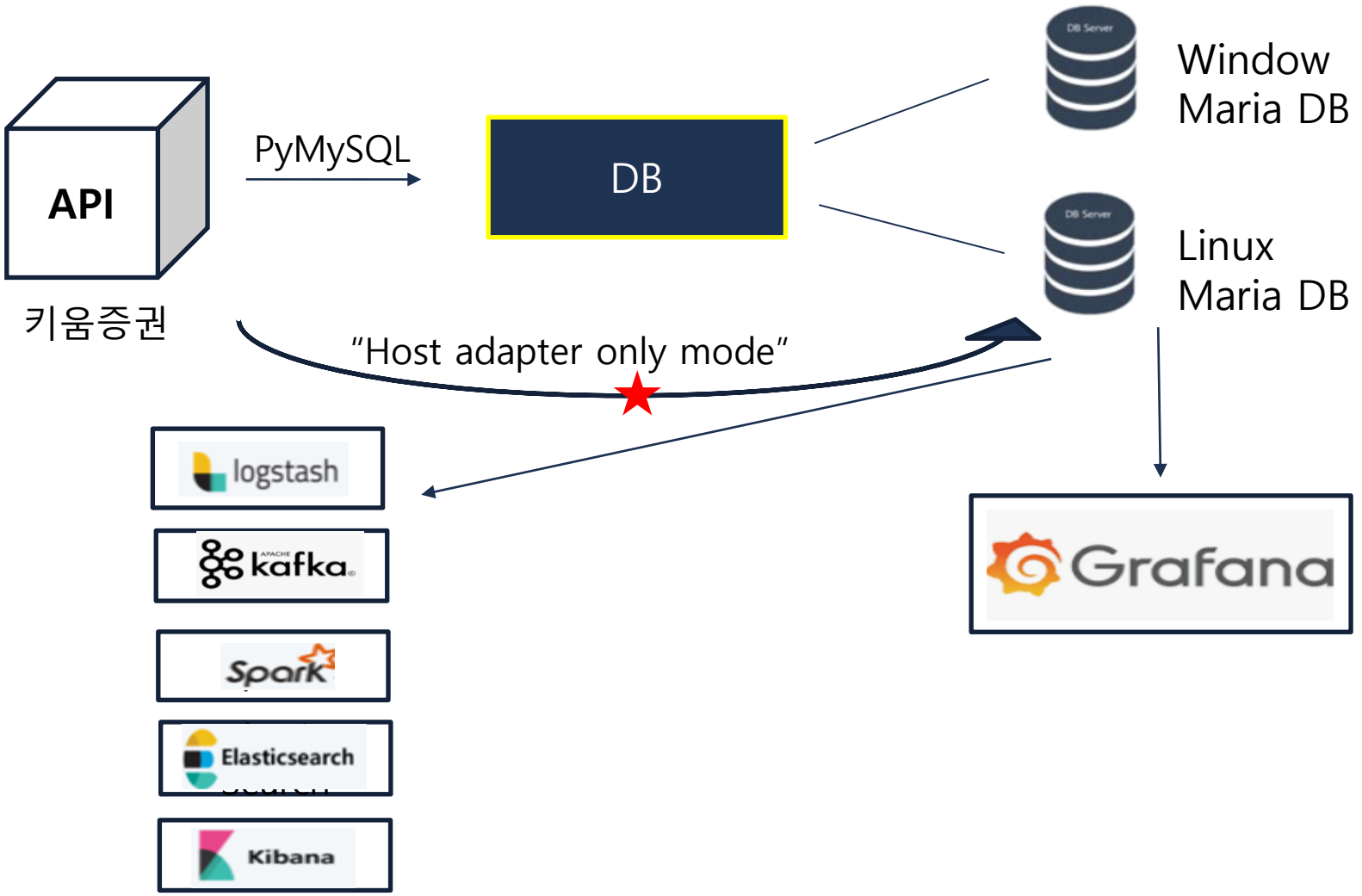
Windows

- Anaconda3 (python, 32-bit)
- Heidisql
- 키움API

Linux

- Oracle VM VirtualBox (Centos7)
- mariaDB
- Zookeeper
- Kafka
- Logstash
- Elasticsearch
- Grafana
- Kibana

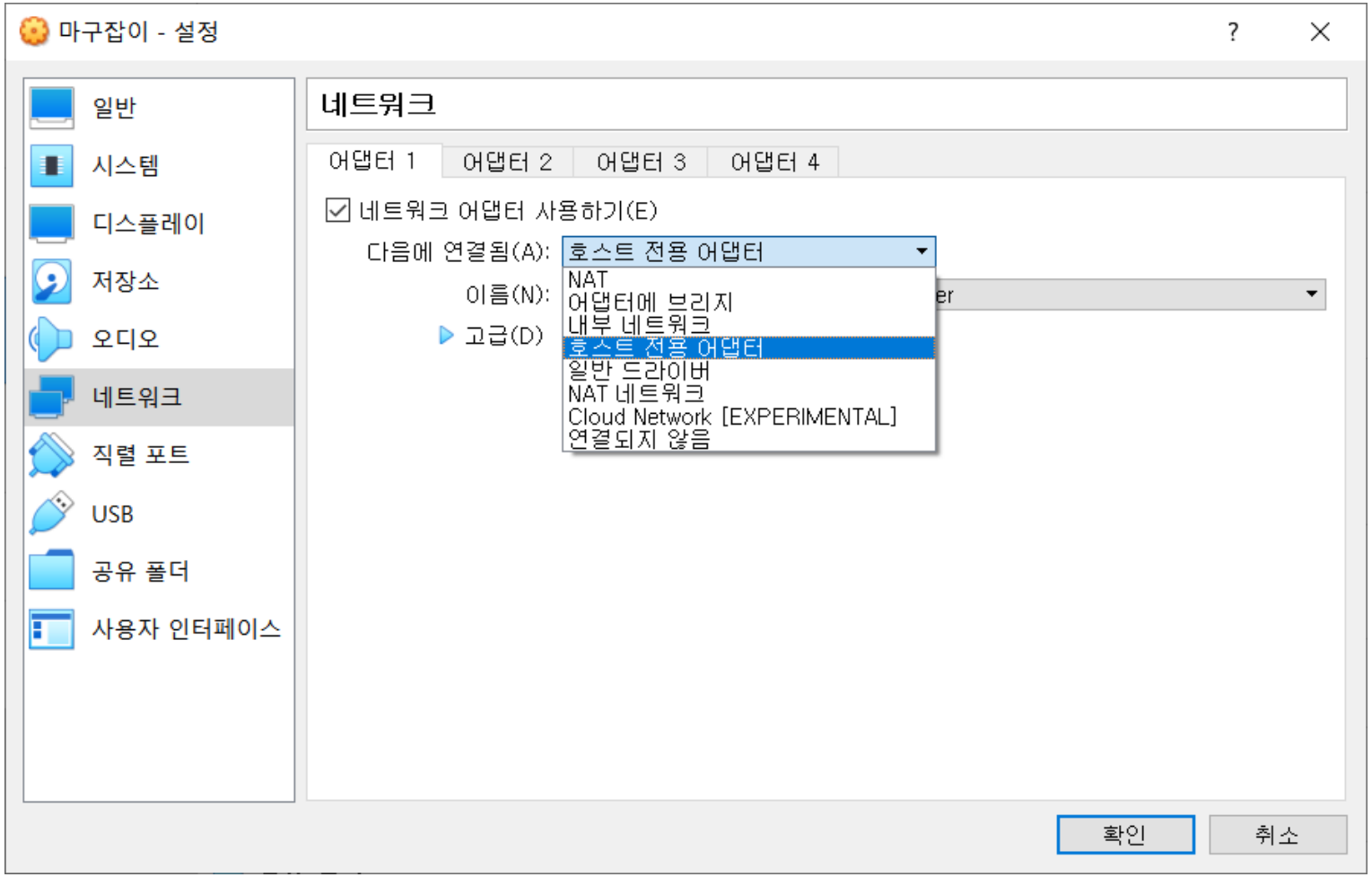
Part 2, 사용 프로그램



Part 3,

프로젝트 과정

Part 3, 프로그램 과정 - DB 연결



Part 3, 프로그램 과정 - DB 연결

Linux

방화벽 off

```
firewall-cmd --permanent --zone=public --add-port=3306/tcp
firewall-cmd --permanent --zone=public --add-service=mysql
firewall-cmd --reload
```

mariaDB에 user 추가

```
mysql -u root -p
use mysql;
create user root@'192.168.10.1'
```

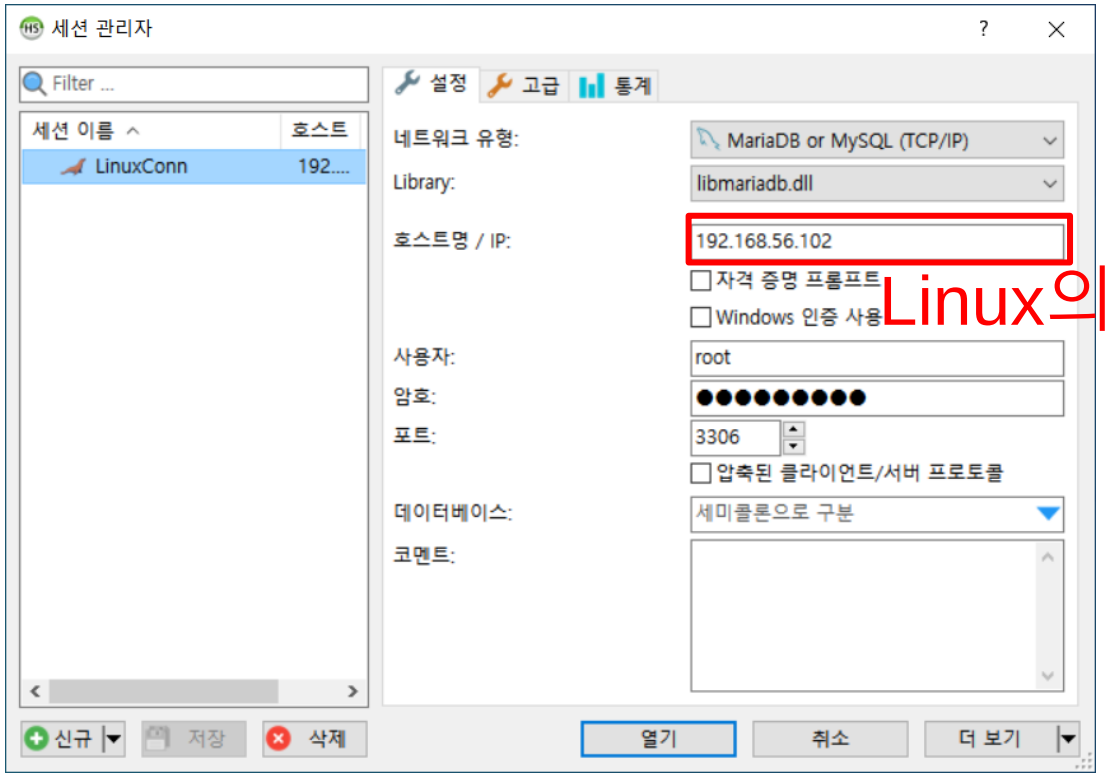
Windows의 IP

root계정 호스트 등록

```
grant all privileges
  -> on *.* to root@'192.168.10.1'
  -> identified by 'asdfsdf';
```

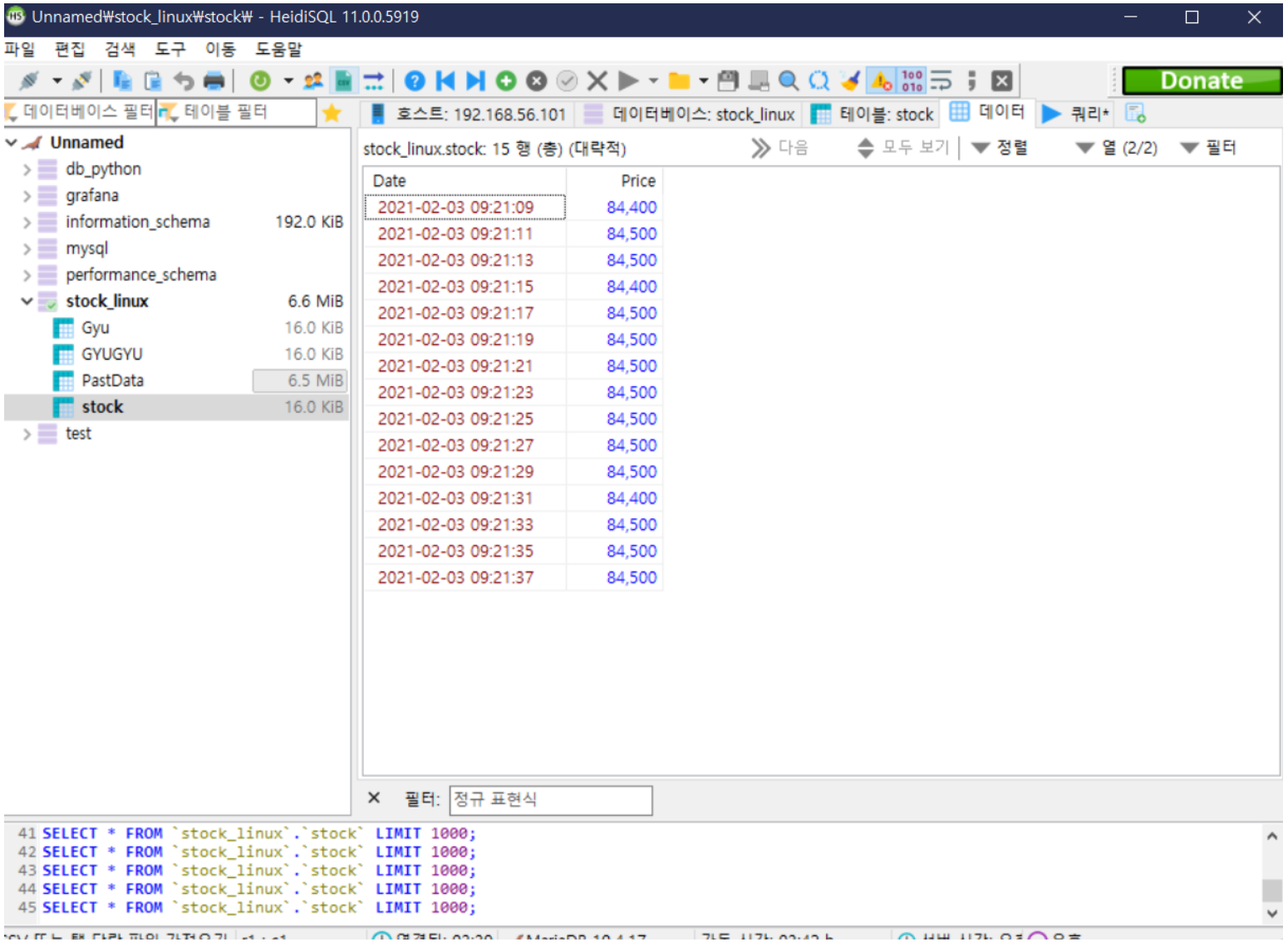
접속할 password

Windows



Linux의 IP

Part 3, 프로그램 과정 - DB 연결



Part 3, 프로그램 과정 - 시각화 과정

?

DB
→ Logstash
→ Elasticsearch, Kafka
→ Spark
→ Kibana

DB
→ Grafana

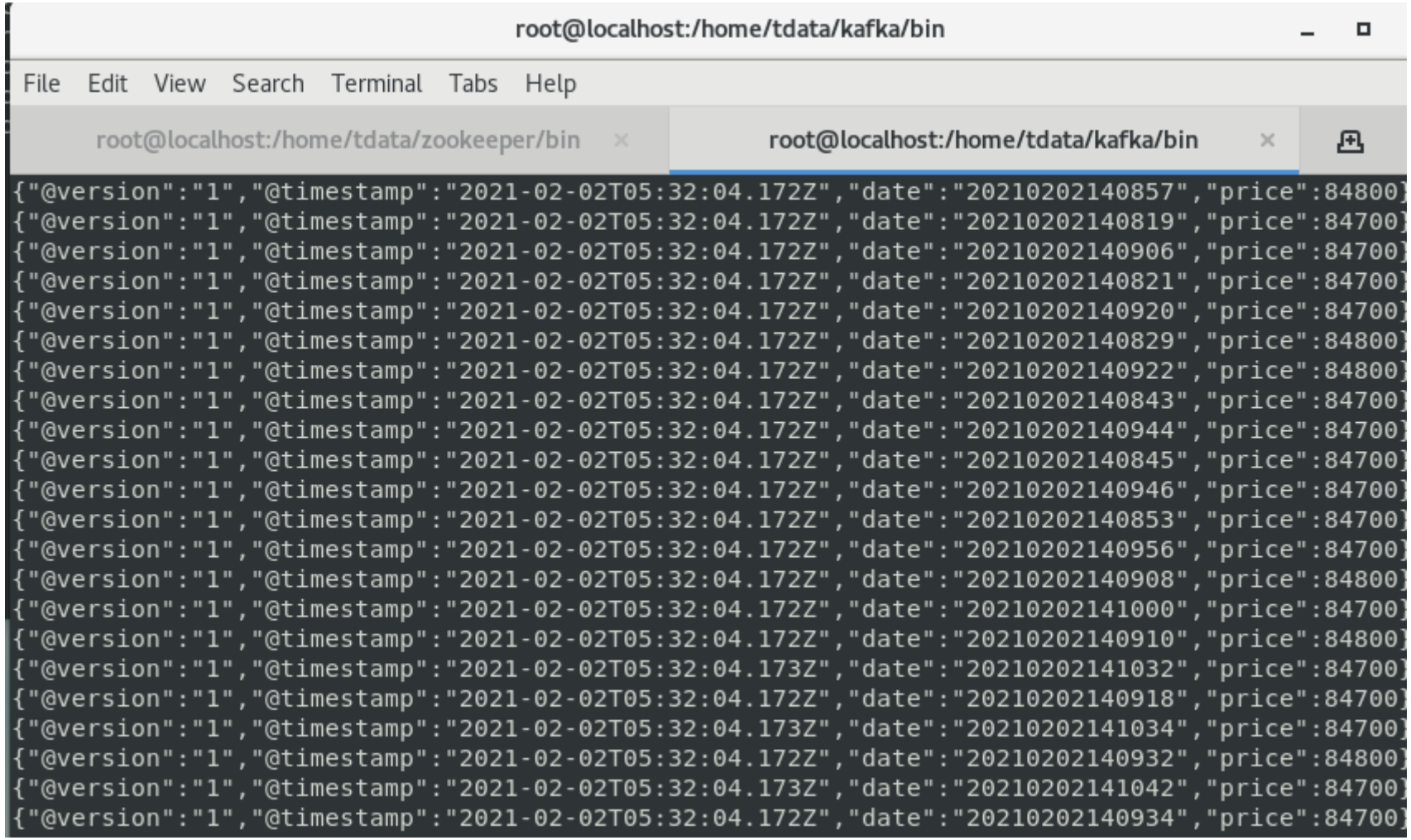
Part 3, 프로그램 과정 - 시각화 과정

```
##
input {
  jdbc {
     jdbc_driver_library => "/home/tdata/mysqlconn/mysql-connector-java-8.0.23.jar"
     jdbc_driver_class => "com.mysql.jdbc.Driver"
     jdbc_connection_string => "jdbc:mysql://localhost:3306/db_python"
     # jdbc_pool_timeout => 3000
     # jdbc_paging_enabled => true
     # jdbc_page_size => 100000
     jdbc_user => "root"
     jdbc_password =>"dltkdrb65"
     schedule => "* * * * * *"
     #tracking_column_typ => "numeric"
     #use_column_value => true
     #tracking_column => CREATE_DATE
     #charset => "UTF-8"
     #parameters => { "tracking_date" => "create_date" }
     statement => "SELECT * FROM stock"
       }
 }

filter {
        }

output {
     kafka{
        bootstrap_servers => "localhost:9092"
        topic_id => "stock_data2"
        codec => json
     }
     elasticsearch {
     hosts => "localhost:9200"
     #document_id => "%{[@metadata]}"
     index => "stock22"
        }
     #stdout { codec => rubydebug  }
  }
```

Part 3, 프로그램 과정 - 시각화 과정

```
Database changed
MariaDB [db_python]> select * from stock;
+----------------+-------+
| Date           | Price |
+----------------+-------+
| 20210202140414 | 84700 |
| 20210202140416 | 84700 |
| 20210202140418 | 84700 |
| 20210202140420 | 84700 |
| 20210202140422 | 84700 |
| 20210202140424 | 84700 |
| 20210202140426 | 84700 |
| 20210202140428 | 84700 |
| 20210202140430 | 84700 |
| 20210202140432 | 84800 |
| 20210202140434 | 84800 |
| 20210202140436 | 84700 |
| 20210202140438 | 84800 |
| 20210202140440 | 84700 |
| 20210202140442 | 84700 |
| 20210202140444 | 84700 |
| 20210202140446 | 84700 |
| 20210202140448 | 84700 |
| 20210202140450 | 84800 |
| 20210202140452 | 84800 |
| 20210202140454 | 84700 |
| 20210202140456 | 84700 |
| 20210202140458 | 84800 |
| 20210202140500 | 84700 |
| 20210202140502 | 84800 |
| 20210202140504 | 84700 |
| 20210202140506 | 84700 |
| 20210202140508 | 84700 |
| 20210202140510 | 84700 |
| 20210202140512 | 84700 |
| 20210202140514 | 84700 |
```

Part 3, 프로그램 과정 - 시각화 과정



문제 1

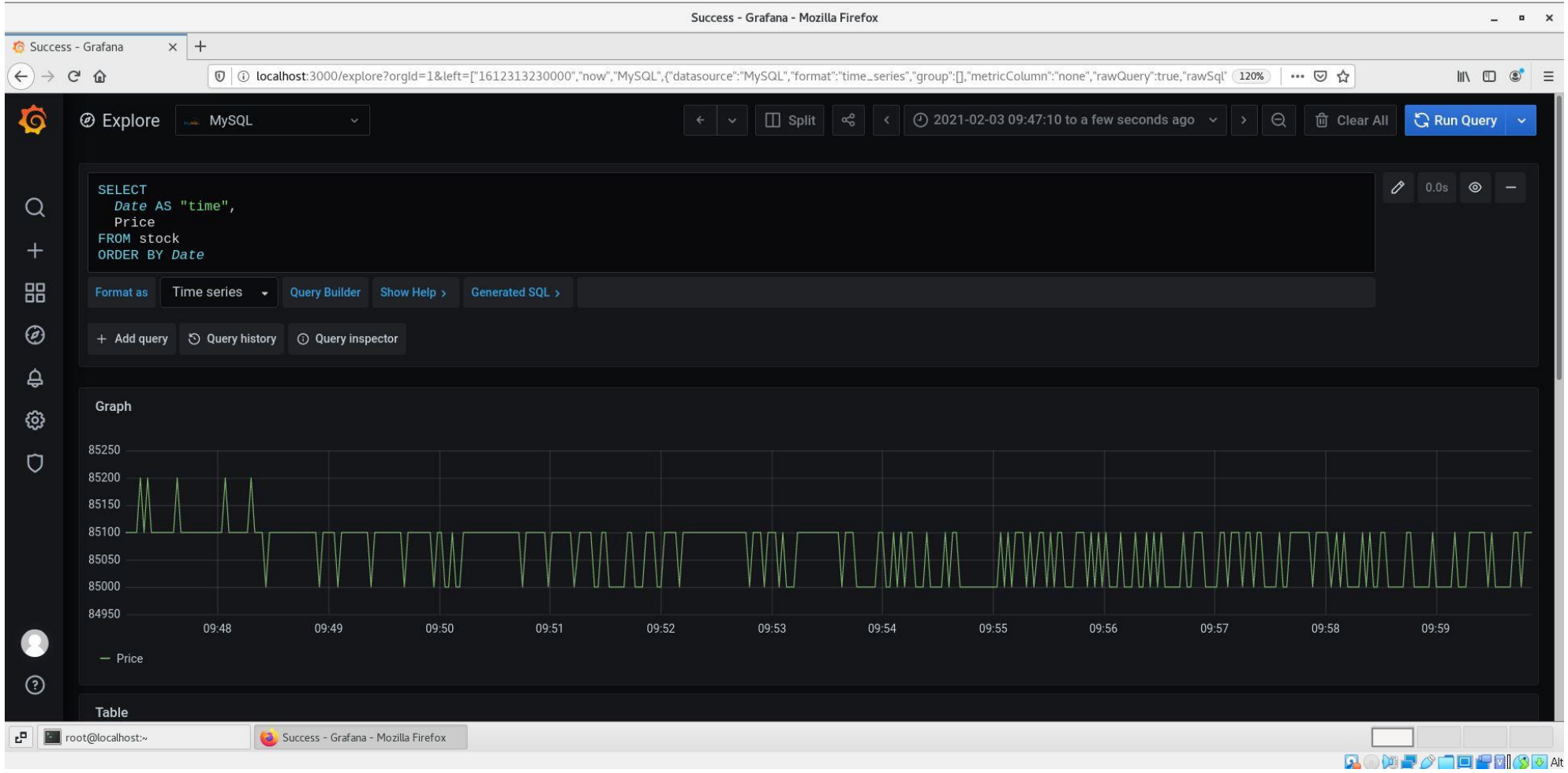
DB를 한번만 읽고, 추가되는 데이터를 읽어야하는데 마지막까지 읽으면 계속 DB를 읽는다.

문제 2

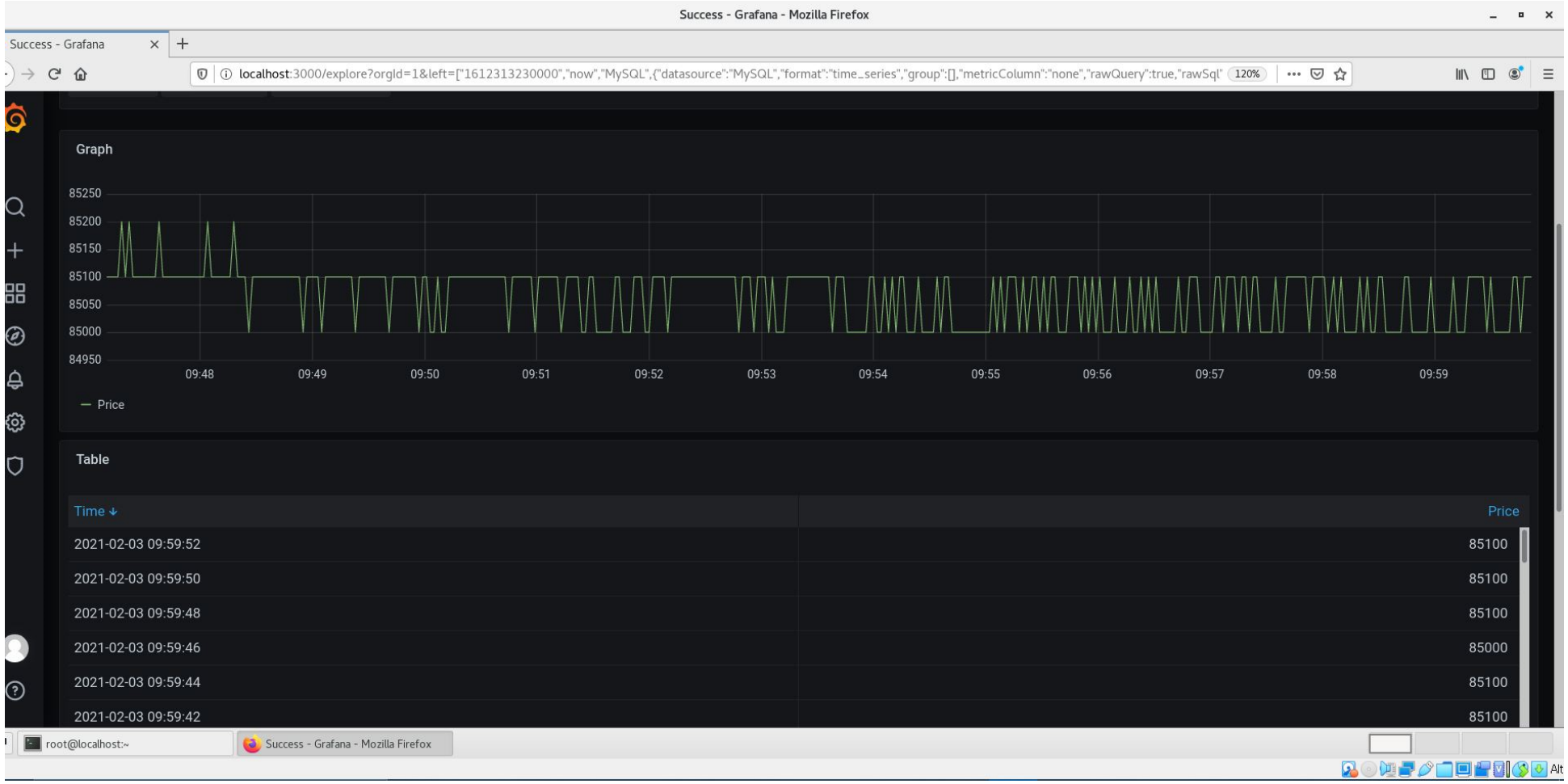
DB에 데이터가 추가되는 속도보다 DB를 읽는 속도가 빠르다.

→ DB에 쌓이는 데이터의 date가 뒤죽박죽이 될 수 있어 추후 정렬이 필요할 수도 있다.

Part 3, 프로그램 과정 - 시각화 과정 Grafana

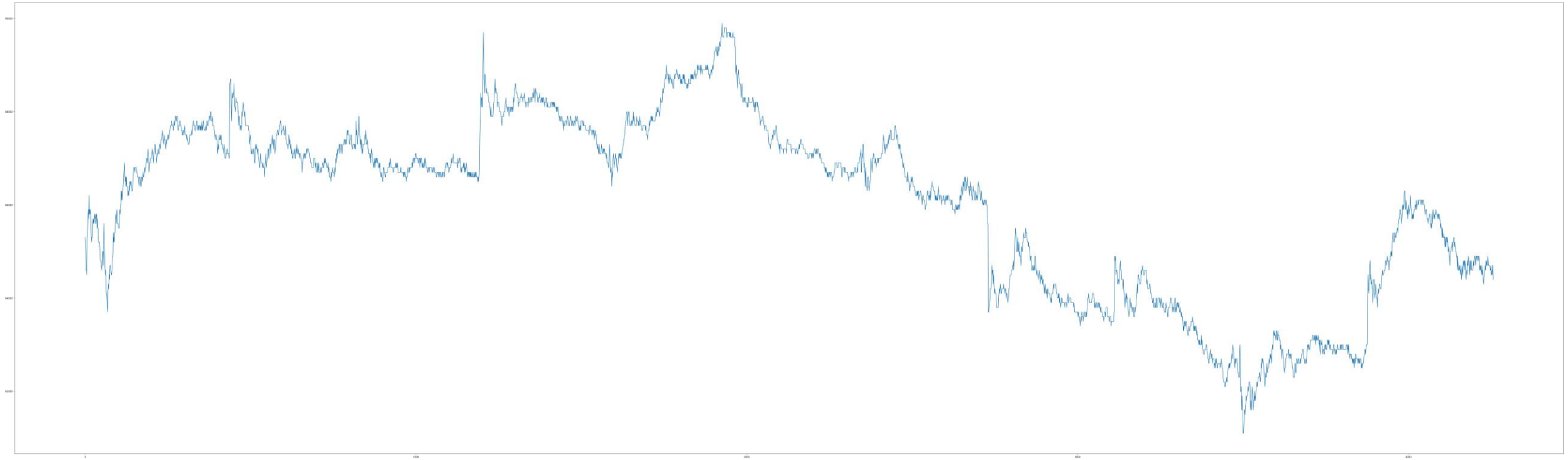


Part 3, 프로그램 과정 - 시각화 과정 Grafana



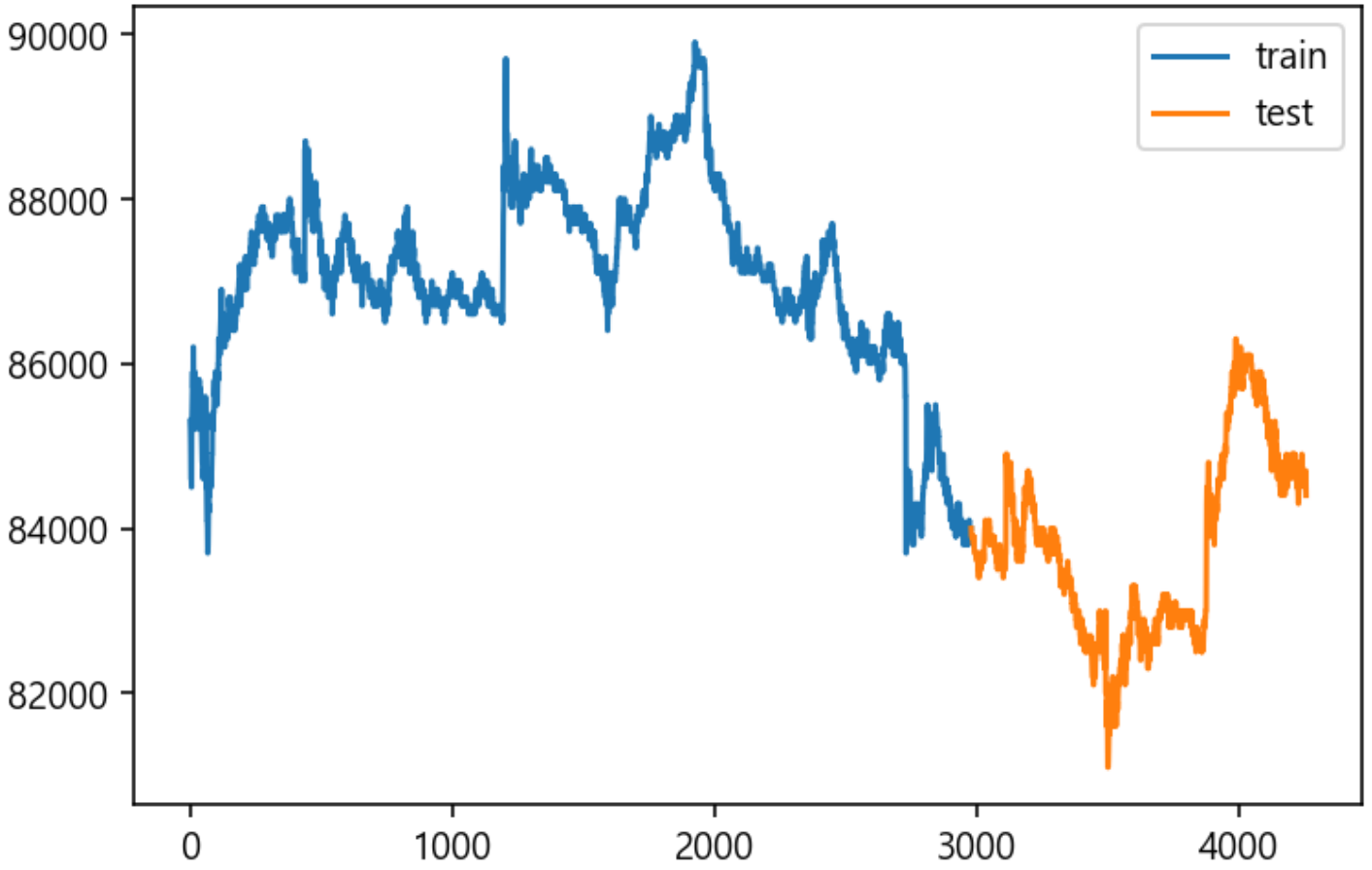
Part 3, 프로그램 과정 - 시각화 과정 Grafana

Part 3, 프로그램 과정 - 딥러닝 시행



삼성전자의 4200분 데이터

Part 3, 프로그램 과정 - 딥러닝 시행



70% 학습 30% test

Part 3, 프로그램 과정 - 딥러닝 시행 by LSTM

LSTM모델(input shape에서도 12개 동일)

```
K.clear_session()
model = Sequential()
model.add(LSTM(30, input_shape=(12, 1)))
# model.add(Dropout(0.3))
model.add(Dense(12))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30)                3840
_________________________________________________________________
dense (Dense)                (None, 12)                372
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 13
=================================================================
Total params: 4,225
Trainable params: 4,225
Non-trainable params: 0
_________________________________________________________________
```

Part 3, 프로그램 과정 - 딥러닝 시행 by LSTM

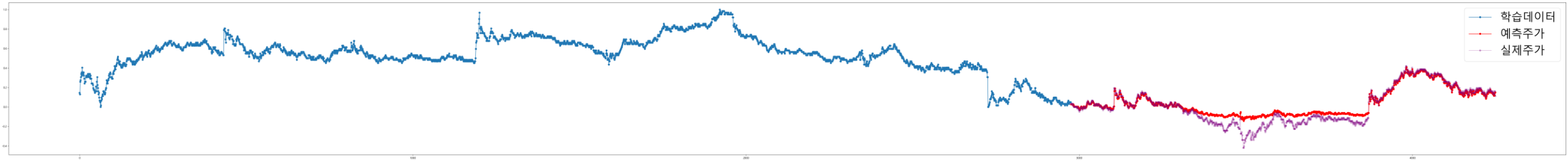
```
# loss의 감소가 의미없을경우 10번 참다가 끝내주도록 early_stop설정, 100번 반복 학습,
early_stop = EarlyStopping(monitor='loss', patience=10, verbose=1)

model.fit(X_train_t, y_train, epochs=100,
          batch_size=1, verbose=1, callbacks=[early_stop])
```

```
Epoch 1/100
2967/2967 [==============================] - 16s 5ms/step - loss: 7481204224.0000
Epoch 2/100
2967/2967 [==============================] - ETA: 0s - loss: 7019104768.00 - 17s 6ms/step - loss: 7017956352.0000
Epoch 3/100
2967/2967 [==============================] - 15s 5ms/step - loss: 6284109824.0000
Epoch 4/100
2967/2967 [==============================] - 15s 5ms/step - loss: 5362839040.0000
Epoch 5/100
2967/2967 [==============================] - 15s 5ms/step - loss: 4321907712.0000
Epoch 6/100
2967/2967 [==============================] - 15s 5ms/step - loss: 3236601088.0000
Epoch 7/100
2967/2967 [==============================] - 15s 5ms/step - loss: 2190249728.0000 0s - loss: 2
Epoch 8/100
2967/2967 [==============================] - 15s 5ms/step - loss: 1271167232.0000
```

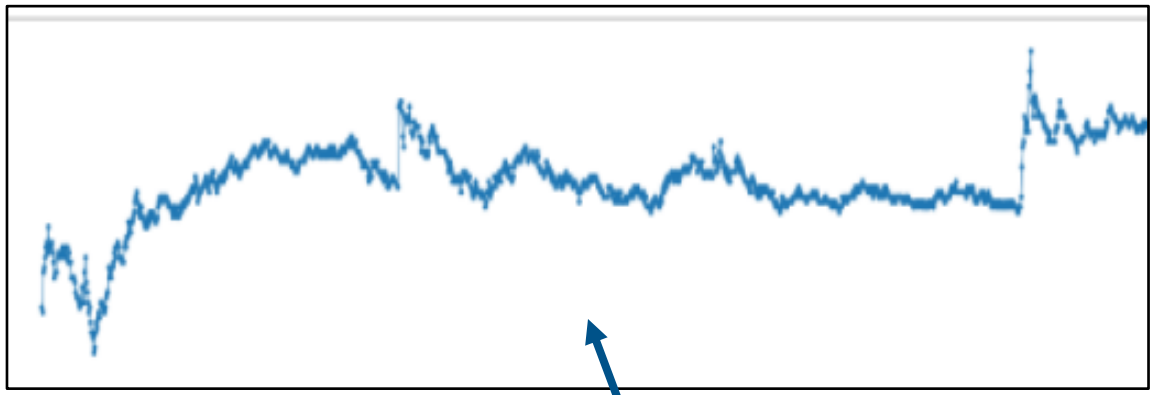
Part 3, 프로그램 과정 - 딥러닝 시행 by LSTM

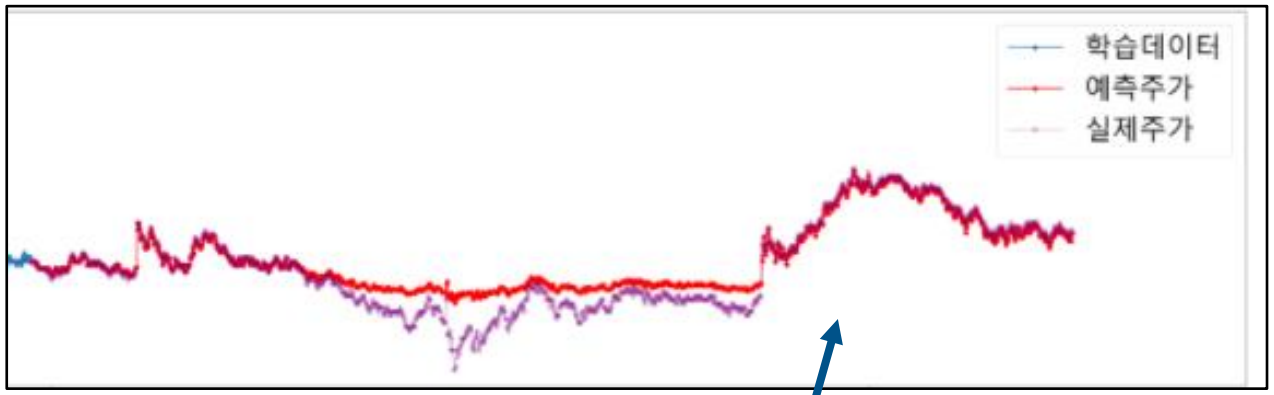
```
LSTM_model('data.pkl', minute = 5)
```

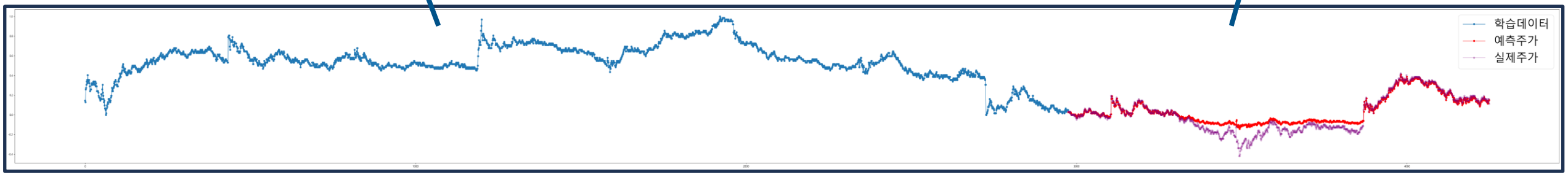
```
Epoch 1/100
2975/2975 [==============================] - 4s 1ms/step - loss: 0.0049
Epoch 2/100
2975/2975 [==============================] - 4s 1ms/step - loss: 7.3718e-04
Epoch 3/100
2975/2975 [==============================] - 4s 1ms/step - loss: 6.9157e-04
Epoch 4/100
2975/2975 [==============================] - 4s 1ms/step - loss: 6.9671e-04
Epoch 5/100
2975/2975 [==============================] - 4s 1ms/step - loss: 6.4174e-04
Epoch 6/100
2975/2975 [==============================] - 4s 1ms/step - loss: 6.0241e-04
Epoch 7/100
```



Part 3, 프로그램 과정 - 딥러닝 시행 by LSTM



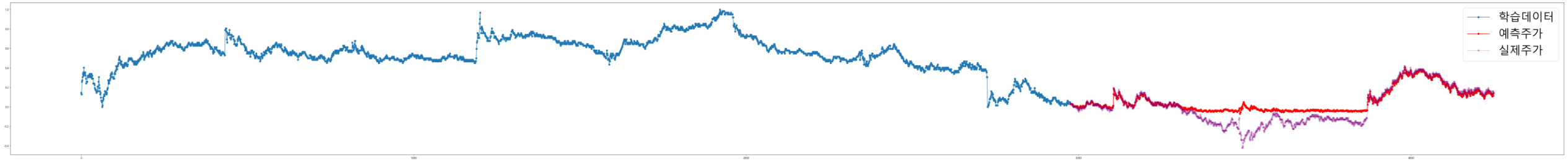




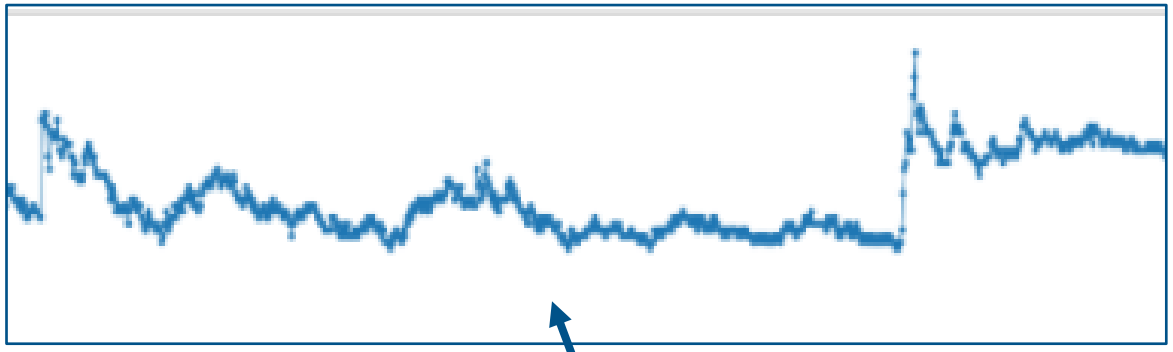
Part 3, 프로그램 과정 - 딥러닝 시행 by GRU

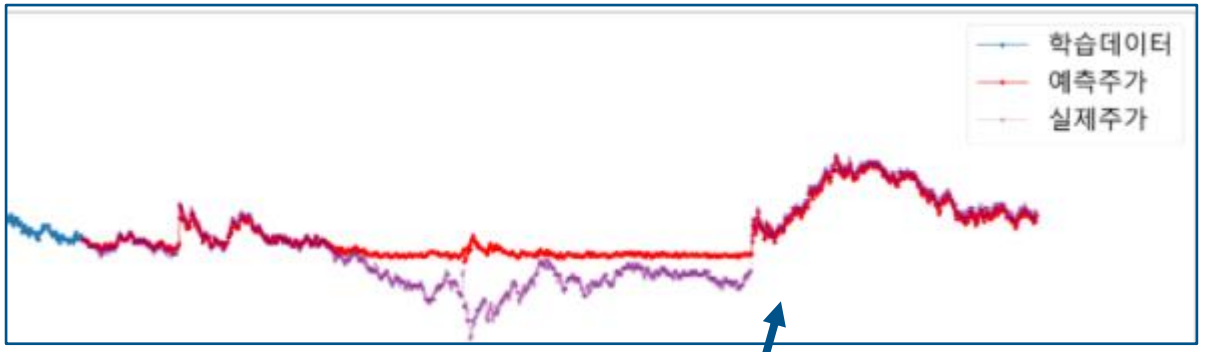
```
GRU_model('data.pkl', minute = 5)
```

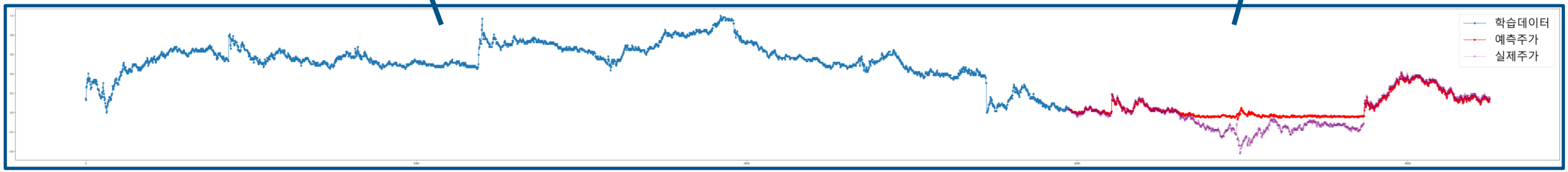
```
Epoch 1/100
2975/2975 [==============================] - 4s 1ms/step - loss: 0.0027A: 0s - loss: 0.002
Epoch 2/100
2975/2975 [==============================] - 4s 1ms/step - loss: 8.6609e-04
Epoch 3/100
2975/2975 [==============================] - 4s 1ms/step - loss: 8.2032e-04
Epoch 4/100
2975/2975 [==============================] - 5s 2ms/step - loss: 7.3459e-04A: 0s - loss:
Epoch 5/100
2975/2975 [==============================] - 4s 1ms/step - loss: 7.0187e-04
Epoch 6/100
2975/2975 [==============================] - 5s 2ms/step - loss: 6.6474e-04
```



Part 3, 프로그램 과정 - 딥러닝 시행 by GRU

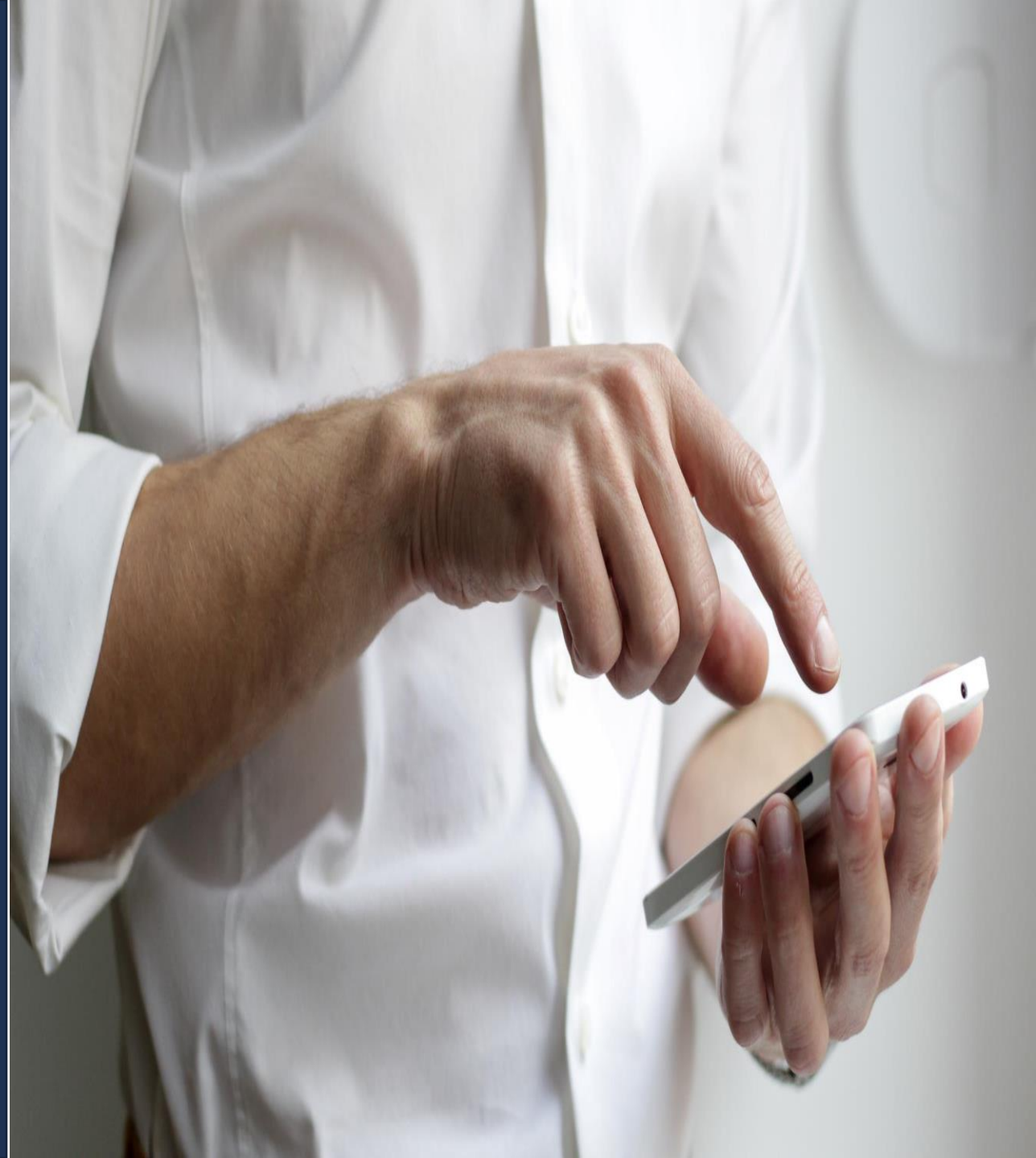






Part 4,

결론 및 고찰



Part 4, 결론 및 고찰 - 결론

LSTM, GRU를 통해 지난 과거의 데이터로 미래를 loss 0점대로 예측을 하는 것을 확인

특이현상(급등)은 예측했으나, 특이현상(급락)은 제대로 예측하지 못함.

4200분의 데이터만으로 학습했기 때문에 더 많은 누적된 데이터를 학습시키면 향상된 결과를 보일 것으로 기대

→ 과적합 방지 위해 시행착오를 통해 최적의 학습 데이터양 조절 필요

Part 4, 결론 및 고찰 - 결론

현재 상황

한 종목의 실시간 데이터 확보 성공

Spark 연동 실패로 머신러닝 X

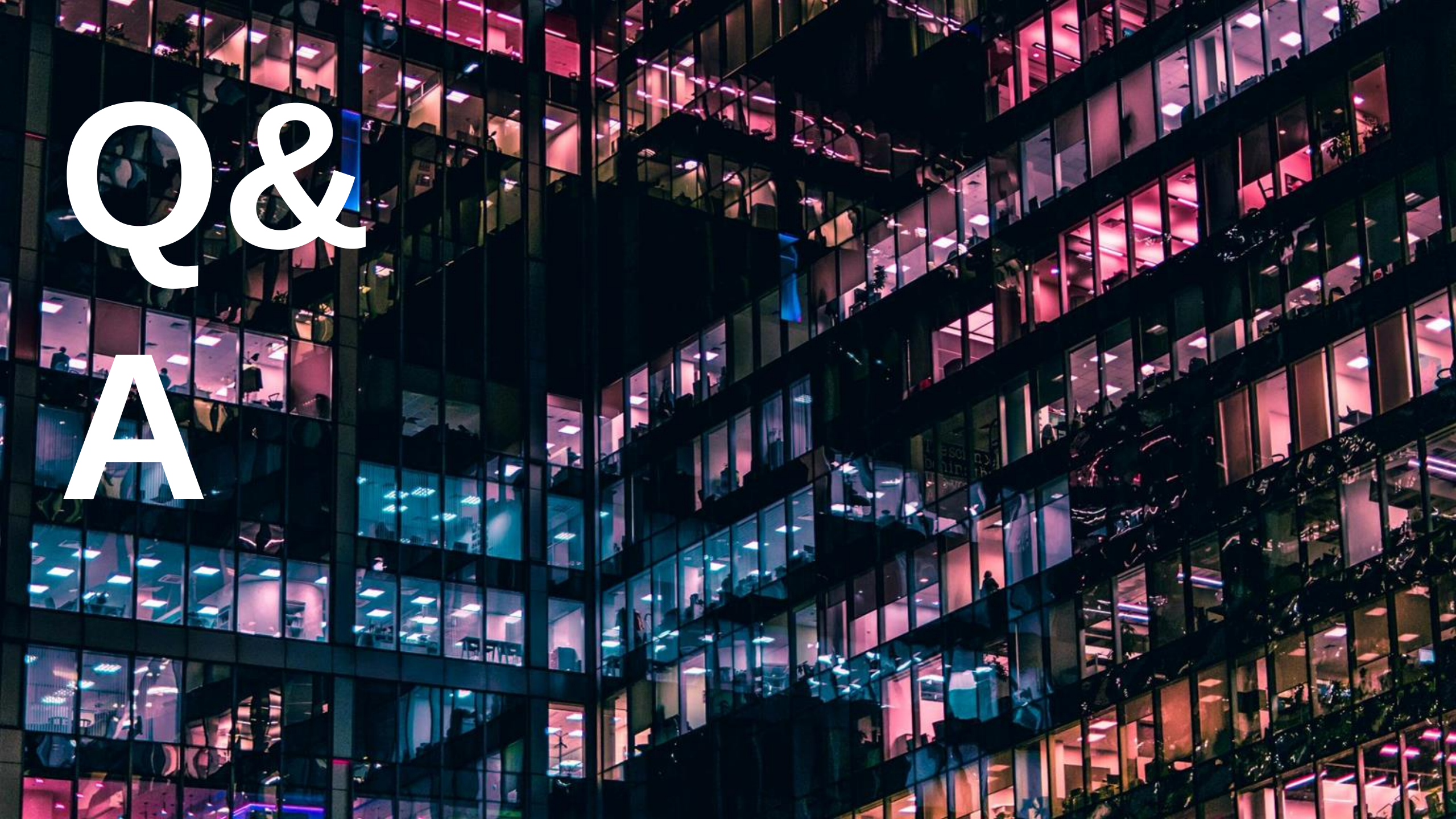
Anaconda tensorflow를 통한 딥러닝 학습

향후 목표

여러 종목의 실시간 데이터 확보

Kafka → Spark 연동을 통한 머신러닝 학습

Grafana 시각화 전까지 데이터 전처리 및 모든 과정 동기화


Q&
A