FTD, 이동평균선, 거래량을 이용한 비트코인에 대한 승률과 수익률 백테스팅

12234195 컴퓨터공학과 조진우 (선택과제 A형)

1. 서론

일상생활 속에서 빅데이터를 활용한 문제 해결이라는 주제를 보고 처음에는 어떤 주제를 다룰지 고민이 많았습니다. 중간 대체 과제로 금리 예측을 하려고 했지만, 선배들의 공개된 리포트 주제를 살펴보니 동일한 주제가 이미 있었습니다. 그래서 부득이하게 주제를 급히 변경해야 했고, 실제로 내가 겪고 있는 문제를 해결할 방법을 고민하며 주변을 둘러보기 시작했습니다.

요즘 다음 차기 대통령이 트럼프가 될 가능성이 거론되고 있고, 트럼프는 코인을 제도권에 편입시키려는 의지를 가진 인물로 알려져 있습니다. 한편, 중국과 러시아 등은 달러에 대항하기 위해 금을 매입하고 있는데, 이는 금으로 쏠리는 유동성과 영향력을 약화시키기 위해 비트 코인을 제도권에 편입하려는 시도와 연결될 수 있다고 생각합니다. 앞으로 비트 코인 가격이 어떻게 변할지는 알 수 없지만, 제도권 편입 가능성은 상당히 높다고 판단됩니다. 이에 이번 데이터 분석 수업을 활용해 코인 시장을 분석하고, 제가 겪고 있는 문제를 해결하고자 합니다.

저는 나름 5 년 가까이 주식 투자 경험을 쌓아왔고, 누적 방문자 3 만 명 이상의 블로그를 운영하고 있습니다. 하지만 코인 투자에는 소극적이었습니다. 그 이유는 주식 투자 시 사용했던 방법들이 코인 투자에서도 효과적일지 확신할 수 없었기 때문입니다. 주식 투자 방식을 그대로 코인에 적용했다가 문제가 생기지 않을까 하는 걱정이 있었습니다. 그래서 평소 사용하던 거래량 분석, 이동평균선(MA), FTD(Follow-Through Day) 방식이 과연 코인 투자에도 유용한지 검증해 보기로 했습니다.

이를 위해 백 테스팅, 즉 과거 데이터를 활용해 제 투자 방식을 검증할 예정입니다. 각 방식의 수익률을 표로 정리하고, 승률을 도출해내며, 어떤 방식이 가장 효과적인지 비교 분석할 계획입니다. **이를 통해 효율적이라고 판단되면 앞으로 코인 투자에 해당 방식을 활용할 생각입니다.**

2. 코인에 대한 기본적 정의와 백테스팅 시 고려할 투자방식 (서론)

KDI 경제교육 정보센터’에 따르면 비트코인은 2009년에 탄생한 글로벌 전자지불 네트워크이자 이를 기반으로 통용되는 디지털 화폐의 명칭입니다. 비트코인은 발행 총량이 정해져 있으며, 약 130여 년 후면 발행이 종료됩니다.

코인 시장에는 다양한 코인이 존재하며, 코인마다 발행 방식이 다릅니다. 비트코인의 경우 반감기라는 특징이 있습니다. 토스뱅크에 따르면 비트코인은 전체 발행량이 제한되어 있어 채굴자들에게 지급되는 보상이 약 4년마다 절반으로 줄어듭니다. 이를 반감기라고 하며, 지금까지 2012년, 2016년, 2020년에 반감기가 발생했고, 이 시기에 비트코인 가격이 크게 상승했습니다. 하지만 반감기의 효과는 시간이 지날수록 약해지며, 이후 큰 폭의 조정이 뒤따르는 경우도 있습니다.

이러한 비트코인의 특성으로 인해, 조정이 올 때보다 가격이 상승하기 시작하는 시점에 매수하는 것이 유리합니다. 조정의 바닥이 어디인지 정확히 판단하기 어렵기 때문입니다. 특히 200일 이동평균선 돌파 시점인 골든 크로스 지점을 중요한 매수 타이밍으로 봅니다. 주식 시장에서도 200

일 이동평균선이 깨지면 시장 참여자들의 관심이 줄어든 것으로 판단되는데, 코인 시장에서도 마찬가지입니다. 따라서, 200일 이동평균선 위에 있을 때 매수하는 것이 적절합니다.

120일 이동평균선 역시 중요한 기준점으로 작용합니다. 일반적인 상승장에서 조정이 발생하면 120일 이동평균선이 지지선 역할을 하는 경우가 많습니다. 김학균 한국증권 선임연구원에 따르면, 120일 이동평균선은 대세 상승 과정에서의 중간 반락 시 양호한 조정의 기준선이 되는 경우가 많다고 합니다. 실제로 저 역시 정배열 상태에서는 120일 이동평균선을 중요한 지표로 사용하고 있습니다.

거래량도 주식이나 코인 거래에서 중요한 요소입니다. 거래량이 증가했다는 것은 투자자들의 관심을 보여주는 지표이며, 높은 거래량을 동반한 상승은 지지선 역할을 하기도 합니다. 주식 시장에서는 거래량이 폭발하며 상승하는 지점을 매수 타이밍으로 보는 경우가 많고, 코인 시장에서도 마찬가지입니다. 거래량이 급증하며 상승했다면, 이는 투자자들이 해당 지점을 바닥으로 간주하거나 큰 호재가 발생했음을 의미할 가능성이 있습니다.

이동평균선을 판단할 때, 저는 20일, 50일, 120일 이동평균선과 200일 이동평균선이 정배열을 이루는지를 중요하게 봅니다. 장기 이동평균선이 정배열이라는 것은 상승장이 이어지고 있다는 신호로 해석되며, 대세 폭락장은 아니라는 뜻으로 이해됩니다. 아무리 매수 신호가 나타나더라도 이동평균선이 역배열 상태라면 신중하게 판단하게 됩니다. 이러한 기준은 코인 시장에도 적용할 수 있습니다.

FTD(추세 전환 매수법)는 윌리엄 오닐이 그의 저서에서 제시한 매수 방법입니다. 그는 떨어지는 칼날을 잡기보다는, 하락장이 충분히 마무리되고 상승 신호가 나타나는 시점에 매수하라고 주장했습니다. 첫 번째 조건으로는 시장이 충분히 하락한 후, 하루 동안 1.25% 이상의 상승이 나타나야 합니다. 그리고 첫날 저점보다 가격이 떨어지지 않은 상태로 3일이 지나야 하며, 4일부터 7일까지 거래량이 계속 증가하고 가격 상승이 이어져야 합니다. 코인 시장의 변동성이 주식 시장과 다르기 때문에, 최근 1년간의 변동성을 기준으로 상승 비율을 조정하여 적용하려고 합니다.

이를 종합해 표를 만들고, 다음과 같은 지점을 체크하려고 합니다.

1. 200일 이동평균선을 돌파하는 골든 크로스 지점
2. 200일 이동평균선 위에서 120일 이동평균선에 도달하는 지점 (조정 시 120일선에 닿거나 이를 돌파하려는 골든 크로스 시점)
3. 200일 이동평균선 위에서 FTD 조건을 만족하는 지점
4. 200일 이동평균선 위에서 거래량이 평소보다 두 배 이상 증가하며 상승하는 지점
5. 거래량이 한 달 평균 거래량보다 세 배 이상 증가하며 상승한 지점 (200일선 기준 없이 체크)
6. 200일 이동평균선 위에서 20일, 50일, 120일, 200일 이동평균선이 정배열을 이루며 FTD 조건을 만족하는 지점
7. 200일 이동평균선 위에서 이동평균선이 정배열 상태를 이루며 120일선을 터치한 지점
8. 200일 이동평균선 위에서 이동평균선이 정배열 상태를 이루며 평균 거래량이 두 배 이상 증가하며 상승한 지점

거래량이 한 달 평균보다 세 배 이상 터진 지점은 특별히 200일 이동평균선 기준을 적용하지 않을 것입니다. 이는 거래량 급증이 매우 큰 호재를 의미한다고 판단되기 때문입니다. 각 체크 지점별로 한 달, 세 달, 여섯 달 후 수익률을 확인하여 승률을 계산하려고 합니다. 제 투자 스타일상 최대 보유 기간은 여섯 달로 설정했으며, 이번 프로젝트의 목적이 제 투자 전략 개선을 위한

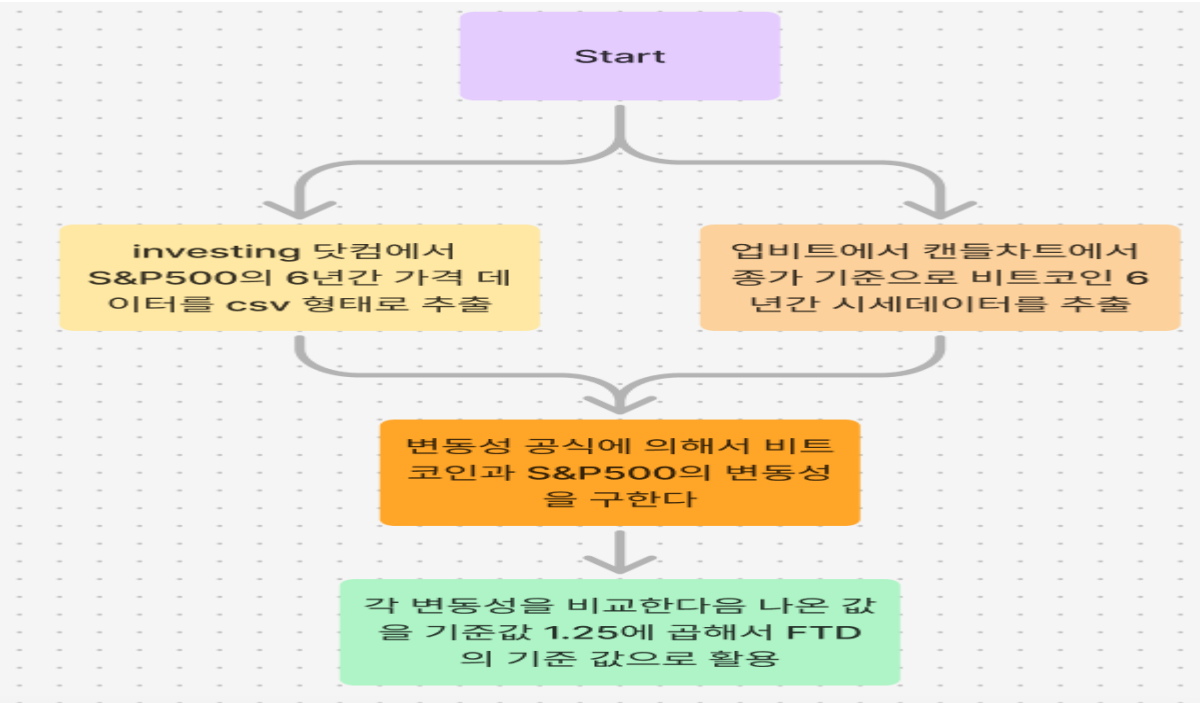
것이므로 이 기간을 기준으로 분석을 진행할 것입니다.

3. 변동성 체크 (본론)

3-0. 변동성 순서도

<고려한점>

비트코인의 변동성 체크를 위한 데이터 기준을 6년으로 설정한 이유는 업비트에서 제공하는 비트코인 데이터가 6년치밖에 없기 때문입니다. 비트코인 가격은 국내와 해외에서 차이가 있을 수 있는데, 이를 김치 프리미엄이라고 부릅니다. 김치 프리미엄을 별도로 고려하지 않은 이유는, 향후 코인 투자를 본격적으로 진행하게 되더라도 레버리지 투자가 아닌 경우 업비트 API를 주로 사용할 가능성이 높다고 판단했기 때문입니다. 따라서 글로벌 1위 코인 거래소인 바이낸스 API 대신 업비트 API를 기반으로 데이터를 수집집하고 분석하였습니다.



<변동성 구하는 순서도>

3-1. 에센피와 비트코인 데이터 추출

그럼 먼저, investing 닷컴에서 가격 데이터를 추출하겠습니다. S&P 500데이터는 인베스팅 닷컴에서 이미 CSV파일이 만들어져있어서 다운만 하면 됩니다. 다운 받아보니 거래량 데이터는 누락되있던데 굳이 필요한 데이터는 아니므로 패스하겠습니다.

날짜	종가	시가	고가	저가	거래량	변동 %
2024-12-20	5,930.85	5,842.00	5,982.06	5,832.30	0.00K	1.09%
2024-12-19	5,867.08	5,912.71	5,935.52	5,866.07	0.00K	-0.09%
2024-12-18	5,872.16	6,047.65	6,070.67	5,867.79	0.00K	-2.95%
2024-12-17	6,050.61	6,052.55	6,057.68	6,035.19	0.00K	-0.39%
2024-12-16	6,074.08	6,063.79	6,085.19	6,059.14	0.00K	0.38%
2024-12-13	6,051.09	6,068.17	6,078.58	6,035.77	0.00K	0.00%
2024-12-12	6,051.25	6,074.29	6,079.68	6,051.25	0.00K	-0.54%
2024-12-11	6,084.19	6,060.15	6,092.59	6,060.15	0.00K	0.82%
2024-12-10	6,034.91	6,057.59	6,065.40	6,029.89	0.00K	-0.30%
2024-12-09	6,052.85	6,083.01	6,088.51	6,048.63	0.00K	-0.61%
2024-12-06	6,090.27	6,081.38	6,099.97	6,079.98	0.00K	0.25%
2024-12-05	6,075.11	6,089.03	6,094.55	6,072.90	0.00K	-0.19%
2024-12-04	6,086.49	6,069.39	6,089.84	6,061.06	0.00K	0.61%
2024-12-03	6,049.88	6,042.97	6,052.07	6,033.39	0.00K	0.05%
2024-12-02	6,047.15	6,040.11	6,053.58	6,035.33	0.00K	0.24%
2024-11-29	6,032.38	6,003.98	6,044.17	6,003.98	0.00K	0.56%
2024-11-27	5,998.74	6,014.11	6,020.16	5,984.87	0.00K	-0.38%
2024-11-26	6,021.63	6,000.03	6,025.42	5,992.27	0.00K	0.57%

<S&P500 인베스팅 닷컴 데이터>

비트코인은 아쉽게도 인베스팅 닷컴에서 CSV로 정리된 데이터를 제공하지 않기 때문에, 직접 업비트 API를 활용해 데이터를 수집하고 CSV로 정리해야 했습니다. 이를 위해 업비트 API를 사용하여 데이터를 가져오는 코드를 작성했습니다.

우선, fetch_candles라는 함수를 만들어 업비트 API에서 비트코인 데이터를 가져오도록 설정했습니다. 데이터를 가져온 후, 요청이 성공적으로 처리되었는지 확인하기 위해 상태 코드가 200(성공)일 경우에만 데이터를 처리하도록 코드를 작성했습니다. 처음에는 이 검증 코드를 포함하지 않았는데, 이로 인해 API 요청 주소를 잘못 입력했음에도 이를 알아차리지 못해 많은 시간을 허비하게 되었습니다.

이 문제를 해결하기 위해 요청이 실패한 경우 "요청이 실패했습니다"라는 메시지를 출력하며 프로그램이 종료되도록 로직을 추가했습니다. 이를 통해 API 요청 주소가 잘못되었음을 빠르게 인지할 수 있었고, 주소를 수정한 뒤 데이터를 원활하게 처리할 수 있었습니다. 이러한 과정 덕분에 데이터 수집이 훨씬 안정적이고 효율적으로 이루어졌습니다.

```python
# 업비트 API에서 캔들 데이터를 가져오는 함수
def fetch_candles(market, count=200, to=None):
    url = "https://api.upbit.com/v1/candles/days"
    params = {"market": market, "count": count}
    if to:
        params["to"] = to
    response = requests.get(url, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"API 요청 실패: {response.status_code}, {response.text}")
        return []
```

<캔들데이터 가져오는 함수부분>

그 다음에 가져온 데이터를 저장하고 바탕화면에 저장하는 함수를 짜보았습니다. 위 에센피 CSV파일처럼 날짜, 종가, 시가, 고가, 저가, 거래량, 변동성을 추출해보았습니다. 코인 데이터는 나중에 거래량을 활용해야되므로 저장시켜 놓았습니다.

```python
# 비트코인 가격 데이터를 수집하고 csv로 저장하는 함수
def fetch_bitcoin_data_to_csv(filename, market="KRW-BTC", days=365):
    # 바탕화면 경로를 C:\Users\user\Desktop으로 설정
    desktop = "C:\\Users\\user\\Desktop"
    full_path = os.path.join(desktop, filename)

    all_data = []
    to_date = None
    while len(all_data) < days:
        data = fetch_candles(market, to=to_date)
        if not data:
            break
        all_data.extend(data)
        to_date = data[-1]["candle_date_time_utc"]  # 마지막 캔들의 날짜
        to_date = datetime.strptime(to_date, "%Y-%m-%dT%H:%M:%S") - timedelta(seconds=1)
        to_date = to_date.strftime("%Y-%m-%dT%H:%M:%S")
        time.sleep(0.5)  # API 추출 간 간격을 0.5초로 설정

    # 필요한 데이터만 추출
    candles = [
        {
            "날짜": item["candle_date_time_kst"].split("T")[0],
            "종가": item["trade_price"],
            "시가": item["opening_price"],
            "고가": item["high_price"],
            "저가": item["low_price"],
            "거래량": item["candle_acc_trade_volume"],
            "변동 %": f"{((item['trade_price'] - item['opening_price']) / item['opening_price']) * 100:.2f}%",
        }
        for item in all_data
    ]
```

<비트코인 데이터를 CSV파일로 저장하는 함수>

그다음으로, DataFrame.to_csv() 메서드를 사용해 데이터를 Excel 형식의 CSV 파일로 저장했습니다. 저장 시 encoding="utf-8-sig"를 지정하여 UTF-8 인코딩 방식을 사용했습니다. 이를 통해 Microsoft Excel에서 한글이 깨지는 문제를 방지할 수 있었으며, BOM(Byte Order Mark)을 추가해 파일의 호환성을 높였습니다. 또한, index=False 옵션을 사용해 DataFrame의 기본 인덱스가 파일에 포함되지 않도록 설정했습니다.

이 과정에서 파일이 제대로 저장되지 않는 경우를 대비해, 저장 실패 시 오류 메시지를 출력하도록 코드를 작성했습니다. 이를 통해 저장 과정에서 발생할 수 있는 문제를 즉시 파악하고 수정할 수 있도록 했습니다. 이러한 검증 과정을 추가하면서 데이터 저장 작업이 더 안정적이고 신뢰성 있게 수행될 수 있었습니다.

```python
    # DataFrame으로 변환 후 저장
    try:
        df = pd.DataFrame(candles)
        df.to_csv(full_path, index=False, encoding="utf-8-sig")
        print(f"데이터가 바탕화면에 저장되었습니다: {full_path}")
    except Exception as e:
        print(f"파일 저장 중 오류가 발생했습니다: {e}")

# 실행
fetch_bitcoin_data_to_csv("bitcoin_price_data.csv", days=6*365)
```

<EXCEL 형식의 DATA frame으로 변환하여 저장시키는 함수>

뽑아낸 비트코인 데이터는 다음과 같습니다.

날짜	종가	시가	고가	저가	거래량	변동 %
2024-12-21	1.48E+08	1.48E+08	1.49E+08	1.47E+08	1310.065	-0.16%
2024-12-20	1.48E+08	1.47E+08	1.48E+08	1.4E+08	11567.35	0.45%
2024-12-19	1.47E+08	1.48E+08	1.53E+08	1.45E+08	8471.893	-0.19%
2024-12-18	1.48E+08	1.55E+08	1.56E+08	1.47E+08	8610.823	-4.81%
2024-12-17	1.55E+08	1.53E+08	1.57E+08	1.53E+08	6186.28	1.19%
2024-12-16	1.53E+08	1.49E+08	1.55E+08	1.49E+08	7128.618	2.79%
2024-12-15	1.49E+08	1.46E+08	1.5E+08	1.46E+08	4680.013	2.27%
2024-12-14	1.46E+08	1.44E+08	1.46E+08	1.44E+08	3195.757	1.04%
2024-12-13	1.44E+08	1.43E+08	1.45E+08	1.42E+08	2524.282	1.14%
2024-12-12	1.43E+08	1.43E+08	1.45E+08	1.42E+08	4109.653	-0.25%
2024-12-11	1.43E+08	1.38E+08	1.44E+08	1.37E+08	5786.503	3.55%
2024-12-10	1.38E+08	1.39E+08	1.4E+08	1.35E+08	5715.111	-0.52%
2024-12-09	1.39E+08	1.41E+08	1.41E+08	1.36E+08	6238.332	-1.55%
2024-12-08	1.41E+08	1.39E+08	1.41E+08	1.39E+08	2575.925	1.46%
2024-12-07	1.39E+08	1.39E+08	1.4E+08	1.38E+08	2646.581	-0.32%
2024-12-06	1.39E+08	1.38E+08	1.42E+08	1.36E+08	6179.758	1.34%
2024-12-05	1.38E+08	1.39E+08	1.46E+08	1.34E+08	13579.29	-0.91%
2024-12-04	1.39E+08	1.34E+08	1.4E+08	1.33E+08	6969.871	3.77%

3-2. 비트코인과 에센피의 변동성 비교분석

변동성을 계산하는 데에는 표준편차를 활용합니다. 이는 각 날짜별 시세 변화를 기반으로 계산됩니다. 구체적으로 변동성을 구하는 공식은 다음과 같습니다.

Annualized Volatility = Daily Std Dev × $\sqrt{252}$

여기서 252는 금융 시장에서 1년 동안의 평균 거래일 수를 나타냅니다.

이미 생성한 CSV 파일에 각 날짜별 시세 변화를 백분율(%)로 계산한 데이터가 포함되어 있으므로, 이 데이터를 이용해 변동성을 구할 수 있습니다. 이를 위해, numpy.std() 함수를 활용하여 일별 수익률의 표준편차를 계산한 후, 여기에 $\sqrt{252}$를 곱하여 연간 변동성을 도출합니다.

이 과정을 비트코인과 S&P 500 지수 각각에 적용하여 두 자산의 변동성을 비교할 수 있도록 코드를 작성했습니다. 코드는 다음과 같은 순서로 구성됩니다.

1. CSV 파일에서 날짜별 수익률 데이터를 불러옵니다.
2. numpy.std()를 사용하여 일별 수익률의 표준편차를 계산합니다.
3. 표준편차에 √252를 곱해 연간 변동성을 계산합니다.
4. 계산된 변동성을 출력하거나 파일로 저장합니다.

```python
# 파일 경로
bitcoin_file_path = 'bitcoin_price_data.csv'
sp500_file_path = 'S&P.csv'

# 데이터 불러오기
bitcoin_data = pd.read_csv(bitcoin_file_path)
sp500_data = pd.read_csv(sp500_file_path)

# 변동 % 컬럼의 % 기호 제거 및 숫자로 변환
bitcoin_data['변동 %'] = pd.to_numeric(bitcoin_data['변동 %'].str.replace('%', ''), errors='coerce') / 100
sp500_data['변동 %'] = pd.to_numeric(sp500_data['변동 %'].str.replace('%', ''), errors='coerce') / 100

# 변동성 계산 함수
def calculate_annualized_volatility(daily_returns):
    return np.std(daily_returns) * np.sqrt(252)  # 연간화

# 비트코인과 S&P 500의 변동성 계산 (변동 %를 바로 활용)
btc_volatility = calculate_annualized_volatility(bitcoin_data['변동 %'])
sp500_volatility = calculate_annualized_volatility(sp500_data['변동 %'])

# 결과 출력
print(f"비트코인의 연간 변동성: {btc_volatility:.2%}")
print(f"S&P 500의 연간 변동성: {sp500_volatility:.2%}")
```

<변동성 구하는 코드(numpy는 np로, pandas는 pd로 대응시켰다)>

```
비트코인의 연간 변동성: 47.93%
S&P 500의 연간 변동성: 19.70%
```

<변동성 결과값>

비트코인과 에센피의 연간 변동성이 약 2.43배(정확히는 2.4329929239….)이므로 1.25에 2.4329929239을 곱해서 약 3.04 (3.0412411549..) 를 기준으로 FTD를 짜보겠습니다.

3-3. 여기서 오류:

1. 6년간 변동성을 계산한 것이 아니라 1년간 변동성만 계산을 했습니다.

- 그래서 6년을 각각 1년 단위로 나눠서 구한다음 그 값을 6으로 나누려고 합니다

2. 변동성 공식의 원리를 모르고 사용했는데 252가 거래 기간을 뜻하는 것이었습니다. 그런데 비트코인의 거래 기간은 365일 이므로 365로 다시 계산을 했어야했습니다.

- 그래서 S&P 500과 코인의 거래 일자를 구분을 했습니다.

```python
# 연도별 변동성 계산 함수
def calculate_annualized_volatility_by_year(data, trading_days):
    data['year'] = data['날짜'].dt.year  # 연도별 그룹화
    annual_volatility = data.groupby('year')['daily_return'].std() * np.sqrt(trading_days)
    return annual_volatility

# 비트코인: 365일 기준
btc_annual_volatility = calculate_annualized_volatility_by_year(bitcoin_data, 365)

# S&P 500: 252일 기준
sp500_annual_volatility = calculate_annualized_volatility_by_year(sp500_data, 252)
```

<수정된 코드의 핵심 부분>

```
비트코인의 평균 연간 변동성 (6년 기준): 60.33%
S&P 500의 평균 연간 변동성 (6년 기준): 13.19%
```

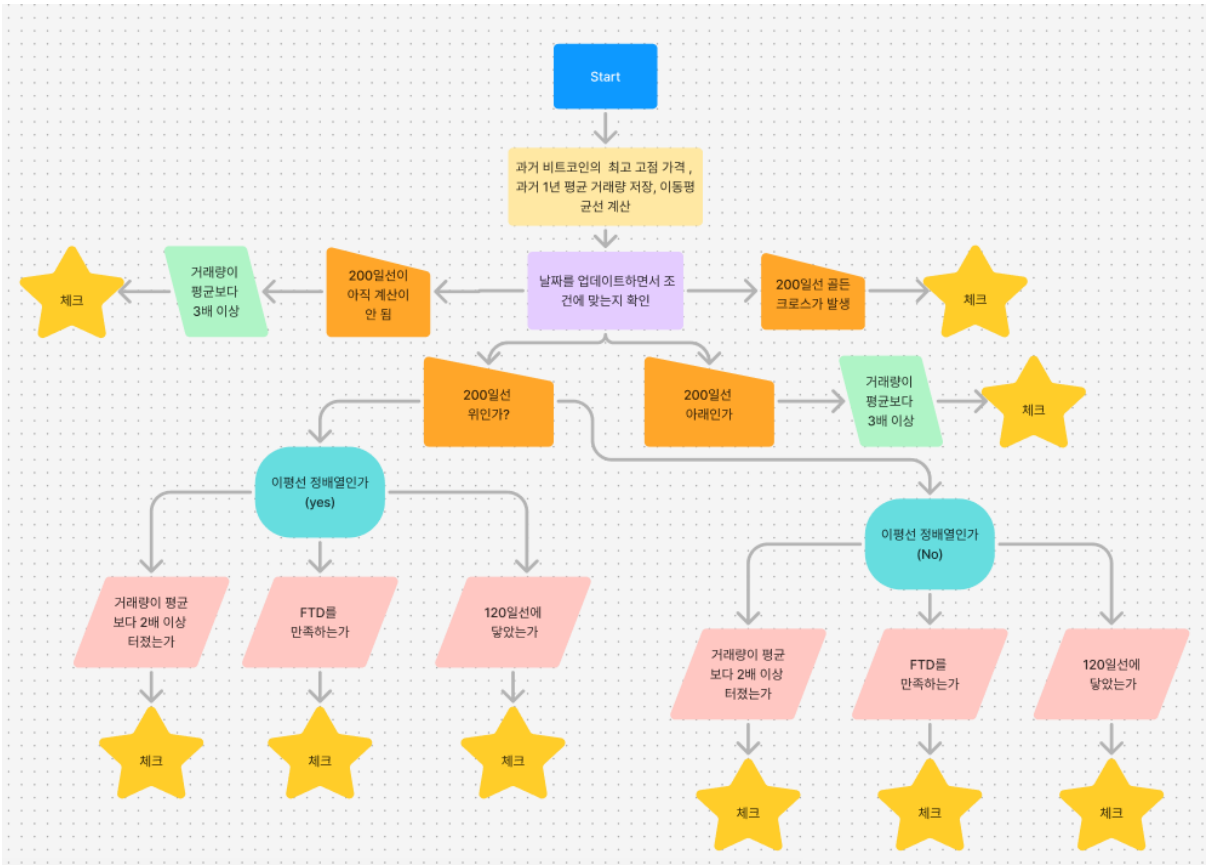
<결과 값>

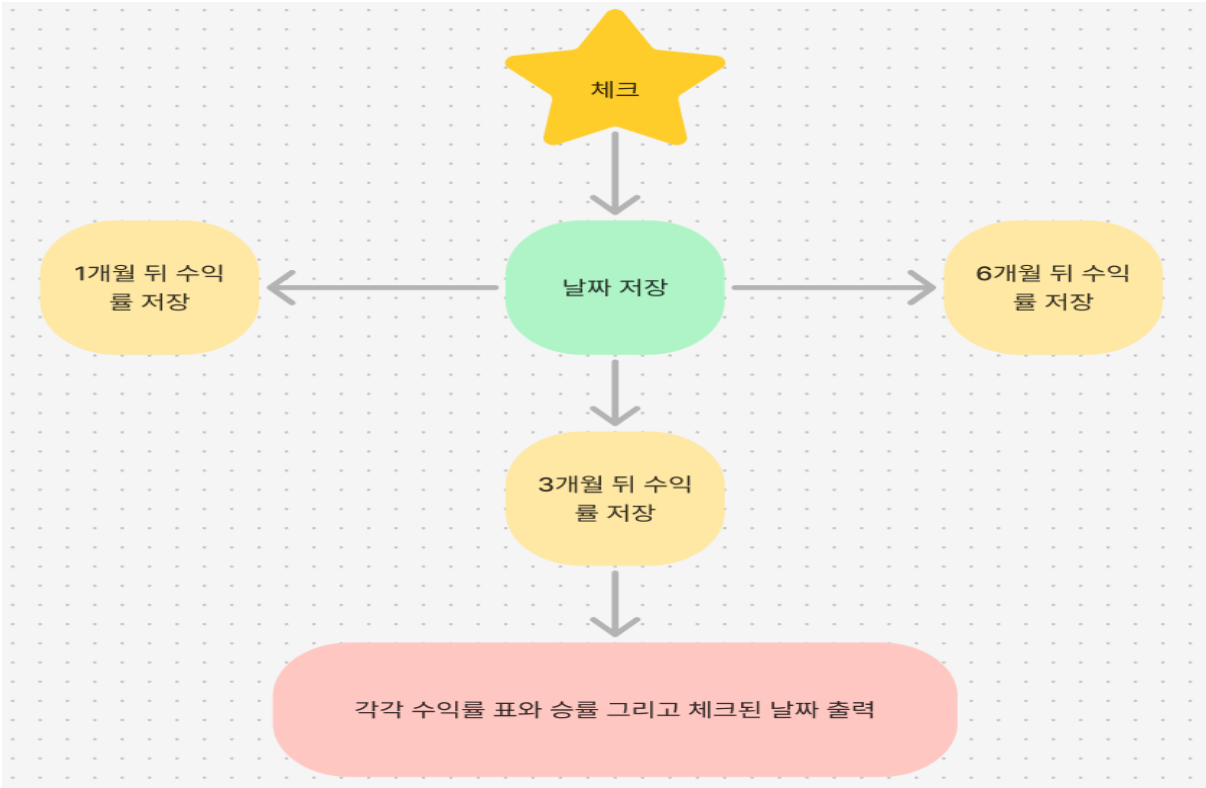
비트코인과 S&P 500의 변동성 차이는 약 4.57391963609배입니다. 그러므로 1.25*4.57391963609를 하면 5.717399545111이 나옵니다. 즉 5.72 퍼센트 이상 오른 날을 FTD의 기준으로 잡겠습니다.

4. 백 테스팅(본론)

4-0. 백 테스팅 순서도

아래의 순서도대로 코드를 짜보려고 합니다. 첫 번째 순서도는 특정 기준을 만족한 날짜를 체크하는 것까지의 과정을 담았고, 두 번째 순서도는 출력하는 과정까지의 순서도를 담았습니다.





4-1. 관련 코드 설명 <PART1>

1. 사용한 데이터

사용할 데이터는 업비트 api에서 가져온 밑에의 데이터를 활용합니다. 다만 코랩에 무슨 이유에서인지 파일이 업로드가 안 되서 정리된 CSV파일을 주는 게 아니라 업비트에서 직접 데이터를 가져와서 활용하는 방식을 선택했습니다. 그러나 밑의 데이터와 비교했을 때 변동%를 제외하면 다르지 않기 때문에 어떤 데이터를 활용하는지 보여주기 위해서 가져왔습니다.

날짜	종가	시가	고가	저가	거래량	변동 %
2024-12-21	1.48E+08	1.48E+08	1.49E+08	1.47E+08	1310.065	-0.16%
2024-12-20	1.48E+08	1.47E+08	1.48E+08	1.4E+08	11567.35	0.45%
2024-12-19	1.47E+08	1.48E+08	1.53E+08	1.45E+08	8471.893	-0.19%
2024-12-18	1.48E+08	1.55E+08	1.56E+08	1.47E+08	8610.823	-4.81%
2024-12-17	1.55E+08	1.53E+08	1.57E+08	1.53E+08	6186.28	1.19%
2024-12-16	1.53E+08	1.49E+08	1.55E+08	1.49E+08	7128.618	2.79%
2024-12-15	1.49E+08	1.46E+08	1.5E+08	1.46E+08	4680.013	2.27%
2024-12-14	1.46E+08	1.44E+08	1.46E+08	1.44E+08	3195.757	1.04%
2024-12-13	1.44E+08	1.43E+08	1.45E+08	1.42E+08	2524.282	1.14%
2024-12-12	1.43E+08	1.43E+08	1.45E+08	1.42E+08	4109.653	-0.25%
2024-12-11	1.43E+08	1.38E+08	1.44E+08	1.37E+08	5786.503	3.55%
2024-12-10	1.38E+08	1.39E+08	1.4E+08	1.35E+08	5715.111	-0.52%
2024-12-09	1.39E+08	1.41E+08	1.41E+08	1.36E+08	6238.332	-1.55%
2024-12-08	1.41E+08	1.39E+08	1.41E+08	1.39E+08	2575.925	1.46%
2024-12-07	1.39E+08	1.39E+08	1.4E+08	1.38E+08	2646.581	-0.32%
2024-12-06	1.39E+08	1.38E+08	1.42E+08	1.36E+08	6179.758	1.34%
2024-12-05	1.38E+08	1.39E+08	1.46E+08	1.34E+08	13579.29	-0.91%
2024-12-04	1.39E+08	1.34E+08	1.4E+08	1.33E+08	6969.871	3.77%

< 활용할 데이터 자료 >

```python
def fetch_candles_bulk(market, count, to=None):
    url = f"https://api.upbit.com/v1/candles/days"
    params = {"market": market, "count": count}
    if to:
        params["to"] = to
    response = requests.get(url, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"API 요청 실패: 상태 코드 {response.status_code}, 메시지: {response.text}")
        return []

# 데이터를 가져와 DataFrame으로 변환
def load_data_from_upbit(market, days):
    candles = []
    to_date = None

    while len(candles) < days:
        batch = fetch_candles_bulk(market, min(200, days - len(candles)), to=to_date)
        if not batch:
            break
        candles.extend(batch)
        to_date = batch[-1]["candle_date_time_utc"]
        time.sleep(0.5)  # API 요청 간격 조정

    candles.reverse()  # 시간 순서 정렬
    data = {
        "날짜": [datetime.strptime(candle["candle_date_time_utc"], "%Y-%m-%dT%H:%M:%S") for candle in candles],
        "종가": [candle["trade_price"] for candle in candles],
        "시가": [candle["opening_price"] for candle in candles],
        "저가": [candle["low_price"] for candle in candles],
        "거래량": [candle["candle_acc_trade_volume"] for candle in candles],
    }
    return pd.DataFrame(data)
```

<데이터를 가져오는 코드>

업비트 api에서 코드를 가져온다음 dataframe으로 만든건 위의 변동성을 구할 때 설명했으니 스킵하겠습니다.

2. 이동평균선을 구하는 코드

이동평균선의 정의는 과거의 일정 기간 주가의 평균치를 뜻합니다. 예를 들면 60일 선 이동평균선은 60일 동안의 주식 종가의 평균치를 매일 점으로 표시하고 이를 계속 이어서 선으로 표시한 것을 뜻합니다. 그런데 이 과제동안에 선을 그릴 필요는 없으므로 그냥 각 날짜마다 해당되는 이동평균선을 계산하면 됩니다.

```
def calculate_moving_average(prices, window):
    series = pd.Series(prices)
    return series.shift(1).rolling(window=window).mean().tolist()
```

<이동평균 계산 함수>

pd.Series(prices)는 종가 데이터를 1차원 배열 형태로 저장한 데이터 구조입니다. 이를 활용해 rolling 함수를 사용하면 지정된 기간(window)만큼 데이터를 묶어서 계산할 수 있습니다. 이후, 이동평균선을 계산하기 위해 묶인 데이터의 평균을 구한 뒤, .tolist()를 사용해 파이썬 리스트로 변환하여 각각의 이동평균선에 저장합니다.

이 함수로 이동평균선 계산을 마치면, 다음 단계에서 별도의 함수로 각 이동평균선을 저장할 수 있습니다. 또한, 계산 과정에서 약간의 디테일을 추가하기 위해 shift를 사용했습니다. shift는 데이터셋을 한 칸씩 이동시켜, 전날까지의 데이터만 사용하도록 만듭니다. 이는 당일 시가에서 종가까지의 데이터를 기반으로 이동평균선이 해당 범위를 지나가는지 확인하기 위한 것으로, 당일 데이터를 포함하지 않도록 조정한 것입니다. 이 방식은 이동평균선을 보다 정확히 활용하기 위한 중요한 요소입니다.

```
# 이동평균 및 거래량 평균 계산
def calculate_moving_averages(df):
    df['MA_20'] = calculate_moving_average(df['종가'], 20)
    df['MA_50'] = calculate_moving_average(df['종가'], 50)
    df['MA_120'] = calculate_moving_average(df['종가'], 120)
    df['MA_200'] = calculate_moving_average(df['종가'], 200)
    return df
```

<각각의 이동평균선 계산식>

3. 평균 거래량 구하는 코드

rolling_volume은 각 날짜별 365일 평균 거래량을 저장하기 위한 빈 리스트를 생성하는 역할을 합니다. current_avg는 현재까지 계산된 평균값을 저장하기 위해 None으로 초기화됩니다.

코드는 DataFrame의 모든 행을 순회하며 각 날짜의 거래량 데이터를 가져옵니다.

- **365일 이전**: 초기 365일 동안에는 현재까지의 모든 거래량의 단순 평균을 계산하여 rolling_volume에 추가합니다.
- **365일 이후**: 365일이 지난 시점부터는 이전 평균값에서 365일 전의 거래량을 제외하고

새로운 거래량을 추가한 뒤, 이를 365로 나누어 평균을 갱신합니다. 이를 통해 계산 속도를 개선할 수 있습니다.

계산된 평균은 rolling_volume 리스트에 계속 추가되며, 모든 행에 대한 계산이 완료되면 rolling_volume을 DataFrame의 새로운 열인 'Volume_Avg_365'에 추가합니다. 결과적으로 각 날짜별로 이전 365일 동안의 평균 거래량을 효율적으로 구할 수 있습니다.

이 접근 방식은 과거 데이터의 이동 평균을 계산하는 데 유용하며, 데이터의 연속성을 유지하면서 계산량을 줄이는 데 기여합니다.

```python
def calculate_rolling_volume_average(df):
    rolling_volume = []
    current_avg = None

    for i in range(len(df)):
        today_volume = df.iloc[i]['거래량']

        if i < 365:
            # 365일 되기 전까지는 현재까지의 평균 계산
            if current_avg is None:
                current_avg = today_volume
            else:
                current_avg = (current_avg * i + today_volume) / (i + 1)
        else:
            # 365일 이후부터는 이전 평균에서 오래된 값 빼고 새 값 추가
            old_volume = df.iloc[i - 365]['거래량']
            current_avg = current_avg + (today_volume - old_volume) / 365

        rolling_volume.append(current_avg)

    df['Volume_Avg_365'] = rolling_volume
    return df
```

<날짜별 1년 평균 거래량 구하는 코드>

4. 최고가 구하는 코드

데이터프레임의 종가 컬럼을 처음부터 순서대로 읽으면서, 각 날짜의 종가와 current_high를 비교해 더 큰 값을 current_high에 저장합니다. 이렇게 업데이트된 current_high를 all_time_highs 리스트에 추가하는 과정을 반복합니다. 예를 들어 종가가 [100, 80, 120, 90, 150]이라면, all_time_highs는 [100, 100, 120, 120, 150]이 됩니다. 마지막으로 이렇게 계산된 all_time_highs 리스트를 데이터프레임의 'All_Time_High'라는 새로운 열로 추가하고 데이터프레임을 반환합니다.

```python
def calculate_all_time_high(df):
    all_time_highs = []
    current_high = float('-inf')
    for price in df['종가']:
        current_high = max(current_high, price)
        all_time_highs.append(current_high)
    df['All_Time_High'] = all_time_highs
    return df
```

<최고가 구하는 코드>

5. 수익률 구하는 함수

함수는 세 가지 인자를 받습니다. 각각 df(데이터프레임), date_indices(조건이 발생한 날짜들의 인덱스), periods(몇 일 후의 수익률을 볼지, 예: [60, 120, 200]일)입니다.

처음에 periods 를 키로 하는 빈 딕셔너리(returns)를 만듭니다. 그 다음 조건이 발생한 각각의 날짜에 대해, 시작가격(그 날의 종가)을 저장하고, 미리 정의된 각 기간(30 일, 90 일, 180 일) 후의 가격과 비교하여 수익률을 계산합니다. 수익률은 ((미래가격 - 시작가격)/ 시작가격 * 100)으로 계산되어 퍼센트로 표시됩니다. 근데 중요한 게 미래에 데이터가 없는 경우를 고려를 해야 하므로 future 인덱스가 전체길이보다 길게 되면 None 을 저장시킵니다.

```
def calculate_returns(df, date_indices, periods):
    returns = {period: [] for period in periods}

    for date_idx in date_indices:
        start_price = df.iloc[date_idx]['종가']
        for period in periods:
            future_idx = date_idx + period
            if future_idx < len(df):
                future_price = df.iloc[future_idx]['종가']
                returns[period].append((future_price - start_price) / start_price * 100)
            else:
                returns[period].append(None)

    return returns
```

<수익률 구하는 함수>

6. 승률 계산함수

빈 딕셔너리를 만들고, returns의 각 기간(period)과 해당 기간의 수익률 리스트(values)를 순회합니다. 각 기간에 대해, None값을 제외한 실제 수익률들만 먼저 추출하여 valid_values에 저장하고, 이 중에서 0보다 큰 값(이익이 난 경우)들만 다시 추출하여 wins에 저장합니다. 그 다음, 이익이 난 경우의 수(len(wins))를 전체 유효한 경우의 수(len(valid_values))로 나누고 100을 곱해서 승률을 계산합니다. 만약 유효한 값이 하나도 없다면(valid_values가 비어있다면) 승률은 0으로 설정됩니다.

```
def calculate_win_rate(returns):
    win_rates = {}
    for period, values in returns.items():
        valid_values = [value for value in values if value is not None]
        wins = [value for value in valid_values if value > 0]
        win_rate = len(wins) / len(valid_values) * 100 if valid_values else 0
        win_rates[period] = win_rate
    return win_rates
```

<승률계산함수>

4-2. 관련코드 설명<PART 2>

1. 200 일 이평선 골든크로스

먼저, 200일 이동평균선이 존재하는지 확인하는 과정이 먼저입니다. 초기 199일간은 200일 이동평균선 계산자체가 불가능합니다. 두 번째로 어제의 종가가 200일 이동평균선 아래에 있었는지 확인해야 합니다. 왜냐하면 골든크로스는 이동평균선을 아래에서 위로 돌파하는 것이기 때문입니다. 세번째로 오늘 장중에 200일 이동평균선을 돌파했는지 확인을 해야하는데 시가와 종가 사이에 200일 이동평균선이 위치해야 합니다. 이 조건을 모두 만족하는 경우에만 result['Golden_Crosss_200'] 리스트에 추가합니다.

```
# Golden Cross 200 조건 확인
if row['MA_200'] is not None and df.iloc[i - 1]['종가'] < df.iloc[i - 1]['MA_200'] and min(row['시가'], row['종가']) <= row['MA_200'] <= max(row['시가'], row['종가']):
    results['Golden_Cross_200'].append(i)
```

<200일 골든크로스 함수>

2. 200 일 이평선 위에 있을 때 FTD

이 조건은 첫 번째로 200일 이평선 위에 있어야하고, 두 번째로 최고가 대비 70프로 미만, 세번째로 첫째 날에 3.04프로가 상승해야합니다. 만약 이 조건들을 만족시켰다면 향후 3일간 당일 저가를 깨면 안되고 4~7일 중 하루라도 3.04프로 이상 상승하고, 4일차부터 3.04프로가 상승하는 해당일까지 거래량이 계속 증가해야 합니다. 그리고 해당일이 되기 전까지 첫째날에 저가를 깨면 안 됩니다. 이 때문에 계속해서 and 조건으로 계속해서 row[저가]와 furure row[저가]를 확인하고 있습니다. Row는 첫째날의 데이터를 담은 행을 말합니다.

```
# FTD Above 200 조건 확인
if row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['종가'] < row['All_Time_High'] * 0.7:
    if row['종가'] >= row['시가'] * 1.0572:
        valid = True
        for j in range(1, 4):
            future_row = df.iloc[i + j]
            if future_row['저가'] < row['저가']:
                valid = False
                break

        if valid:
            for k in range(4, 8):
                future_row = df.iloc[i + k]
                if future_row['종가'] >= row['시가'] * 1.0572 and future_row['저가'] >= row['저가']:
                    volume_increasing = all(
                        df.iloc[i + n]['거래량'] > df.iloc[i + n - 1]['거래량']
                        for n in range(4, k + 1)
                    )
                    if volume_increasing:
                        results['FTD_Above_200'].append(i)
                        break
```

<200일 이평선 위에 있을 때 FTD 코드>

3. 200 일 위면서 거래량 2 배인 지점과 200 일 조건이 없으면서 거래량 3 배인 지점

일단 200일 위면서 거래량 2배인 지점을 계산해보면 일단 200일선이 있는지 체크하는 게 먼저이다. 그 다음 해당 row의 종가가 해당 row의 200일 이동평균지점보다 높게 위치해있으며 당일 row 거래량이 당일 row의 365일 평균 거래량의 2배인 지점을 체크한다.

단순히 거래량 3배 이상인 지점은 200일선 조건 없이 단순히 해당 row으 거래량이 해당 row의 365일 평균 거래량보다 높은지 확인한다. 주가는 200일선 위든 아래든 상관없다.

```
# High Volume 2x Above 200 조건 확인
if row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['거래량'] > row['Volume_Avg_365'] * 2:
    results['High_Volume_2x_above_200'].append(i)
```

```
# High Volume 3x 조건 확인 (200일선 조건 없음)
if row['거래량'] > row['Volume_Avg_365'] * 3:
    results['High_Volume_3x'].append(i)
```

<각 거래량 조건별 함수>

4. 200 일 선 위에 있으면서 120 일선을 터치하는 지점

일단 120일선이 존재하고 200일선도 존재해야 합니다. 그 다음 종가가 200일 선 위에 있어야 되고, 120일선이 해당일의 시가 종가 사이에 위치해야한다. 그래서 코드를 120일선이 시가와 종가 중 작은 값보다 크거나 같고, 시가와 종가 중 큰 값 보다 작거나 같다는 조건을 모두 만족시킬 때 저장하는 식으로 만들었습니다.

처음에 120일선을 터치하는 부분을 체크할 때 계속해서 120일선이 체크가 안 되는 것을 발견했습니다. 그런데 곰곰히 생각해보니 120일선과 비트코인의 1일 종가가 정확하게 겹치는 지점만 체크하다보니 하나도 체크가 하나도 안 되었습니다. 생각해보니 정확하게 같은 가격일 때가 없는 것이 당연했습니다. 그래서 시가랑 종가 사이에 120일선이 위치하는 지점을 체크하는 식으로 수정했습니다.

```
# MA 120 Touch Above 200 조건 확인 (주가가 200일선 위이고 종가와 시가 사이에 120일선 위치)
if row['MA_120'] is not None and row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['MA_120'] >= min(row['종가'], row['시가']) and row['MA_120'] <= max(row['종가'], row['시가']):
    results['MA_120_Touch_above_200'].append(i)
```

<120일 터치하면서 200일 선 위에 있는 조건 함수>

5. 정배열 계산

이제 나머지는 위의 코드에서 정배열 계산이 추가된 코드이므로 정배열 계산만 추가적으로 설명해보려고 합니다. 정배열의 정의가 상대적으로 장기 이평선이 단기 이평선보다 아래에 위치하는 것입니다. 제가 주식을 볼 때 가장 즐겨보는 사이트가 finviz라는 사이트인데 그 사이트에서는 기본적으로 20일선 50일선 200일선을 띄워주고, 지금까지 주식 거래를 할 때 이 이평선들을 이용했었습니다. 이걸 코인에도 적용을 할 예정이고, 120일 선을 추가하여 좀 더 정밀한 정배열을 판단하려고 합니다.

일단 20일선 50일선 120일선 200일선이 존재하는 지 체크를 해야 합니다. 그 다음 각각의 지점들이 상대적 위치를 충족하는지 부등호를 통해서 체크합니다.

```
if (
    row['MA_20'] is not None and row['MA_50'] is not None and
    row['MA_120'] is not None and row['MA_200'] is not None and
    row['MA_20'] > row['MA_50'] > row['MA_120'] > row['MA_200'] and  # 정배열 수정
    row['종가'] > row['MA_200'] and row['종가'] < row['All_Time_High'] * 0.7 and
    row['종가'] >= row['시가'] * 1.0572
):
```

<정배열식 예시>

4-3. 관련코드 설명<PART 3>

1개월 3개월 6개월 뒤 수익률을 볼 것이기에 periods를 30 60 180으로 설정하고, 수익률과 승률을 PART1에서 설명한 함수를 통해 계산합니다. 그런다음 출력합니다. 그 후 그래프를 만들어야 되므로 크기와 x축, y축에 관한 값을 입력하여 그래프를 출력하게 합니다.

```
# 결과 출력 및 시각화
def print_and_visualize_results(df, results):
    periods = [30, 90, 180]  # 1개월, 3개월, 6개월
    all_returns = {}

    for condition, indices in results.items():
        returns = calculate_returns(df, indices, periods)
        all_returns[condition] = returns
        win_rates = calculate_win_rate(returns)

        print(f"{condition} 승률:")
        print(f"1M: {win_rates[30]:.2f}%, 3M: {win_rates[90]:.2f}%, 6M: {win_rates[180]:.2f}%")
        print(f"{condition} 날짜:")
        print([df.iloc[idx]['날짜'].strftime('%Y-%m-%d') for idx in indices])

    # 시각화
    for period in periods:
        plt.figure(figsize=(12, 8))
        for condition, returns in all_returns.items():
            values = returns[period]
            plt.scatter([condition] * len(values), values, label=condition, alpha=0.7)
        plt.axhline(0, color="red", linestyle="--")
        plt.title(f"{period // 30}M Returns Comparison")
        plt.ylabel("Return (%)")
        plt.xlabel("Conditions")
        plt.legend()
        plt.show()
```

<출력 및 시각화 함수>

5. 결과 분석(본론)

2000일 간의 비트코인 데이터를 분석하신 백테스팅 결과에서 정말 흥미로운 인사이트들이 도출되었습니다.

특히 FTD(Follow Through Day) 전략이 보여준 승률이 정말 인상적이었습니다. 1개월과 3개월에서 75%의 높은 승률을 보였고, 6개월 기준으로는 무려 100%의 완벽한 승률을 기록했습니다. 이는 FTD가 매수 신호로서 얼마나 신뢰할 만한지를 잘 보여주는 결과입니다. 다만 이 전략을 정배열 조건과 함께 적용하려 했을 때 신호가 한 번도 발생하지 않았던 것은, FTD의 핵심인 '30% 이상 하락' 상황에서 이평선이 불가피하게 역배열되는 현상 때문이었습니다.

거래량 지표의 성과도 매우 인상적이었습니다. High_Volume_2x_above_200 전략은 67-73%라는 안정적인 승률을 보여주면서도, 가장 많은 매매 기회를 제공했습니다. 특히 3개월 기준으로 200%에 가까운 최고 수익률을 기록한 케이스도 있어서, 거래량 지표의 유효성을 잘 입증했습니다.

한편 이평선 정배열 조건에서는 흥미로운 점을 발견했습니다.
Sequential_MA_120_Touch_above_200 전략이 2000일 동안 한 번도 체크되지 않았던 것은, 120일 이동평균선을 터치할 정도로 가격이 하락하면 20일 이동평균선도 자연스럽게 하락하여 역배열이 발생할 수밖에 없다는 시장의 특성을 보여주었습니다.

이번 분석을 통해 FTD와 거래량 지표는 각각 독립적으로 사용할 때 매우 효과적인 전략이 될 수 있다는 것이 입증되었습니다. 다만 이를 정배열 조건과 결합하면 현실적으로 발생하기 어려운

상황이 된다는 점도 알 수 있었습니다. FTD나 120일선 터치는 모두 어느 정도의 하락을 전제로 하는데, 이런 상황에서는 정배열 조건을 만족시키기가 현실적으로 어렵기 때문입니다. 앞으로는 이평선 정배열 조건을 독립적인 전략으로 분리하여 검토하거나, 다른 보조지표와의 새로운 조합을 연구해보면 더 좋은 결과를 얻을 수 있을 것으로 보입니다.

```
Golden_Cross_200 승률:
1M: 73.33%, 3M: 80.00%, 6M: 60.00%
Golden_Cross_200 날짜:
['2020-01-28', '2020-04-27', '2021-08-13', '2021-08-19', '2021-09-12', '2021-09-14', '2021-10-01', '2022-01-01', '2023-01-20', '2023-08-19', '2023
FTD_Above_200 승률:
1M: 75.00%, 3M: 75.00%, 6M: 100.00%
FTD_Above_200 날짜:
['2023-01-20', '2023-02-15', '2023-03-13', '2023-06-21']
High_Volume_2x_above_200 승률:
1M: 67.00%, 3M: 73.00%, 6M: 71.00%
High_Volume_2x_above_200 날짜:
['2020-04-30', '2020-05-10', '2020-05-11', '2020-11-05', '2020-11-18', '2020-11-24', '2020-11-26', '2020-11-30', '2020-12-16', '2020-12-17', '2020
MA_120_Touch_above_200 승률:
1M: 76.47%, 3M: 70.59%, 6M: 94.12%
MA_120_Touch_above_200 날짜:
['2020-04-27', '2020-04-28', '2020-04-29', '2020-09-23', '2020-09-24', '2021-05-15', '2021-12-04', '2023-06-05', '2023-06-06', '2023-06-07', '2023
High_Volume_3x 승률:
1M: 68.89%, 3M: 80.00%, 6M: 71.11%
High_Volume_3x 날짜:
['2020-03-12', '2020-03-13', '2020-03-19', '2020-03-20', '2020-11-24', '2020-12-17', '2020-12-27', '2021-01-02', '2021-01-03', '2021-01-04', '2021
Sequential_FTD_above_200 승률:
1M: 0.00%, 3M: 0.00%, 6M: 0.00%
Sequential_FTD_above_200 날짜:
[]
Sequential_High_Volume_2x_above_200 승률:
1M: 75.00%, 3M: 72.22%, 6M: 63.89%
Sequential_High_Volume_2x_above_200 날짜:
['2020-11-05', '2020-11-18', '2020-11-24', '2020-11-26', '2020-11-30', '2020-12-16', '2020-12-17', '2020-12-21', '2020-12-23', '2020-12-27', '2021
Sequential_MA_120_Touch_above_200 승률:
1M: 0.00%, 3M: 0.00%, 6M: 0.00%
Sequential_MA_120_Touch_above_200 날짜:
[]
```

<승률과 체크날짜>

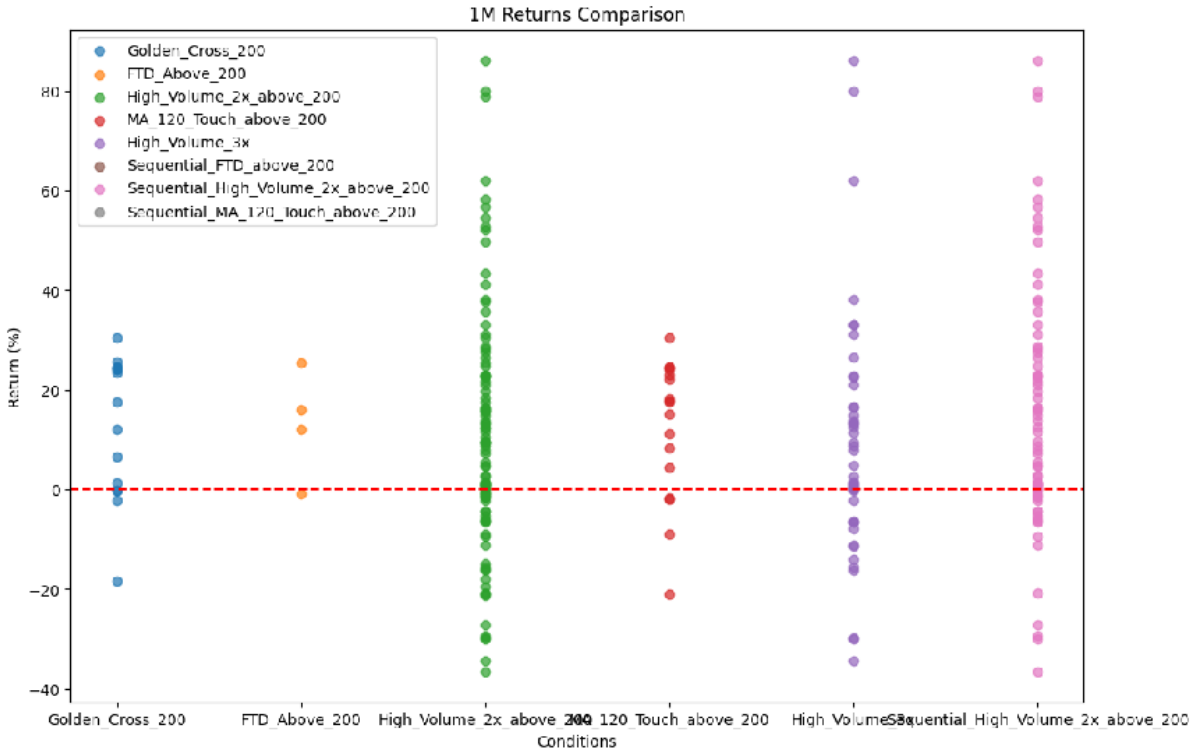
이번에는 표를 통해서 수익률을 비교해보겠습니다. 표의 X축은 체크하는 기준을 뜻하고, y축은 수익률을 뜻합니다. 그리고 체크된 지점의 수익률을 점으로 찍어서 모두 체크하였습니다. 이렇게 한 이유는 단순히 수익률 평균으로 하게 되면 코인의 경우 비정상적으로 수익률이 높거나 낮은 지점이 있을텐데 그런 지점들 때문에 평균의 오류가 생길 수 있다고 판단하였습니다. 그래서 표를 통해서 전체적으로 어떤 기준이 높은 수익률을 보이는 지 보기 위해서 표로 나타내었습니다.

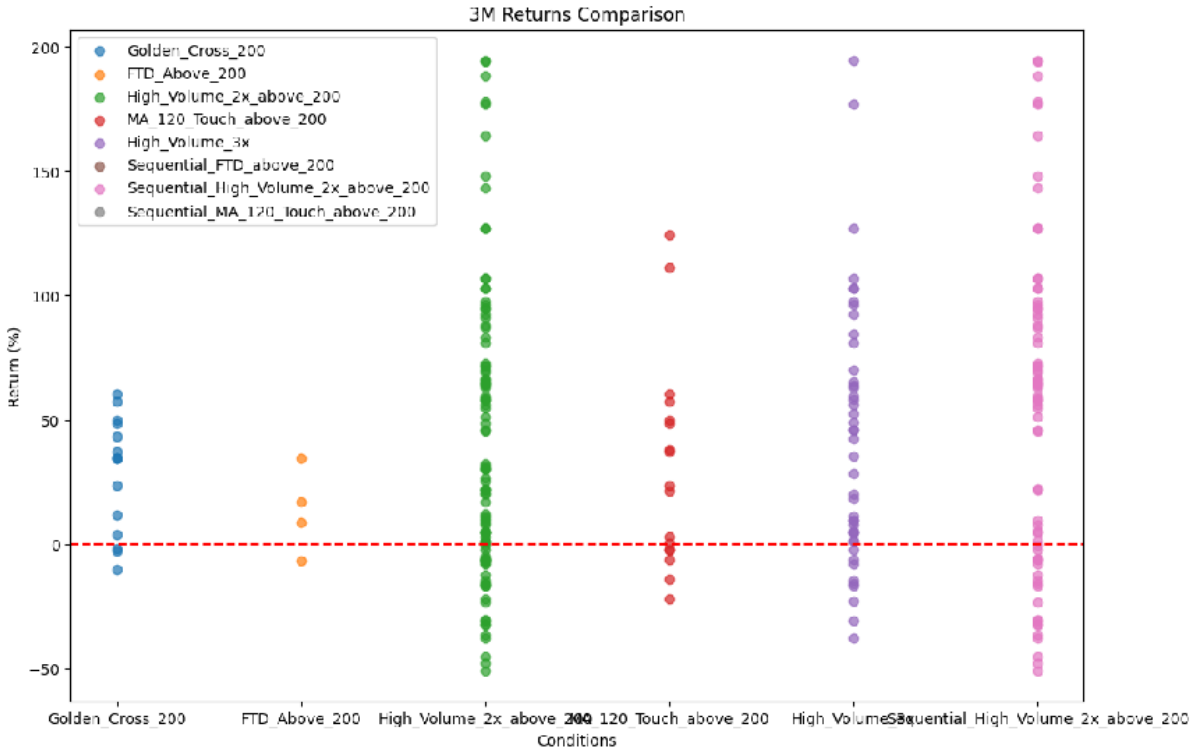
1개월 수익률을 보면, High_Volume_2x_above_200와 Sequential_High_Volume_2x_above_200가 가장 넓은 분포를 보여주고 있습니다. 특히 양의 수익률 구간에 많은 데이터가 분포되어 있습니다. Golden_Cross_200와 FTD_Above_200는 상대적으로 적은 횟수지만, 대부분 +20~30% 구간에 집중되어 있어 안정적인 수익을 보여주었습니다. 안정적인 수익률이 중요한 이유는 대부분의 사람들이 큰 변동성을 가질 때 큰 수익을 내기 힘들기 때문입니다.

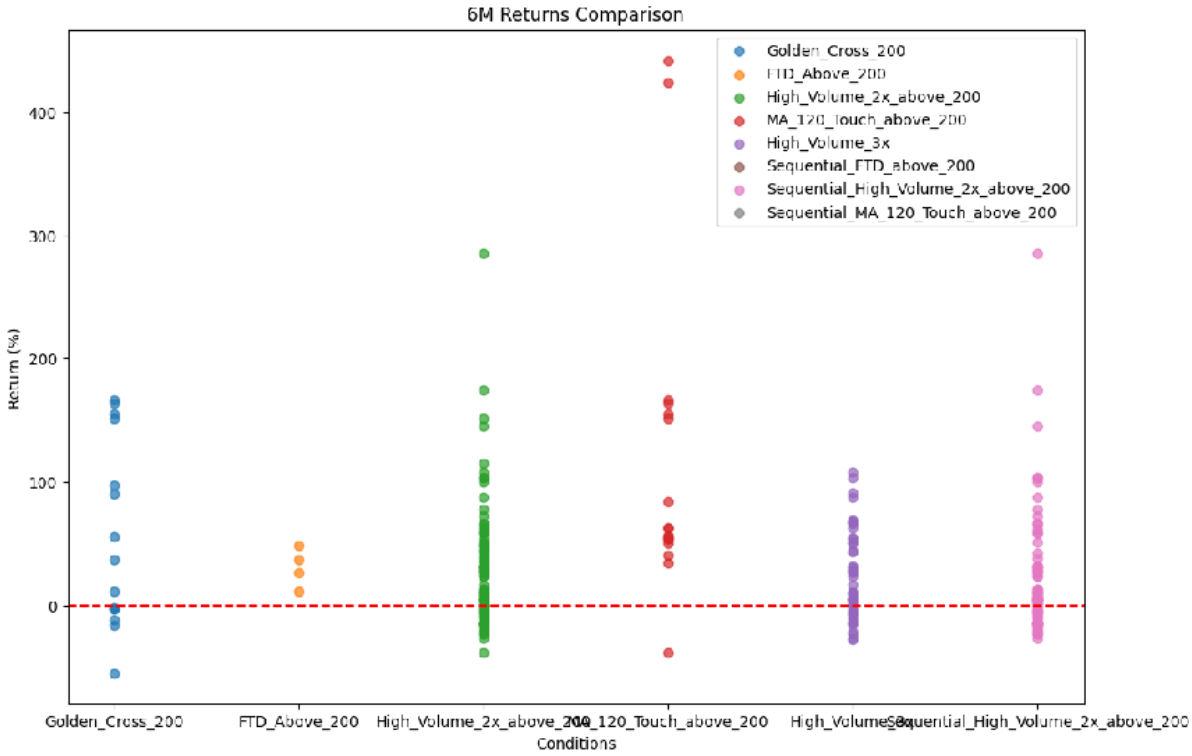
3개월 수익률에서는 전반적으로 수익률 범위가 더 넓어졌습니다. High_Volume_2x_above_200 전략이 가장 인상적인 성과를 보여주었는데, 최대 200% 가까운 수익률을 기록한 케이스도 있었습니다. FTD_Above_200는 여전히 안정적인 +20~40% 수익률을 유지했으며, MA_120_Touch_above_200도 대부분의 케이스가 양의 수익률을 기록했습니다.

6개월 수익률에서는 MA_120_Touch_above_200 전략이 가장 높은 수익률을 기록했는데, 최대 400% 이상의 수익률을 보여준 케이스도 있었습니다. High_Volume들은 6갸월 수익률도 꾸준히 좋은 성과를 보여주었으며, 대부분의 전략들이 장기적으로 양의 수익률을 기록했습니다.

위의 승률에서는 FTD가 두각을 나타냈던과 다르게 이번엔 거래량을 기준으로 잡은 기준들이 높은 성과를 보였습니다. 점이 찍힌 것들을 보았을 때 200일선 위에서 거래량이 2배가 터진 지점, 거래량이 3배가 터진 지점, 200일 선 위에 있으면서 거래량이 2배가 터진 지점이 다른 기준들보다 높은 수익률을 보인 것을 볼 수 있었습니다.


1M Returns Comparison
Golden_Cross_200
FTD_Above_200
High_Volume_2x_above_200
MA_120_Touch_above_200
High_Volume_3x
Sequential_FTD_above_200
Sequential_High_Volume_2x_above_200
Sequential_MA_120_Touch_above_200
Return (%)
80
60
40
20
0
−20
−40
Golden_Cross_200
FTD_Above_200
High_Volume_2x_above_200
MA_120_Touch_above_200
High_Volume_3x
Sequential_High_Volume_2x_above_200
Conditions


3M Returns Comparison
Golden_Cross_200
FTD_Above_200
High_Volume_2x_above_200
MA_120_Touch_above_200
High_Volume_3x
Sequential_FTD_above_200
Sequential_High_Volume_2x_above_200
Sequential_MA_120_Touch_above_200
Return (%)
200
150
100
50
0
−50
Golden_Cross_200
FTD_Above_200
High_Volume_2x_above_200
MA_120_Touch_above_200
High_Volume_3x
Sequential_High_Volume_2x_above_200
Conditions



<수익률 결과>

6. 결론 및 문제 해결

우선 FTD 전략은 1 개월 75%, 3 개월 75%, 6 개월 100%라는 놀라운 승률을 보여주었고, 수익률도 1 개월 20~30%, 3 개월 40% 정도로 안정적인 성과를 보여주었습니다. 거래량 지표(High_Volume_2x_above_200)의 경우도 67-73%의 높은 승률과 함께 최대 200%에 달하는 수익률을 기록했으며, 특히 가장 많은 매매 기회를 제공했다는 점에서 실전 투자에서의 활용도가 매우 높을 것으로 보입니다. 120 일선 터치 전략(MA_120_Touch_above_200) 역시 76.47%의 단기 승률과 94.12%의 장기 승률, 그리고 최대 400% 이상의 수익률을 기록하며 전략의 유효성을 입증했습니다.

이 정도면 위에서 설명했던 알고리즘을 통해 충분히 투자방식을 검토한 것 같습니다. 주식 투자를 할 때 사용했던 이동평균선, FTD, 거래량 등을 활용한 투자 방식을 코인을 투자할 때 적용해도 충분히 유의미한 결과를 뽑을 수 있을 거라는 확신이 들었습니다. 서론에서 이 기회를 통해 나의 투자 방식이 코인에도 적용가능한지 보고 싶다고 했었는데 실제로 유의미한 결과를 보니 실제 코인 투자를 할 때 충분히 활용을 해도 될 거 같습니다 물론 투자는 개인책임입니다.

7. 프로젝트와 수업 후기

이번 프로젝트를 통해 주식 시장에서 사용되는 기술적 분석 기법들이 암호화폐 시장에서도 유효하다는 점을 실증적으로 확인할 수 있었습니다. 특히 거래량 분석이 수익률 측면에서, 120일 이동평균선 터치와 FTD(First Touch Day) 전략이 승률 측면에서 유의미한 결과를 보여준 것은 매우 흥미로웠습니다.

이러한 경험은 앞으로의 투자 활동에 있어 매우 중요한 기반이 될 것으로 생각됩니다. 데이터 기반의 객관적인 분석과 체계적인 전략 수립의 중요성을 실감했으며, 이는 감정적 투자를 지양하고 합리적인 투자 결정을 내리는 데 큰 도움이 될 것입니다. 또한, 프로그래밍을 통한 자동화된 분석 시스템 구축의 가능성도 확인할 수 있었습니다.

데이터 분석 기초 수업이 아니었다면 언젠가는 해야지라고 미뤘을 주제인데 이 시간을 통해서 그리고 이 수업을 통해서 항상 지니고 있던 고민이 사라진 느낌입니다. 누구나 하나쯤은 분석해 보고 싶은 데이터가 있지만 귀찮다는 이유로 또는 어렵다는 이유로 미뤘을 것입니다. 그러나 이 수업은 궁금증 해결을 위한 한 발자국을 빅데이터 분석으로서써 해결하는 기회를 주는 것 같습니다.

참고문헌:

https://ko.wikipedia.org/wiki/%EC%9D%B4%EB%8F%99%ED%8F%89%EA%B7%A0%EC%84%A0 -위키백과 이동평균선

https://blog.naver.com/bon7373/223545915111 - 이동평균선 설정과 매매기법 정배열 역배열 투자법

https://m.blog.naver.com/bushwalk18/222101009160 - [책] 최고의 주식 최적의 타이밍-윌리엄 오닐

https://brunch.co.kr/@helix/9 - Follow though로 베팅의 구질 개선

https://docs.upbit.com/ - 업비트 개발자 센터

https://yang-wistory1009.tistory.com/71 - 파이썬 평균, 분산, 표준편차 구하기 mean(), var(), std() 함수 사용 - 공부하는 도비

https://www.tossbank.com/articles/bitcoinhalf - **비트코인 반감기가 뭐예요?**

https://eiec.kdi.re.kr/material/clickView.do?click_yymm=201512&cidx=2131 - 비트코인이란 무엇인가

부록- 전체코드

1. **코드 1- 승률, 수익률 출력 전체 코드**

```python
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
import requests
import time

# Upbit API 에서 캔들 데이터를 가져오는 함수
def fetch_candles_bulk(market, count, to=None):
    url = f"https://api.upbit.com/v1/candles/days"
    params = {"market": market, "count": count}
    if to:
        params["to"] = to
    response = requests.get(url, params=params)
    if response.status_code == 200:
        return response.json()
```

```
    else:
        print(f"API 요청 실패: 상태 코드 {response.status_code}, 메시지: {response.text}")
        return []

# 데이터를 가져와 DataFrame 으로 변환
def load_data_from_upbit(market, days):
    candles = []
    to_date = None

    while len(candles) < days:
        batch = fetch_candles_bulk(market, min(200, days - len(candles)), to=to_date)
        if not batch:
            break
        candles.extend(batch)
        to_date = batch[-1]["candle_date_time_utc"]
        time.sleep(0.5)  # API 요청 간격 조정

    candles.reverse()  # 시간 순서 정렬
    data = {
        "날짜": [datetime.strptime(candle["candle_date_time_utc"], "%Y-%m-%dT%H:%M:%S") for candle in candles],
        "종가": [candle["trade_price"] for candle in candles],
        "시가": [candle["opening_price"] for candle in candles],
        "저가": [candle["low_price"] for candle in candles],
        "거래량": [candle["candle_acc_trade_volume"] for candle in candles],
    }
    return pd.DataFrame(data)

# 이동평균 계산 함수
def calculate_moving_average(prices, window):
    series = pd.Series(prices)
    return series.shift(1).rolling(window=window).mean().tolist()

# 이동평균 및 거래량 평균 계산
def calculate_moving_averages(df):
    df['MA_20'] = calculate_moving_average(df['종가'], 20)
    df['MA_50'] = calculate_moving_average(df['종가'], 50)
    df['MA_120'] = calculate_moving_average(df['종가'], 120)
    df['MA_200'] = calculate_moving_average(df['종가'], 200)
    return df

def calculate_rolling_volume_average(df):
    rolling_volume = []
    current_avg = None

    for i in range(len(df)):
        today_volume = df.iloc[i]['거래량']

        if i < 365:
            # 365 일 되기 전까지는 현재까지의 평균 계산
            if current_avg is None:
                current_avg = today_volume
            else:
                current_avg = (current_avg * i + today_volume) / (i + 1)
        else:
            # 365 일 이후부터는 이전 평균에서 오래된 값 빼고 새 값 추가
            old_volume = df.iloc[i - 365]['거래량']
            current_avg = current_avg + (today_volume - old_volume) / 365

        rolling_volume.append(current_avg)

    df['Volume_Avg_365'] = rolling_volume
    return df

# 과거 최고가 계산 함수
def calculate_all_time_high(df):
```

```
    all_time_highs = []
    current_high = float('-inf')
    for price in df['종가']:
        current_high = max(current_high, price)
        all_time_highs.append(current_high)
    df['All_Time_High'] = all_time_highs
    return df

# 수익률 계산 함수
def calculate_returns(df, date_indices, periods):
    returns = {period: [] for period in periods}

    for date_idx in date_indices:
        start_price = df.iloc[date_idx]['종가']
        for period in periods:
            future_idx = date_idx + period
            if future_idx < len(df):
                future_price = df.iloc[future_idx]['종가']
                returns[period].append((future_price - start_price) / start_price * 100)
            else:
                returns[period].append(None)

    return returns

# 승률 계산 함수
def calculate_win_rate(returns):
    win_rates = {}
    for period, values in returns.items():
        valid_values = [value for value in values if value is not None]
        wins = [value for value in valid_values if value > 0]
        win_rate = len(wins) / len(valid_values) * 100 if valid_values else 0
        win_rates[period] = win_rate
    return win_rates

# 데이터 분석 함수
def analyze_conditions(df):
    # 이동평균선 및 거래량 계산
    df = calculate_moving_averages(df)
    df = calculate_rolling_volume_average(df)

    # 과거 최고가 계산
    df = calculate_all_time_high(df)

    # 결과 저장
    results = {
        'Golden_Cross_200': [],
        'FTD_Above_200': [],
        'High_Volume_2x_above_200': [],
        'MA_120_Touch_above_200': [],
        'High_Volume_3x': [],
        'Sequential_FTD_above_200': [],
        'Sequential_High_Volume_2x_above_200': [],
        'Sequential_MA_120_Touch_above_200': []
    }

    for i in range(200, len(df) - 180):
        row = df.iloc[i]

        # Golden Cross 200 조건 확인
        if row['MA_200'] is not None and df.iloc[i - 1]['종가'] < df.iloc[i - 1]['MA_200']
and min(row['시가'], row['종가']) <= row['MA_200'] <= max(row['시가'], row['종가']):
            results['Golden_Cross_200'].append(i)

        # FTD Above 200 조건 확인
        if row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['종가'] <
row['All_Time_High'] * 0.7:
```

```
            if row['종가'] >= row['시가'] * 1.0572:
                valid = True
                for j in range(1, 4):
                    future_row = df.iloc[i + j]
                    if future_row['저가'] < row['저가']:
                        valid = False
                        break

                if valid:
                    for k in range(4, 8):
                        future_row = df.iloc[i + k]
                        if future_row['종가'] >= row['시가'] * 1.0572 and future_row['저가'] >= row['저가']:
                            volume_increasing = all(
                                df.iloc[i + n]['거래량'] > df.iloc[i + n - 1]['거래량']
                                for n in range(4, k + 1)
                            )
                            if volume_increasing:
                                results['FTD_Above_200'].append(i)
                                break

        # High Volume 2x Above 200 조건 확인
        if row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['거래량'] > row['Volume_Avg_365'] * 2:
            results['High_Volume_2x_above_200'].append(i)

        # MA 120 Touch Above 200 조건 확인 (주가가 200일선 위이고 종가와 시가 사이에 120일선 위치)
        if row['MA_120'] is not None and row['MA_200'] is not None and row['종가'] > row['MA_200'] and row['MA_120'] >= min(row['종가'], row['시가']) and row['MA_120'] <= max(row['종가'], row['시가']):
            results['MA_120_Touch_above_200'].append(i)

        # High Volume 3x 조건 확인 (200일선 조건 없음)
        if row['거래량'] > row['Volume_Avg_365'] * 3:
            results['High_Volume_3x'].append(i)

        # Sequential FTD Above 200 조건 확인 (정배열 조건 수정)
        if (
            row['MA_20'] is not None and row['MA_50'] is not None and
            row['MA_120'] is not None and row['MA_200'] is not None and
            row['MA_20'] > row['MA_50'] > row['MA_120'] > row['MA_200'] and  # 정배열 수정
            row['종가'] > row['MA_200'] and row['종가'] < row['All_Time_High'] * 0.7 and
            row['종가'] >= row['시가'] * 1.0572
        ):
            valid = True
            for j in range(1, 4):
                future_row = df.iloc[i + j]
                if future_row['저가'] < row['저가']:
                    valid = False
                    break

            if valid:
                for k in range(4, 8):
                    future_row = df.iloc[i + k]
                    if future_row['종가'] >= row['시가'] * 1.0572 and future_row['저가'] >= row['저가']:
                        volume_increasing = all(
                            df.iloc[i + n]['거래량'] > df.iloc[i + n - 1]['거래량']
                            for n in range(4, k + 1)
                        )
                        if volume_increasing:
                            results['Sequential_FTD_above_200'].append(i)
                            break

```

```
        # Sequential High Volume 2x Above 200 조건 확인
        if (
            row['MA_20'] is not None and row['MA_50'] is not None and
            row['MA_120'] is not None and row['MA_200'] is not None and
            row['MA_20'] > row['MA_50'] > row['MA_120'] > row['MA_200'] and  # 정배열 수정
            row['거래량'] > row['Volume_Avg_365'] * 2
        ):
            results['Sequential_High_Volume_2x_above_200'].append(i)

        # Sequential MA 120 Touch Above 200 조건 확인
        if (
            row['MA_20'] is not None and row['MA_50'] is not None and
            row['MA_120'] is not None and row['MA_200'] is not None and
            row['MA_20'] > row['MA_50'] > row['MA_120'] > row['MA_200'] and  # 정배열 수정
            row['MA_120'] >= min(row['종가'], row['시가']) and
            row['MA_120'] <= max(row['종가'], row['시가'])
        ):
            results['Sequential_MA_120_Touch_above_200'].append(i)

    return results

# 결과 출력 및 시각화
def print_and_visualize_results(df, results):
    periods = [30, 90, 180]  # 1 개월, 3 개월, 6 개월
    all_returns = {}

    for condition, indices in results.items():
        returns = calculate_returns(df, indices, periods)
        all_returns[condition] = returns
        win_rates = calculate_win_rate(returns)

        print(f"{condition} 승률:")
        print(f"1M: {win_rates[30]:.2f}%, 3M: {win_rates[90]:.2f}%, 6M:
{win_rates[180]:.2f}%")
        print(f"{condition} 날짜:")
        print([df.iloc[idx]['날짜'].strftime('%Y-%m-%d') for idx in indices])

    # 시각화
    for period in periods:
        plt.figure(figsize=(12, 8))
        for condition, returns in all_returns.items():
            values = returns[period]
            plt.scatter([condition] * len(values), values, label=condition, alpha=0.7)
        plt.axhline(0, color="red", linestyle="--")
        plt.title(f"{period // 30}M Returns Comparison")
        plt.ylabel("Return (%)")
        plt.xlabel("Conditions")
        plt.legend()
        plt.show()

# 실행
market = "KRW-BTC"
data = load_data_from_upbit(market, 2000)

if data is not None:
    analysis_results = analyze_conditions(data)
    print_and_visualize_results(data, analysis_results)

```

2. 코드 2 - 변동성 계산<수정후>

```python
import pandas as pd
import numpy as np

# 파일 경로
bitcoin_file_path = 'bitcoin_price_data.csv'
sp500_file_path = 'S&P.csv'

# 데이터 불러오기
bitcoin_data = pd.read_csv(bitcoin_file_path)
sp500_data = pd.read_csv(sp500_file_path)

# 날짜 컬럼을 datetime 형식으로 변환
bitcoin_data['날짜'] = pd.to_datetime(bitcoin_data['날짜'])
sp500_data['날짜'] = pd.to_datetime(sp500_data['날짜'])

# 종가 컬럼을 숫자로 변환 (콤마 제거 필요 시 적용)
bitcoin_data['종가'] = pd.to_numeric(bitcoin_data['종가'], errors='coerce')
sp500_data['종가'] = pd.to_numeric(sp500_data['종가'].str.replace(',', ''), errors='coerce')

# 날짜를 기준으로 정렬
bitcoin_data = bitcoin_data.sort_values('날짜').reset_index(drop=True)
sp500_data = sp500_data.sort_values('날짜').reset_index(drop=True)

# 일일 수익률 계산
bitcoin_data['daily_return'] = bitcoin_data['종가'].pct_change()
sp500_data['daily_return'] = sp500_data['종가'].pct_change()

# NaN 제거
bitcoin_data = bitcoin_data.dropna(subset=['daily_return'])
sp500_data = sp500_data.dropna(subset=['daily_return'])

# 연도별 변동성 계산 함수
def calculate_annualized_volatility_by_year(data, trading_days):
    data['year'] = data['날짜'].dt.year  # 연도별 그룹화
    annual_volatility = data.groupby('year')['daily_return'].std() * np.sqrt(trading_days)
    return annual_volatility

# 비트코인: 365 일 기준
btc_annual_volatility = calculate_annualized_volatility_by_year(bitcoin_data, 365)

# S&P 500: 252 일 기준
sp500_annual_volatility = calculate_annualized_volatility_by_year(sp500_data, 252)

# 결과 출력
print("비트코인의 연도별 연간 변동성:")
print(btc_annual_volatility)
print("\nS&P 500 의 연도별 연간 변동성:")
print(sp500_annual_volatility)

# 전체 평균 연간 변동성 계산
btc_avg_volatility = btc_annual_volatility.mean()
sp500_avg_volatility = sp500_annual_volatility.mean()

print(f"\n 비트코인의 평균 연간 변동성 (6 년 기준): {btc_avg_volatility:.2%}")
print(f"S&P 500 의 평균 연간 변동성 (6 년 기준): {sp500_avg_volatility:.2%}")


```

3. 코드 3 – 변동성 계산 <수정전>

```
 1. import pandas as pd
 2. import numpy as np
 3.
 4. # 파일 경로
 5. bitcoin_file_path = 'bitcoin_price_data.csv'
 6. sp500_file_path = 'S&P.csv'
 7.
 8. # 데이터 불러오기
 9. bitcoin_data = pd.read_csv(bitcoin_file_path)
10. sp500_data = pd.read_csv(sp500_file_path)
11.
12. # 변동 % 컬럼의 % 기호 제거 및 숫자로 변환
13. bitcoin_data['변동 %'] = pd.to_numeric(bitcoin_data['변동 %'].str.replace('%', ''),
errors='coerce') / 100
14. sp500_data['변동 %'] = pd.to_numeric(sp500_data['변동 %'].str.replace('%', ''),
errors='coerce') / 100
15.
16. # 변동성 계산 함수
17. def calculate_annualized_volatility(daily_returns):
18.     return np.std(daily_returns) * np.sqrt(252)  # 연간화
19.
20. # 비트코인과 S&P 500 의 변동성 계산 (변동 %를 바로 활용)
21. btc_volatility = calculate_annualized_volatility(bitcoin_data['변동 %'])
22. sp500_volatility = calculate_annualized_volatility(sp500_data['변동 %'])
23.
24. # 결과 출력
25. print(f"비트코인의 연간 변동성: {btc_volatility:.2%}")
26. print(f"S&P 500 의 연간 변동성: {sp500_volatility:.2%}")
27.
```