# <최종 보고서>

가상화폐 대시보드 서비스


**32174919 허경환**

# 목 차

# Ⅰ. 개요

## 1.1 개발동기 및 기존의 문제점

*“여러 형태의 자산을 한눈에 파악할 수 있어야 한다.”*

기존의 서비스는 한가지 형태의 자산이라도 각각 다른 플랫폼이라면 다르게 시각화하여 보여주었습니다. 예를 들어, 가상화폐 자산의 경우, 특정 거래소에서 거래하는 사용자의 자산만을 보여주고, 다른 거래소에 있는 사용자의 자산은 또 다른 플랫폼에서 관리해야 했습니다. 만약 여러 플랫폼에 분산되어 있는 보유자산들의 정보를 한눈에 파악할 수 있도록 서비스를 제공한다면, 사용자들은 추후 자산관리를 어떻게 할 것인지 계획을 세울 수 있습니다. 또한, 현재 가치를 기준으로 전체 자산에서 특정 자산이 얼마의 비중을 차지하는지 확인이 가능하고, 자산형태의 변동을 유동적으로 진행할 수 있을 것입니다.

## 1.2 목표

1) 사용자에게 여러 종류의 가상화폐에 대한 현재가 정보를 제공해야합니다.
2) 사용자에게 여러 플랫폼에 있는 가상화폐 자산을 통합하여 제공함으로써 자산상태를 쉽게 파악할 수 있도록 해야합니다.
3) 사용자에게 제시된 대시보드를 기반으로 자산관리의 인사이트를 얻을 수 있어야 합니다.

# Ⅱ. 작품제작 핵심기술/개념

## 2.1 스파크

아파치 스파크는 통합 컴퓨팅 엔진이며 클러스터 환경에서 데이터를 병렬로 처리하는 라이브러리 집합입니다. 스파크는 가장 활발하게 개발되고 있는 병렬처리 오픈소스 엔진이며 빅데이터에 관심 있는 여러 개발자와 데이터 과학자에게 표준 도구가 되어가고 있습니다. 스파크는 파이썬, 자바, 스칼라, R 이렇게 네 가지 언어를 지원합니다. 이번 프로젝트에서는 파이썬을 이용해 스파크를 다루었습니다. 스파크는 수천 대의 서버로 구성된 클러스터뿐만 아니라 단일 노트북 환경에서도 실행할 수 있습니다. 이런 특성들로인해 본 프로젝트에서 아파치 스파크를 채택하여 프로젝트를 수행하였습니다.

## 2.2 MySQL

MySQL은 전세계적으로 가장 널리 사용되고 있는 오픈소스 데이터베이스입니다. 매우 빠르고, 유연하며, 사용하기 쉽다는 특징이 있습니다. 또한 MySQL은 유닉스나 리눅스, Windows 등 다양한 운영체제에서 사용할 수 있기도 합니다. 무엇보다도 프로젝트를 위해 사용할 아파치 스파크, 태블로와 상호연동이 잘되면서도 오픈소스로 개발되는 무료 프로그램이어서 본 프로젝트에서 데이터베이스 구축을 위해 채택하였습니다.

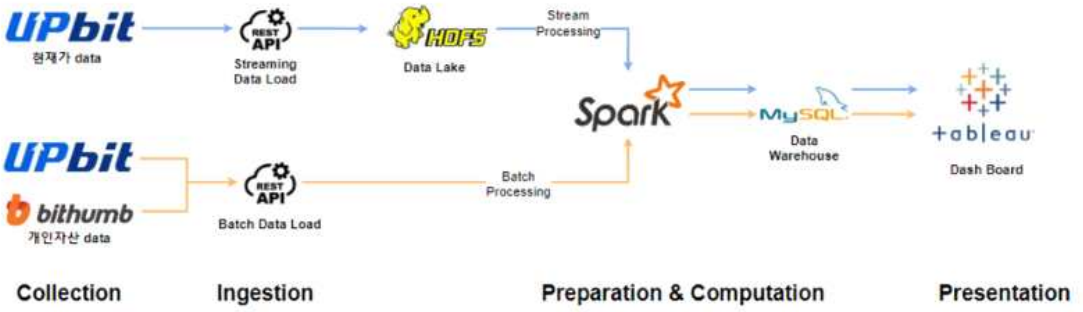
## 2.3 Tableau

태블로는 기본적으로 데이터 시각화 도구입니다. 주로 BI (Business Intelligence)에서 비즈니스 시각화 도구로 사용됩니다. 세일즈, 마케팅, 커머스, 저널리즘, 제조업 등 수도 없이 많은 비즈니스 분야에 태블로가 활용되고 있으며,  최근에는 공공의 영역에서도 태블로를 도입하고 있는 추세이기도 합니다. 다양하게 활용되는 데이터 시각화 툴로 그만큼 다양한 데이터베이스나 프로그램과 상호연동이 수월하고, 시각화와 관련된 거의 모든 기능을 갖추고 있어서 본 프로젝트에서 시각화를 위한 툴로 채택하였습니다.

## 2.4 데이터 파이프라인

컴퓨터 과학에서 파이프라인은 한 데이터 처리 단계의 출력이 다음 단계의 입력으로 이어지는 형태로 연결된 구조를 가리키는 개념입니다. 이렇게 연결된 데이터 처리 단계는 한 여러 단계가 서로 동시에, 또는 병렬적으로 수행될 수 있어 효율성의 향상을 꾀할 수 있다. 본 프로젝트에서는 위에서 언급한 핵심기술들을 파이프라인의 컴포넌트로 사용하여 효율적인 데이터 파이프라인을 구현하고자 하였습니다.

# Ⅲ. 시스템 구성도




- 파이프라인 구조
  - Data Source : 로우 데이터를 가져올 특정 플랫폼 서버
  - Storage : 정형/반정형/비정형 데이터를 비롯한 모든 가공되지 않은 데이터를 한 곳에 모아두는 저장소.
  - Data Transform : 로우 데이터를 원하는 형태로 가공하는 과정. 빅데이터 분산 처리 프로그램 사용
  - Database : 의사결정에 도움을 주기 위해 분석가능한 포맷으로 변환한 데이터들의 저장소
  - Data Visualization : 데이터를 시각화하여 대시보드를 제작

- 상세 구축 환경 및 프레임워크
  - Data Source : Upbit API
  - Storage : Local Storage
  - Data Transform : Pyspark (spark version 3.1.2 + python version 3.9.13)
  - Database : MySQL
  - Data Visualization : Tableau

# Ⅳ. 개발과정

## 4-1. 업비트api/빗썸api를 이용해서 raw data 가져오기

### 4-1-1. 업비트 서버로부터 현재가, 개인자산 데이터 가져오기

**1. Data Source로부터 현재가 데이터 가져오기**

```
#btc, eth, xrp의 현재가 정보 가져와서 json 파일로 저장
#저장하는 파일명은 현재 시간

import requests
import sys
from datetime import datetime
import time

#업비트
url = "https://api.upbit.com/v1/ticker?markets=KRW-BTC%2C%20KRW-ETH%2C%20KRW-XRP"
headers = {"Accept": "application/json"}

#현재시간 가져오기
now = datetime.now()
#가져온 현재시각을 문자열처리
current_time = now.strftime("%Y-%m-%d_%H-%M-%S")
#업비트 서버에 request하고, 응답받은 것을 response에 넣기
response = requests.request("GET", url, headers=headers)

#파일열기
f = open("C:/Users/gocjs/Desktop/raw/upbit/"+str(current_time)+".json", 'w')
#업비트 서버로부터 받은 response를 텍스트 형태로 출력
print(response.text, file=f)
#파일닫기
f.close()
```

**1. Data Source로부터 개인자산 데이터 가져오기**

```
#개인자산 데이터를 가져와서 json 파일로 저장
#저장하는 파일명은 현재 시간

import jwt
import hashlib
import os
import requests
import uuid
from urllib.parse import urlencode, unquote
from datetime import datetime
import time

#업비트
access_key = "[redacted]"
secret_key = "[redacted]"
server_url = "https://api.upbit.com"

payload = {
    'access_key': access_key,
    'nonce': str(uuid.uuid4()),
}

jwt_token = jwt.encode(payload, secret_key)
authorization = 'Bearer {}'.format(jwt_token)
headers = {
  'Authorization': authorization,
}

res = requests.get(server_url + '/v1/accounts', headers=headers)


#현재시간 가져오기
now = datetime.now()
#가져온 현재시각을 문자열처리
current_time = now.strftime("%Y-%m-%d")


#파일열기
f = open("C:/Users/gocjs/Desktop/raw/upbit-budget/"+str(current_time)+".json", 'w')
#업비트 서버로부터 받은 response를 텍스트 형태로 출력
print(res.json(), file=f)
#파일닫기
f.close()
```

## 4-1-2. 빗썸 서버로부터 개인자산 데이터 가져오기

```
import pybithumb
from datetime import datetime
import time
import json
```

```
#현재시간 가져오기
now = datetime.now()
#가져온 현재시각을 문자열처리
current_time = now.strftime("%Y-%m-%d")
```

```
connect_key = "9                              a44"
secret_key = "eb                              96"
```

```
bithumb = pybithumb.Bithumb(connect_key, secret_key)
```

```
res=[]
```

```
#비트코인의 총 잔고
#거래중인 비트코인의 수량
#보유중인 총 원화
#주문에 사용된 원화
#위의 4종류 값을 튜플로 묶어 반환
btc=bithumb.get_balance('BTC')
```

```
res.append({'currency': 'BTC', 'balance': str(btc[0]), 'locked': str(btc[1]), 'avg_buy_price': '0', 'datetime
```

```
#이더리움의 총 잔고
#거래중인 이더리움의 수량
#보유중인 총 원화
#주문에 사용된 원화
#위의 4종류 값을 튜플로 묶어 반환
eth=bithumb.get_balance('ETH')
```

```
res.append({'currency': 'ETH', 'balance': str(eth[0]), 'locked': str(eth[1]), 'avg_buy_price': '0', 'datetime
```

```
#리플의 총 잔고
#거래중인 리플의 수량
#보유중인 총 원화
#주문에 사용된 원화
#위의 4종류 값을 튜플로 묶어 반환
xrp=bithumb.get_balance('XRP')
```

```
res.append({'currency': 'XRP', 'balance': str(xrp[0]), 'locked': str(xrp[1]), 'avg_buy_price': '0', 'datetime
```

```
#파일열기
f = open("C:/wealth-management-dashboard/raw/upbit-budget/"+str(current_time)+".json", 'w')
#업비트 서버로부터 받은 response를 텍스트 형태로 출력
with open("C:/wealth-management-dashboard/raw/upbit-budget/"+str(current_time)+".json", 'w') as f:
    json.dump(res, f, ensure_ascii=False, indent=4)
```

# 4-2. Apache Spark를 이용해 데이터 처리

## 4-2-1. 업비트

### 2. 코인데이터를 스파크로 처리하기

```
#json파일을 데이터프레임으로 변환하기
df = spark.read.json("C:/Users/gocjs/Desktop/raw/upbit/"+str(current_time)+".json")
```

```
df.show()
```

```
#데이터프레임에서 특정 column만 추출하기
df = df.select("trade_date", "opening_price", "high_price", "low_price", "trade_price", "trade_volume",
```

```
df.show()
```

```
+----------+-------------+----------+---------+-----------+------------+-------+
|trade_date|opening_price|high_price|low_price|trade_price|trade_volume| market|
+----------+-------------+----------+---------+-----------+------------+-------+
|  20221218|     22230000|  22253000| 22140000|   22247000|   4.4949E-4|KRW-BTC|
|  20221218|      1574500|   1584000|  1567500|    1577000|  0.00722384|KRW-ETH|
|  20221218|          471|       472|      466|        469|424.98900055|KRW-XRP|
+----------+-------------+----------+---------+-----------+------------+-------+
```

### 2. 개인자산 데이터를 스파크로 처리하기

```
#키가 'avg_buy_price_modified'이면 'datetime'으로, 값는 string(파일이름)으로 수정
def replace_in_file(file_path, old_str, new_str):
    # 파일 읽어들이기
    fr = open(file_path, 'r')
    lines = fr.readlines()
    fr.close()

    # old_str -> new_str 치환
    fw = open(file_path, 'w')
    for line in lines:
        fw.write(line.replace(old_str, new_str))
    fw.close()

# 호출: file1.txt 파일에서 comma(,) 없애기
replace_in_file("C:/Users/gocjs/Desktop/raw/upbit-budget/"+str(current_time)+".json", 'avg_buy_price_mod
replace_in_file("C:/Users/gocjs/Desktop/raw/upbit-budget/"+str(current_time)+".json", 'False', '"'+str(c
replace_in_file("C:/Users/gocjs/Desktop/raw/upbit-budget/"+str(current_time)+".json", 'True', '"'+str(cu
```

```
from pyspark.sql.types import *
```

```
#json파일의 스키마를 설정하기
devColumns = [
    StructField("currency",StringType()),
    StructField("balance",StringType()),
    StructField("locked",StringType()),
    StructField("avg_buy_price",StringType()),
    StructField("unit_currency",StringType()),
    StructField("datetime",StringType()),
]
```

```
devSchema = StructType(devColumns)
```

```
df = spark.read.option("multiline","true").json("C:/Users/gocjs/Desktop/raw/upbit-budget/"+str(current_t
```

```
df.printSchema()
```

```
root
 |-- avg_buy_price: string (nullable = true)
 |-- balance: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- datetime: string (nullable = true)
 |-- locked: string (nullable = true)
 |-- unit_currency: string (nullable = true)
```

```
df.show()
```

```
+-------------+-------------+--------+----------+------+-------------+
|avg_buy_price|      balance|currency|  datetime|locked|unit_currency|
+-------------+-------------+--------+----------+------+-------------+
|     27441000|    0.0001822|     BTC|2022-12-20|     0|          KRW|
|            0|3492.83000601|     KRW|2022-12-20|     0|          KRW|
|      1562000|   0.00352112|     ETH|2022-12-20|     0|          KRW|
|          451|   13.3037694|     XRP|2022-12-20|     0|          KRW|
+-------------+-------------+--------+----------+------+-------------+
```

## 4-2-2. 빗썸

```
from pyspark.sql.types import *
```

```
#json파일의 스키마를 설정하기
devColumns = [
    StructField("currency",StringType()),
    StructField("balance",StringType()),
    StructField("locked",StringType()),
    StructField("avg_buy_price",StringType()),
    StructField("avg_buy_price_modified",StringType()),
    StructField("unit_currency",StringType()),
    StructField("datetime",StringType()),
]
```

```
devSchema = StructType(devColumns)
```

```
df = spark.read.option("multiline","true").json("C:/wealth-management-dashboard/raw/upbit-budget/"+str(curren
```

```
df.printSchema()
```

```
root
 |-- avg_buy_price: string (nullable = true)
 |-- balance: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- datetime: string (nullable = true)
 |-- locked: string (nullable = true)
 |-- unit_currency: string (nullable = true)
```

```
df.show()
```

```
+-------------+-------+--------+----------+------+-------------+
|avg_buy_price|balance|currency|  datetime|locked|unit_currency|
+-------------+-------+--------+----------+------+-------------+
|            0|    0.0|     BTC|2023-06-06|   0.0|          KRW|
|            0|    0.0|     ETH|2023-06-06|   0.0|          KRW|
|            0|    0.0|     XRP|2023-06-06|   0.0|          KRW|
+-------------+-------+--------+----------+------+-------------+
```

```
#데이터프레임에서 특정 column만 추출하기
df = df.select("avg_buy_price", "balance", "currency", "datetime", "locked", "unit_currency")
```

```
df.show()
```

```
+-------------+-------+--------+----------+------+-------------+
|avg_buy_price|balance|currency|  datetime|locked|unit_currency|
+-------------+-------+--------+----------+------+-------------+
|            0|    0.0|     BTC|2023-06-06|   0.0|          KRW|
|            0|    0.0|     ETH|2023-06-06|   0.0|          KRW|
|            0|    0.0|     XRP|2023-06-06|   0.0|          KRW|
+-------------+-------+--------+----------+------+-------------+
```

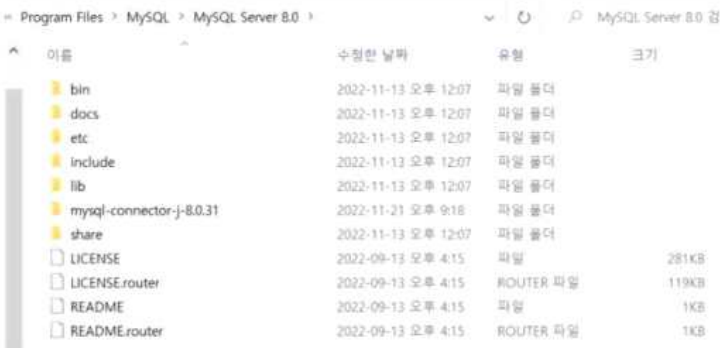
## 4-3. Apache Spark와 MySQL 연결(Mysql-java-connector 이용)

① 다음의 링크에서 버전에 맞게 커넥터 다운받기

https://dev.mysql.com/downloads/connector/j/

② MySQL의 Base Directory를 찾아서 해당 위치에 커넥터를 넣어주기

```
mysql> show variables like 'basedir';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| basedir       | C:₩Program Files₩MySQL₩MySQL Server 8.0₩ |
+---------------+-----------------------------------------+
1 row in set, 1 warning (0.01 sec)
```

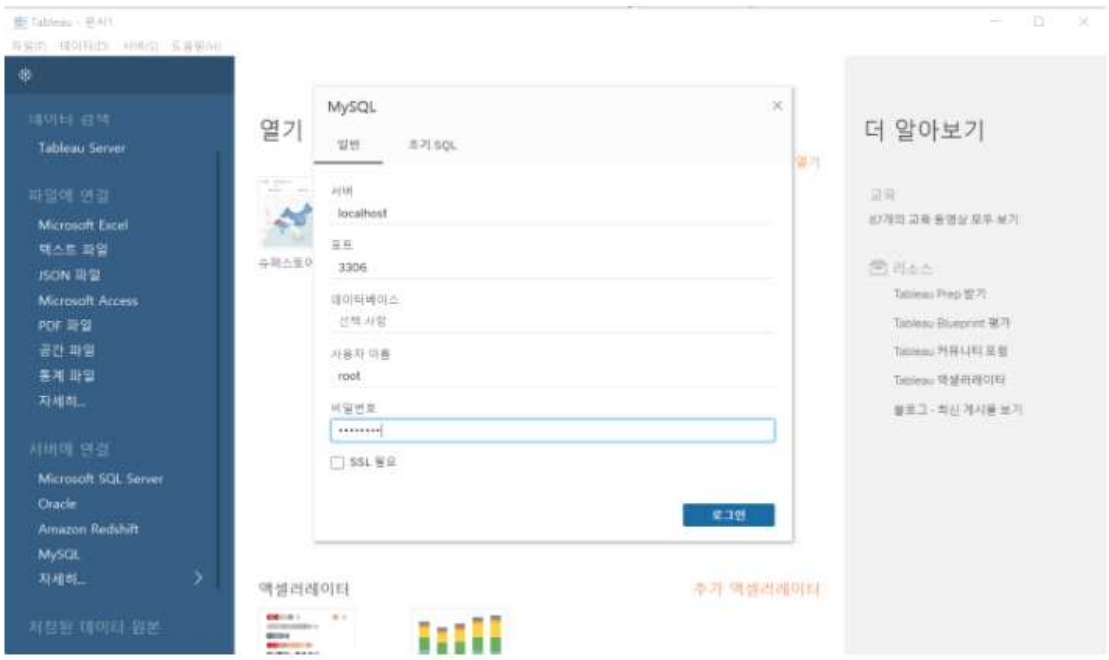



```python
import mysql.connector
from pyspark.sql import SparkSession

# spark 세션 연결
spark = SparkSession.builder.config("spark.jars", "mysql-connector-j-8.0.31.zip") \
    .master("local").appName("PySpark_MySQL_test").getOrCreate()

#데이터베이스의 정보를 스파크의 데이터프레임으로 읽어오기
df = (spark
    .read
    .format("jdbc")
    .option("url", "jdbc:mysql://localhost:3306/{데이터베이스이름}")
    .option("driver", "com.mysql.jdbc.Driver")
    .option("dbtable", "{테이블이름}")
    .option("user", "root").option("password", "******")
    .load())


#데이터 확인하기
df.show()
```

## 4-4. MySQL에 처리한 데이터 저장

```
#스파크의 데이터프레임을 데이터베이스에 넣기
#1. json파일을 데이터프레임으로 변환하기
df=spark.read.json("C:/wealth-management-dashboard/raw/upbit/2023-06-06_12-35-26.json")


#2. 데이터프레임을 mysql의 테이블로 저장 - 단, 해당 이름의 테이블을 새로 생성시키는 것으로 같은 이름의 테이블이 없어야한다.
df.write.format('jdbc').mode("append").option("url","jdbc:mysql://localhost:3306/opensource_db").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "ticker2").option("user", "root").option("password", "********").save()
```

## 4-5. MySQL과 Tableau 연결(ODBC Driver 이용)




서버 : localhost

포트 : 3306

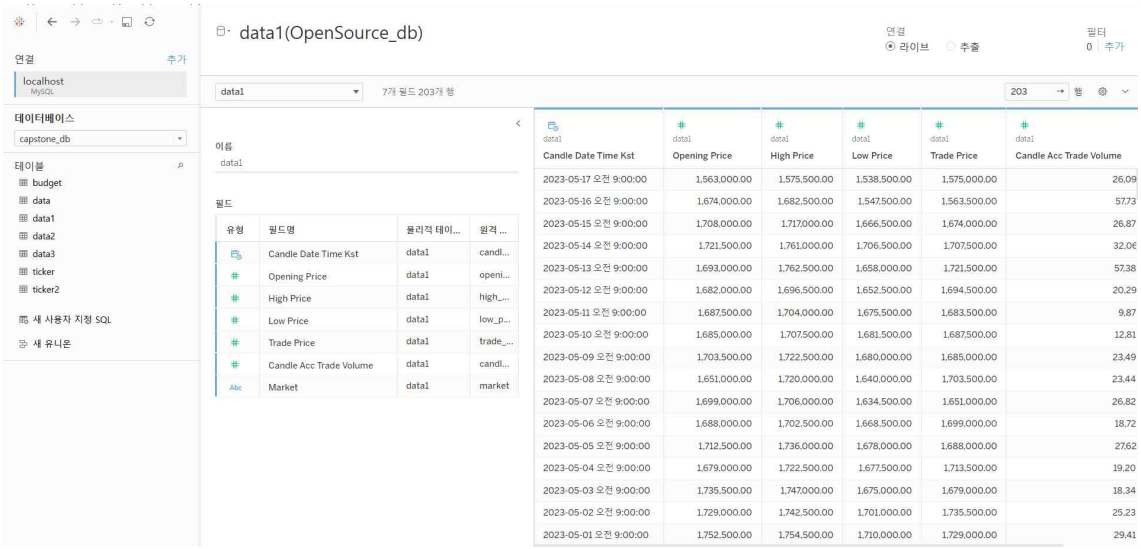
사용자 이름 : root

비밀번호

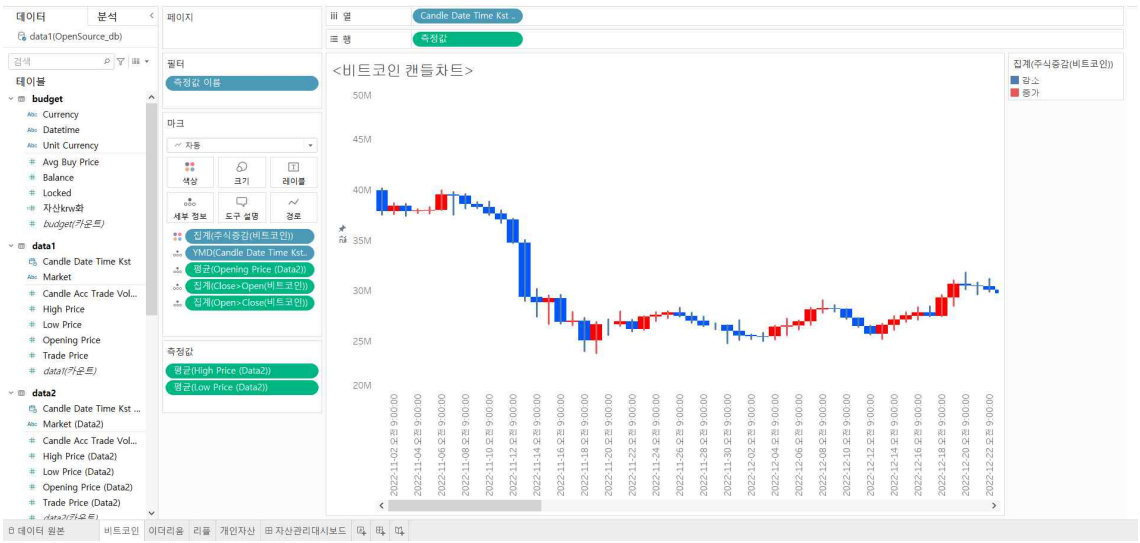
를 적어주고 로그인

## 4-6. Tableau서버로 DB의 데이터 가져와 시각화

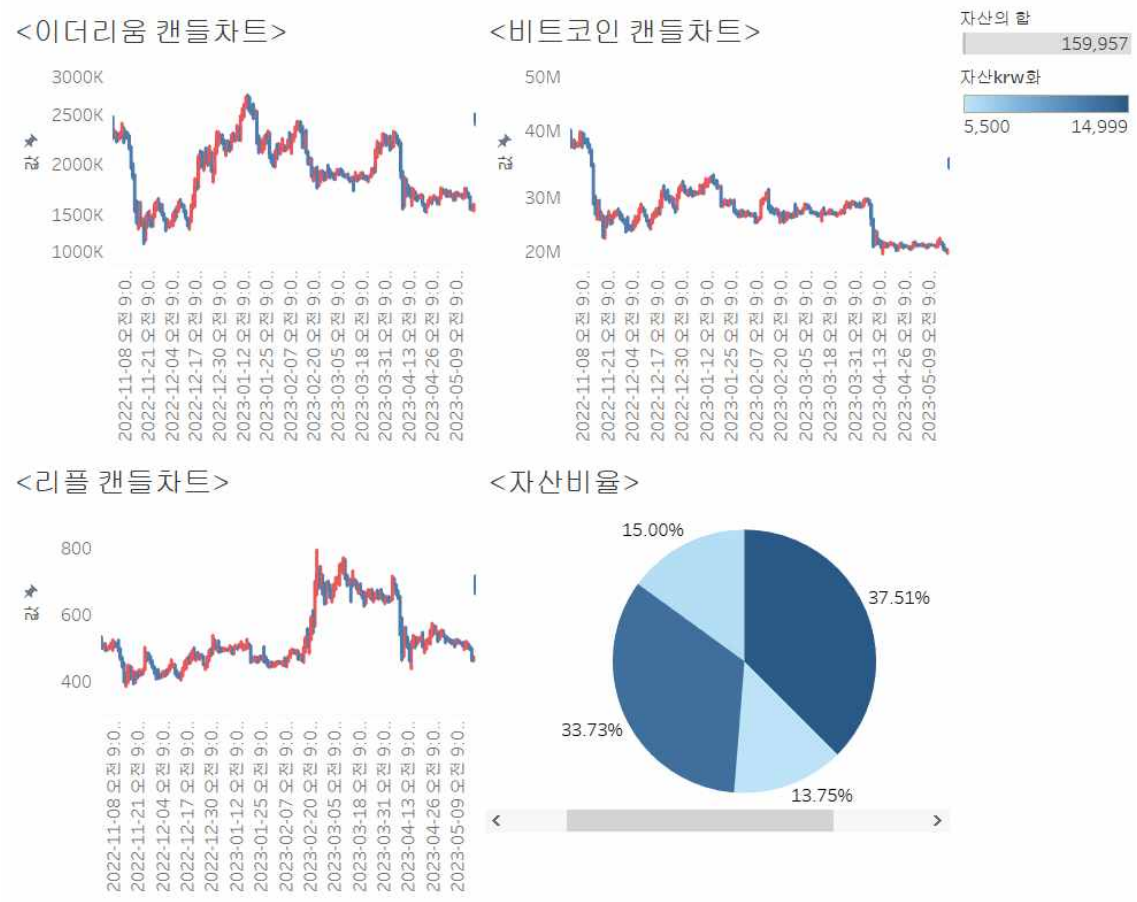
### 1) 태블로 서버와 연결된 MySQL의 데이터




### 2) MySQL 데이터를 이용한 시각화 작업



# V. 주요 실행화면


<이더리움 캔들차트>
<비트코인 캔들차트>
자산의 합
159,957
자산krw화
5,500
14,999
3000K
2500K
2000K
1500K
1000K
50M
40M
30M
20M
값
2022-11-08 오전 9:0..
2022-11-21 오전 9:0..
2022-12-04 오전 9:0..
2022-12-17 오전 9:0..
2022-12-30 오전 9:0..
2023-01-12 오전 9:0..
2023-01-25 오전 9:0..
2023-02-07 오전 9:0..
2023-02-20 오전 9:0..
2023-03-05 오전 9:0..
2023-03-18 오전 9:0..
2023-03-31 오전 9:0..
2023-04-13 오전 9:0..
2023-04-26 오전 9:0..
2023-05-09 오전 9:0..
<리플 캔들차트>
<자산비율>
800
600
400
15.00%
37.51%
33.73%
13.75%

# 부록1 [데이터 저장 경로]

**[로우데이터 경로]**

1. 현재가 데이터

   C:\wealth-management-dashboard\raw\upbit

2. 사용자 자산 데이터-업비트

   C:\wealth-management-dashboard\raw\upbit-budget

3. 사용자 자산 데이터-빗썸

   C:\wealth-management-dashboard\raw\bithumb-budget

**[데이터 프로세싱 후 저장할 경로]**

1. 현재가 데이터

   C:\wealth-management-dashboard\prepared\upbit

2. 사용자 자산 데이터-업비트

   C:\wealth-management-dashboard\prepared\upbit-budget

3. 사용자 자산 데이터-빗썸

   C:\wealth-management-dashboard\prepared\bithumb-budget

# 부록2 [참고문헌]


1. 빌 체임버스, 마테이 자하리아(2018). 스파크 완벽 가이드. O'REILLY
2. 황재진, 윤영진(2022). 사례분석으로 배우는 데이터 시각화. 한빛미디어
3. 제러드 마스, 프랑수아 가릴로(2021). 스파크를 활용한 실시간 처리. O'REILLY
4. 홀든 카로 외 3인(2015). 러닝 스파크. O'REILLY
5. 페타 제체비치, 마르코 보나치(2018). 스파크를 다루는 기술. 길벗
6. 아파치 스파크 doc(3.3.1), https://spark.apache.org/docs/latest/api/python
7. MySQL connector, https://www.mysql.com/products/connector/
8. Upbit API Docs, https://docs.upbit.com/
9. Bithumb API Docs, https://apidocs.bithumb.com/