시장 심리의 온도계

Buffett Indicator & Fear & Greed Index 완전 정복

강의 전체 목표 및 로드맵

전체 강의 목표

- 시장 심리를 이해하고, 투자 의사결정에 적용할 수 있다.
- Buffett Indicator 와 Fear & Greed Index 를 개념부터 수식, 실전까지 깊이 있게 습득한다.
- Colab + DuckDB + Streamlit 으로 데이터 수집 → 저장 → 분석 → 시각화 → 대시보드를 직접 구현한다.

학습 기대 효과

- 시장의 과열 / 침체 국면을 정확히 판단하는 능력 배양
- 데이터 기반 사고 훈련: 감정에 흔들리지 않는 투자 판단
- 실제 투자 전략 및 리서치 리포트에 바로 적용 가능한 심리 지표 분석 능력 확보

강의 전체 목표 및 로드맵

강의 로드맵

- 오리엔테이션: 투자 심리의 중요성 이해
- Buffett Indicator 심층 분석
- Fear & Greed Index 심층 분석
- 두 지표 통합 분석 & 실전 활용 전략
- Colab + DuckDB + Streamlit 실습
- 완성형 대시보드 구현
- 마무리 Q&A 및 복습

시장 심리의 온도계

1) 시장은 왜 '심리'로 움직일까?

시장 참여자는 모두 인간입니다. 경제적 합리성을 추구하지만, 탐욕, 공포, 희망, 좌절, 확증편향, 군중 심리 등 다양한 감정과 심리적 요인이 투자 결정에 깊이 작용합니다.

🔍 투자 심리에 시장이 영향을 받는 이유

- **정보의 불확실성**
  - 시장은 항상 미래를 예상합니다.
  - "지금 가격"은 미래의 기대와 공포가 반영된 결과물.
  - 아무리 데이터가 풍부해도 100% 확신할 수 없으니 심리가 개입됩니다.

- **군중심리와 사회적 증폭**
  - "다른 사람들도 사니까 나도 산다"
  - "모두가 불안해하니 나도 불안하다"
  - 뉴스, 소셜미디어, 애널리스트 리포트 등 외부 신호가 군중심리를 증폭

- **인지 편향과 감정**
  - 확증편향: 보고 싶은 정보만 찾는다
  - 손실회피 성향: 손실이 이익보다 두 배 이상 강한 심리적 충격
  - FOMO (Fear of Missing Out): 놓칠까 봐 조급해지는 심리

시장 심리의 온도계

🧩 대표적인 사례 (확장)

1) 닷컴버블 (2000년대 초)

- "인터넷이 세상을 바꾼다"는 기대감에 수많은 IT·인터넷 기업들이 실적 없이도 높은 평가를 받음
- "미래에 엄청난 성장을 할 것이다"는 희망이 탐욕으로 연결
- 결과: 거품 붕괴, 나스닥 5000 → 1100포인트까지 폭락
- 📌 교훈: 지나친 기대감이 시장 과열을 불러일으킴

2) 리먼 브라더스 파산 (2008년 금융위기)

- 부동산 가격 상승기 동안 탐욕적 투자자들이 레버리지를 늘리며 매수
- 부채 기반의 투자 구조가 붕괴하면서 시장이 공포에 빠짐
- 결과: 글로벌 금융 시스템 붕괴 위기
- 📌 교훈: 과도한 신용과 공포 심리가 결합하면 시장 붕괴

3) 코로나19 팬데믹 초기 (2020년 3월)

- 바이러스 확산 → 글로벌 경제 셧다운 → 공포가 극단으로 치닫음
- 주식, 원자재, 채권 모두 급락
- 결과: 정부와 중앙은행의 유례없는 유동성 공급으로 V자 반등
- 📌 교훈: 공포의 극단이 시장의 저점일 수 있음

4) 2021년 밈 스톡 (GameStop 사태)

- 개인 투자자들이 Reddit 커뮤니티를 통해 대규모 매수
- 군중심리 + FOMO + 숏스퀴즈로 주가 급등
- 결과: 며칠 만에 수백 % 상승과 하락
- 📌 교훈: 군중심리가 정보보다 앞설 때 시장은 비이성적으로 움직인다

시장 심리의 온도계

💡 심리가 시장에 미치는 메커니즘 (단계별)

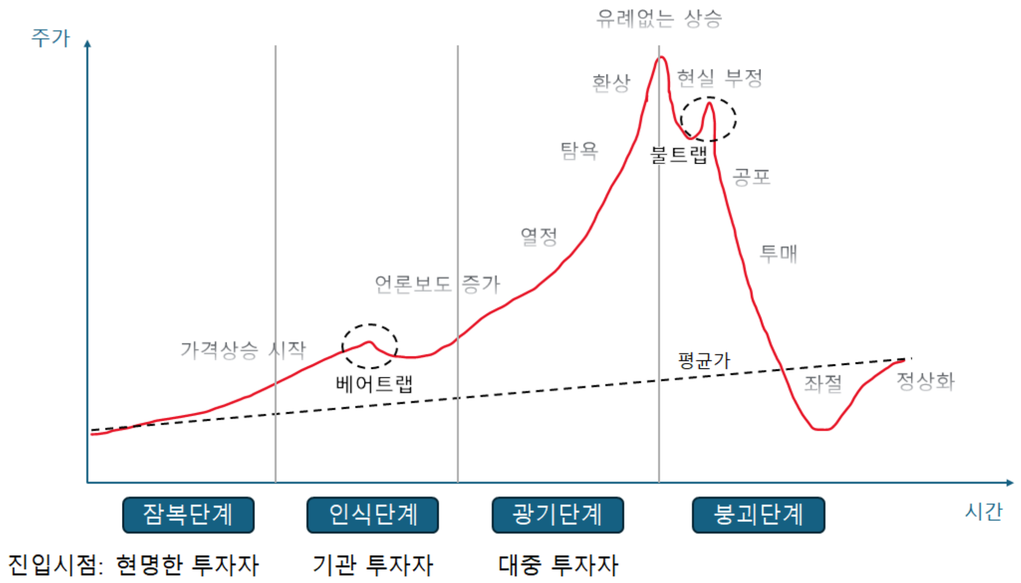
- 초기 낙관 / 비관
  - 좋은 뉴스나 악재가 퍼지기 시작
  - 투자자 기대감 / 우려 증폭

- 과잉 확신 / 공포 심화
  - 상승장: "더 오를 거야!" (탐욕)
  - 하락장: "더 떨어질 거야…" (공포)

- 군중 심리 극단화
  - 소셜 미디어, 뉴스, 전문가 발언이 투자자 심리 극단화

- 시장 가격에 반영
  - 실제 기업 가치와 괴리 발생
  - 버블 형성 / 패닉셀링

- 결국 Fundamentals (기초 체력) 로 복귀
  - 시간의 흐름 속에서 시장은 실적, 가치로 회귀
  - 그러나 심리가 방향성에는 강하게 작용



시장 심리의 온도계

심리적 요인	시장 반응	대표 사례
탐욕	과열 (버블)	닷컴버블, 부동산 버블
공포	급락 / 패닉셀	2008 금융위기, 코로나19 초기
군중심리	비이성적 가격 형성	밈 스톡 (GameStop)
확증편향	편향된 정보 해석	상승장 과열 시 일반적
손실회피 성향	손절매 가속화	패닉장
FOMO	늦은 추격 매수	급등장 후 추격 매수

왜 시장의 심리를 이해해야 할까?

시장의 본질적 특징

시장은 인간 심리가 지배하는 생명체와 같다.

- 시장 가격은 경제 데이터 + 투자자의 심리적 반응의 결과물
- 수많은 투자자의 기대, 두려움, 욕망이 모여 시장을 움직임
- 시장은 숫자처럼 보이지만, 사실상 심리의 반영

이론적 배경: 효율적 시장 가설 (EMH)

“모든 정보는 가격에 반영되어 있다.”
- 투자자는 합리적이고, 시장은 항상 효율적이라는 가설
- 경제 데이터와 기업 실적이 투자 판단의 핵심이 된다는 이론

하지만 현실에서는?
- 투자자들은 합리적인 듯 보이지만, 실제로는 감정의 영향을 강하게 받음
- 예측할 수 없는 사건이나 뉴스가 투자 심리에 불씨를 지피며 시장에 파급

구분	이론 (EMH)	현실 (행동 재무학)
시장 작동 방식	효율적, 모든 정보 반영	비효율적, 감정적 요인 작용
투자자 행동	합리적	감정적, 비이성적
가격 형성	데이터 기반	데이터 + 심리 반응

데이터 기반 투자자로 거듭나기

감정적 투자자 vs 데이터 기반 투자자

항목	감정적 투자자	데이터 기반 투자자
투자 결정	뉴스와 루머에 반응	데이터와 근거에 기반
심리 상태	공포와 탐욕에 흔들림	감정을 거리 두고 객관화
투자 태도	단기적 충동	장기적 계획 수립
사용 도구	감	데이터 지표, 분석 도구

데이터 기반 투자자로 거듭나기

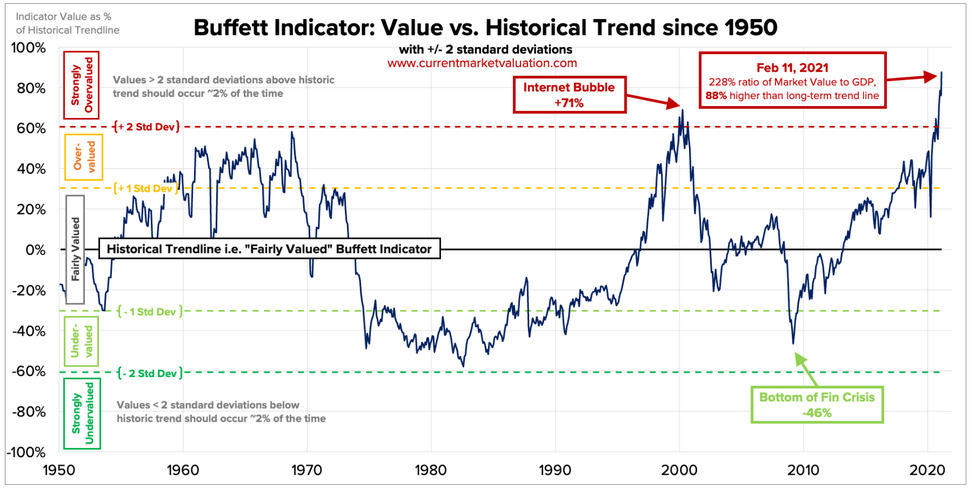
시장을 '감'이 아닌 '데이터'로 진단합니다

- 시장의 과열과 침체를 단순한 느낌이 아니라 다양한 지표를 통해 입체적으로 파악
- 한 가지 지표가 아닌, 복합적 분석이 중요

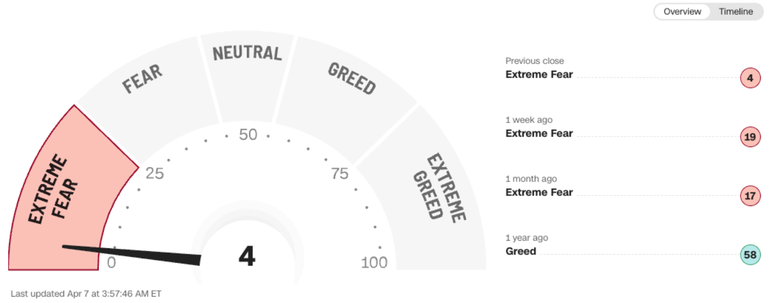
데이터 기반 투자자로 거듭나기

대표 심리지표

- Buffett Indicator
  - 전체 시가총액 ÷ GDP
  - 시장 전체가 경제의 실물 규모 대비 어느 정도까지 확장되었는지 보여주는 지표.
  - 높은 값: 시장 과열 가능성 (거품 경고)
  - 낮은 값: 시장 저평가 가능성 (기회 요인)



- Fear & Greed Index (CNN)
  - 7가지 지표 조합
    - 주식 가격 모멘텀 (S&P 500 vs 125일 이동 평균)
    - 주식 가격 강도 (52주 고가/저가 종목 비율)
    - 옵션 시장 변동성 (Put/Call Ratio)
    - 시장 변동성 (VIX 지수)
    - 안전자산 수요 (채권과 주식 수익률 스프레드)
    - 안전자산 수요 (채권 가격)
    - 시장 참여도 (거래량, 상승 하락 비율)
  - "Extreme Fear" (극단적 공포) ~ "Extreme Greed" (극단적 탐욕) 으로 평가



데이터 기반 투자자로 거듭나기

💡심리 + 수급 보조 지표들 (확장)

VIX 지수 (공포 지수)
- S&P500 옵션의 변동성을 기반으로 시장의 불확실성을 측정
- VIX 상승 → 시장 불안 심리 증가
- VIX 하락 → 시장 안정

Put/Call Ratio (옵션 매매 심리 지표)
- 투자자들이 풋옵션(하락 베팅) vs 콜옵션(상승 베팅)을 얼마나 매수하는지 비율
- 높을수록 시장에 대한 비관적 전망 증가

Advance/Decline Line (시장 내부 강도)
- 상승 종목 수와 하락 종목 수의 차이
- 시장의 상승/하락이 소수 종목에 의해 결정되는지, 광범위한지 판단

Margin Debt (신용융자 잔고)
- 투자자들이 증권사를 통해 빌린 자금 총액
- 급격한 증가 → 과도한 레버리지 → 거품 가능성

데이터 기반 투자자로 거듭나기

🌏 거시 경제 지표

10년 - 2년 국채 금리차 (Yield Curve)
- 장단기 금리 차이가 역전되면 경기 침체 신호
- 정상 구조: 장기 금리가 단기보다 높음
- 역전 구조: 경기 침체 우려

PMI (구매관리자지수)
- 50 이상: 경기 확장
- 50 이하: 경기 위축

소비자심리지수 (Consumer Confidence Index)
- 소비자의 경기 전망 심리를 수치화

🧩 시장 밸류에이션 지표

Shiller CAPE Ratio (사이클 조정 PER)
- 인플레이션 조정된 10년 평균 PER
- 장기적 관점에서 시장의 고평가 / 저평가 판단

Price-to-Book Ratio (PBR)
- 주가 / 순자산
- 시장이 기업 자산을 얼마나 프리미엄 붙여 평가하는지 측정

데이터 기반 투자자로 거듭나기

분류	지표명	설명
심리지표	Fear & Greed Index	투자자 심리 종합 측정
	VIX 지수 (공포 지수)	시장 변동성 측정
	Put/Call Ratio	옵션 투자자 심리
수급/시장 내부 흐름 지표	Advance/Decline Line	시장 내부 강도 분석
	Margin Debt	레버리지 수준
거시경제 지표	금리차 (Yield Curve)	경기 침체 선행 신호
	PMI	제조업/서비스업 경기 판단
	소비자심리지수 (CCI)	소비자 경기 전망
밸류에이션 지표	Buffett Indicator	시장 가치 평가
	Shiller CAPE Ratio	장기 PER 평가
	Price-to-Book Ratio (PBR)	자산 대비 시장 평가

Buffett Indicator — 시장 가치의 나침반

Buffett Indicator 란?

"시장의 밸류에이션을 단 하나의 숫자로!”

워런 버핏이 시장 전체 밸류에이션 판단에 사용하는 지표

- **공식**
  - Buffett Indicator = 전체 시가총액 ÷ GDP
- **의미**
  - 전체 주식 시장이 경제 규모 대비 얼마나 과열 혹은 저평가되어 있는지를 나타냄
  - 투자자들의 기대감이 반영된 시가총액과 실물경제의 크기(GDP)를 비교
- **기원**
  - 워런 버핏이 '가장 간단하고 유용한 시장 가치 평가 도구'라며 소개

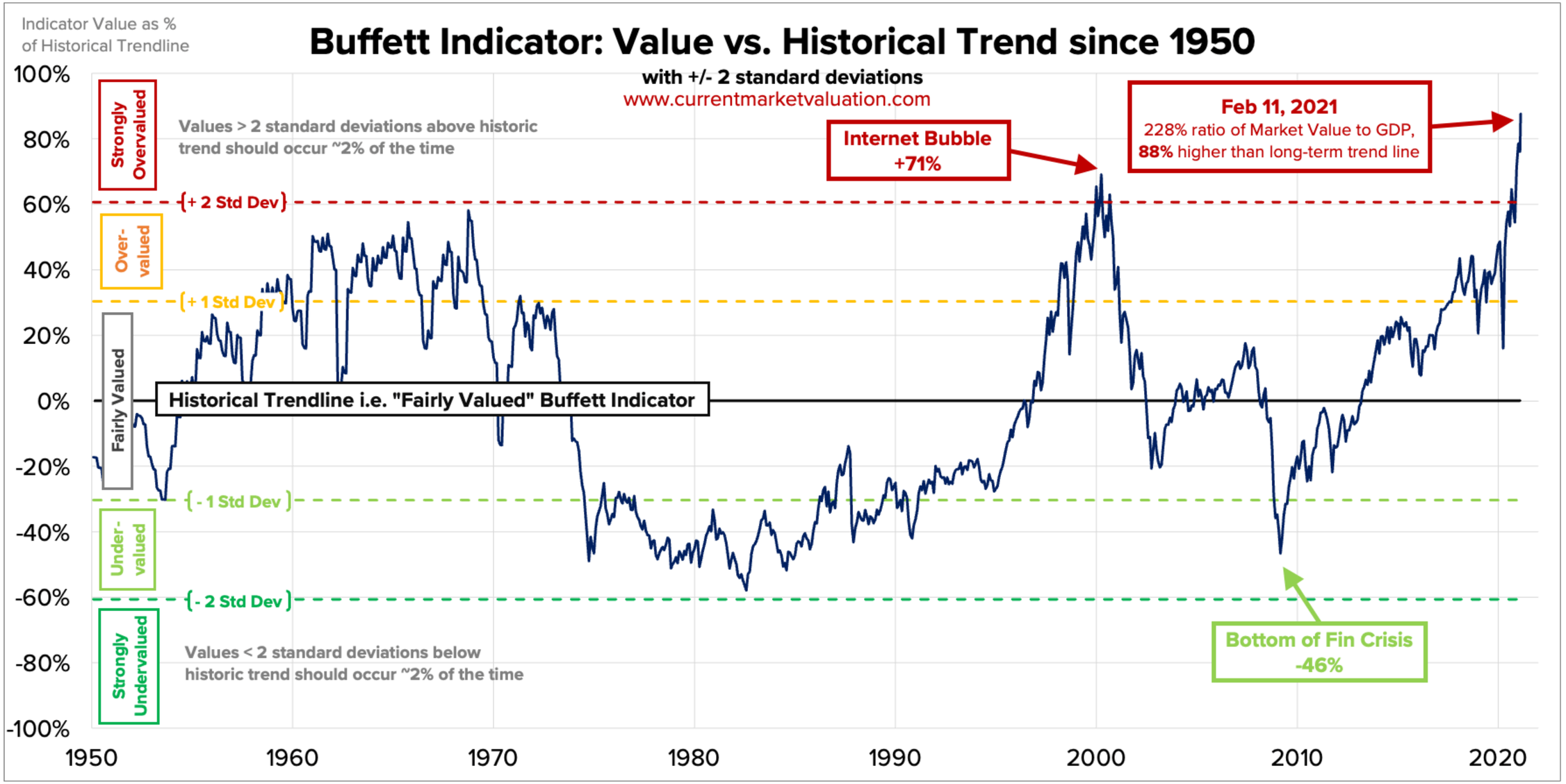
Buffett Indicator — 시장 가치의 나침반

Buffett Indicator 기준선

지표 수치	해석
80% 이하	저평가 (시장이 싸다)
100% 내외	적정가 (합리적 수준)
120% 이상	과열 (거품 가능성)

- 참고: 기준선은 시장 및 국가마다 다를 수 있음
- 미국 시장에서는 2020~2021년 동안 200% 돌파 (역사적 고점)

Buffett Indicator — 시장 가치의 나침반



Buffett Indicator — 시장 가치의 나침반

Buffett Indicator 의 장점과 한계

구분	장점	한계
직관성	한눈에 시장 과열/저평가 파악	기준선이 국가별로 다름
데이터	전체 시장과 GDP 비교	GDP 데이터는 분기마다 업데이트
활용도	장기적 투자 시점 판단	단기 시장 변동성 반영 어려움

Buffett Indicator 구현 실습

📖 실습 목표

- 미국 시장 버핏 인디케이터 실시간 계산
- Colab 에서 데이터 수집 → DuckDB 저장 → 분석 → 시각화
- Streamlit 으로 대시보드 구현

✅ 이 실습을 마치면:

- Buffett Indicator 를 계산하고, 시장의 과열/침체 상황을 수치로 판단할 수 있습니다.
- Colab 환경에서 데이터 저장소로 DuckDB 를 활용하는 방법을 익힙니다.
- 시각화 및 대시보드 구현까지, 데이터 기반 시장 분석 도구를 완성합니다!

Buffett Indicator 구현 실습

라이브러리 설치

```
# ------------------------
# 1. 필요한 라이브러리 설치
# ------------------------


print("🚀 라이브러리 설치 시작!")
!pip install yfinance pandas matplotlib duckdb streamlit fredapi pyngrok --quiet
print("✅ 라이브러리 설치 완료!")
```

Buffett Indicator 구현 실습

✅ **FRED API란?**

- FRED (Federal Reserve Economic Data) 는 미국 세인트루이스 연준(Federal Reserve Bank of St. Louis) 에서 제공하는
- 세계적인 경제 데이터 플랫폼입니다.
- 미국 GDP, 물가상승률, 고용지표 등 다양한 경제지표를 실시간으로 API를 통해 사용할 수 있습니다.
- https://fred.stlouisfed.org/
- ✅ 강의 포인트: 우리는 이번 실습에서 미국 명목 GDP 데이터를 가져옵니다.

FRED API 키 설정 (FRED 홈페이지에서 무료 발급)

1.FRED 홈페이지 접속: https://fred.stlouisfed.org/
2.'My Account' 클릭 후 회원가입 (무료)
3.'API Keys' 메뉴 선택 후 'Request API Key' 클릭
4.발급된 API 키를 복사하여 아래에 붙여넣기

❗ 주의: API 키는 개인 키이므로 외부에 노출되지 않도록 관리하세요.

Buffett Indicator 구현 실습

데이터 수집

```
# ------------------------
# 2. 데이터 수집
# ------------------------

print("\n🚀 데이터 수집 시작!")

import pandas as pd
import yfinance as yf
from fredapi import Fred

FRED_API_KEY = '여기에 본인 API 키 입력'  # 여기에 본인 API 키 입력
fred = Fred(api_key=FRED_API_KEY)

print("\n📥 미국 GDP 데이터 수집 중 (FRED)...")
gdp_series = fred.get_series('GDP')
gdp_data = gdp_series.reset_index()
gdp_data.columns = ['Date', 'GDP']
gdp_data['Date'] = pd.to_datetime(gdp_data['Date'])
gdp_data = gdp_data.sort_values('Date')
print("✅ FRED GDP 데이터 수집 완료! 총 데이터 포인트:", len(gdp_data))
print(gdp_data.tail())

print("\n📥 미국 전체 시가총액 데이터 수집 중 (Yahoo Finance)...")
wilshire = yf.Ticker("^W5000")
market_cap_data = wilshire.history(period="max")
market_cap_data = market_cap_data[['Close']].rename(columns={'Close': 'Market_Cap'})
market_cap_data['Date'] = market_cap_data.index
market_cap_data.reset_index(drop=True, inplace=True)
print("✅ Yahoo Finance 시가총액 데이터 수집 완료! 총 데이터 포인트:", len(market_cap_data))
print(market_cap_data.tail())
```

Buffett Indicator 구현 실습

수집된 데이터 저장

```
# -------------------------
# 3. DuckDB 저장
# -------------------------
print("\n🚀 DuckDB 데이터베이스 저장 시작!")

import duckdb

con = duckdb.connect(database='buffett_indicator.duckdb', read_only=False)

print("📦 GDP 데이터 DuckDB 저장 중...")
con.execute("CREATE TABLE IF NOT EXISTS gdp_data AS SELECT * FROM gdp_data")

print("📦 시가총액 데이터 DuckDB 저장 중...")
con.execute("CREATE TABLE IF NOT EXISTS market_cap_data AS SELECT * FROM market_cap_data")

print("✅ DuckDB 저장 완료! 저장된 데이터 미리보기:")
print(con.execute("SELECT * FROM gdp_data LIMIT 5").fetchdf())
print(con.execute("SELECT * FROM market_cap_data LIMIT 5").fetchdf())
```

Buffett Indicator 구현 실습

데이터 분석&저장

```
print("\n🚀 DuckDB 에서 데이터 불러오기 및 Buffett Indicator 계산 시작!")

# DuckDB 에서 데이터 읽기
gdp_df = con.execute("SELECT * FROM gdp_data").fetchdf()
market_cap_df = con.execute("SELECT * FROM market_cap_data").fetchdf()

# 데이터 전처리
print("📅 데이터 전처리 중...")
gdp_df['Date'] = pd.to_datetime(gdp_df['Date'])
market_cap_df['Date'] = pd.to_datetime(market_cap_df['Date'])

# 🩹 타임존 제거 (market_cap_df)
market_cap_df['Date'] = market_cap_df['Date'].dt.tz_localize(None)

# 날짜 리샘플링 및 보간
gdp_df = gdp_df.set_index('Date').resample('D').ffill().reset_index()
market_cap_df = market_cap_df.set_index('Date').resample('D').ffill().reset_index()

# 병합
print("🔗 데이터 병합 중...")
buffett_df = pd.merge(market_cap_df, gdp_df, on='Date', how='inner')

# Buffett Indicator 계산
buffett_df['Buffett_Indicator'] = (buffett_df['Market_Cap'] / buffett_df['GDP']) * 100

print("✅ Buffett Indicator 계산 완료! 결과 미리보기:")
print(buffett_df.tail())

print("📦 Buffett Indicator 결과 DuckDB 저장 중...")
con.execute("CREATE OR REPLACE TABLE buffett_indicator_data AS SELECT * FROM buffett_df")
print("✅ 저장 완료!")

# DuckDB 연결 종료
con.close()
print("🔒 DuckDB 연결 종료 완료!")
```

Buffett Indicator 구현 실습

시각화

```
# --------------------------
# 4. 시각화 (Matplotlib)
# --------------------------

print("\n🚀 시각화 시작!")

import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.plot(buffett_df['Date'], buffett_df['Buffett_Indicator'], label='Buffett Indicator (%)', color='blue',linewidth=0.5)
plt.axhline(y=100, color='green', linestyle='--', label='Fair Value (100%)')
plt.axhline(y=120, color='red', linestyle='--', label='Overheated (120%)')
plt.fill_between(buffett_df['Date'], 100, 120, color='green', alpha=0.1)
plt.fill_between(buffett_df['Date'], 120, buffett_df['Buffett_Indicator'].max(), color='red', alpha=0.1)
plt.title('Buffett Indicator Over Time')
plt.xlabel('Date')
plt.ylabel('Indicator (%)')
plt.legend()
plt.grid(True)
plt.show()

print("✅ 시각화 완료! Buffett Indicator 의 변화를 시각적으로 확인했습니다.")
```

Buffett Indicator 구현 실습

대시보드

```
# ---------------------------
# 5. Streamlit 대시보드 코드 저장
# ---------------------------

print("\n🚀 Streamlit 대시보드 코드 저장 시작!")

streamlit_dashboard_code = """
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import duckdb

st.title(\"\U0001F4C8 Buffett Indicator Dashboard\")

# DuckDB 에서 데이터 불러오기
con = duckdb.connect(database='buffett_indicator.duckdb', read_only=True)
buffett_df = con.execute('SELECT * FROM buffett_indicator_data').fetchdf()

st.line_chart(buffett_df.set_index('Date')['Buffett_Indicator'])

st.markdown(\"\"\"
- ✅ **100% 이하:** 시장이 저평가 상태일 수 있음
- ⚠️ **100% ~ 120%:** 합리적 수준
- 🔥 **120% 이상:** 시장 과열 구간
\"\"\")

fig, ax = plt.subplots()
ax.plot(buffett_df['Date'], buffett_df['Buffett_Indicator'], label='Buffett Indicator (%)')
ax.axhline(y=100, color='gray', linestyle='--', label='Fair Value (100%)')
ax.axhline(y=120, color='red', linestyle='--', label='Overheated (120%)')
ax.set_xlabel('Date')
ax.set_ylabel('Indicator (%)')
ax.legend()
st.pyplot(fig)
"""

with open("app.py", "w") as file:
    file.write(streamlit_dashboard_code)

print("✅ Streamlit 대시보드 코드가 'buffett_dashboard.py' 로 저장되었습니다!")
```

Buffett Indicator 구현 실습

서비스 실행

```
# ngrok으로 공개 URL 생성 (Authtoken 필요)
from pyngrok import ngrok

# 여기에 발급받은 ngrok authtoken 입력하세요!
NGROK_AUTH_TOKEN = "여기에 발급받은 ngrok authtoken 입력하세요!"
ngrok.set_auth_token(NGROK_AUTH_TOKEN)

print("[INFO] ngrok 터널 생성 중...")
public_url = ngrok.connect(addr="8501", proto="http")
print(f"[INFO] Streamlit 앱 Public URL: {public_url}")

# Streamlit 앱 실행
!streamlit run app.py --server.port 8501 --server.enableCORS false

print("[INFO] Streamlit 웹 서비스가 실행되었습니다. 위의 Public URL을 클릭하여 접속하세요!")
```

Buffett Indicator 구현 실습

서비스 실행

```
# ngrok으로 공개 URL 생성 (Authtoken 필요)
from pyngrok import ngrok

# 여기에 발급받은 ngrok authtoken 입력하세요!
NGROK_AUTH_TOKEN = "여기에 발급받은 ngrok authtoken 입력하세요!"
ngrok.set_auth_token(NGROK_AUTH_TOKEN)

print("[INFO] ngrok 터널 생성 중...")
public_url = ngrok.connect(addr="8501", proto="http")
print(f"[INFO] Streamlit 앱 Public URL: {public_url}")

# Streamlit 앱 실행
!streamlit run app.py --server.port 8501 --server.enableCORS false

print("[INFO] Streamlit 웹 서비스가 실행되었습니다. 위의 Public URL을 클릭하여 접속하세요!")
```

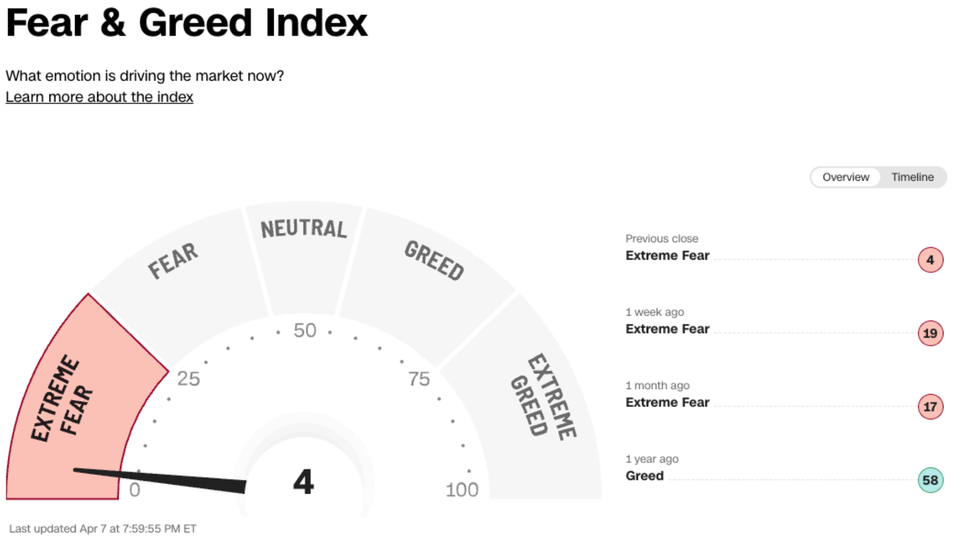
Fear & Greed Index

Fear & Greed Index (공포탐욕지수) 란?

"투자자들의 감정이 시장에 어떤 영향을 미치고 있는지를 수치로 나타내는 심리지표"

CNN Business 가 개발한
주식 시장에서 탐욕이 우세한지, 공포가 우세한지 시계열로 보여주는 도구

- 🧠 투자 심리의 지표화: 막연했던 시장 심리를 수치로 가시화
- 🕰️ 시장의 과열 / 침체 타이밍 포착: 고점, 저점 근접 시점 참고
- 🧩 투자 전략 수립: 시장 심리에 따라 매매 전략 결정



Fear & Greed Index

왜 공포와 탐욕인가?

- 투자자는 항상 심리적 압력 속에서 결정합니다.
- 공포: 불확실성이 커지면 매도를 선택.
- 탐욕: 상승장이 지속되면 더 높은 수익을 기대하며 과도한 매수.
- 워런 버핏: "Be fearful when others are greedy and greedy when others are fearful."

사례

시점	F&G Index	시장 상황
2020년 3월	12 (Extreme Fear)	코로나 쇼크, 시장 바닥
2021년 11월	84 (Extreme Greed)	유동성 장세, S&P500 최고점
2022년 6월	25 (Fear)	금리 인상 불안, 시장 조정

Fear & Greed Index

Fear & Greed Index 구성 요소

CNN Money 의 Fear & Greed Index 는 7가지 지표로 구성됩니다:

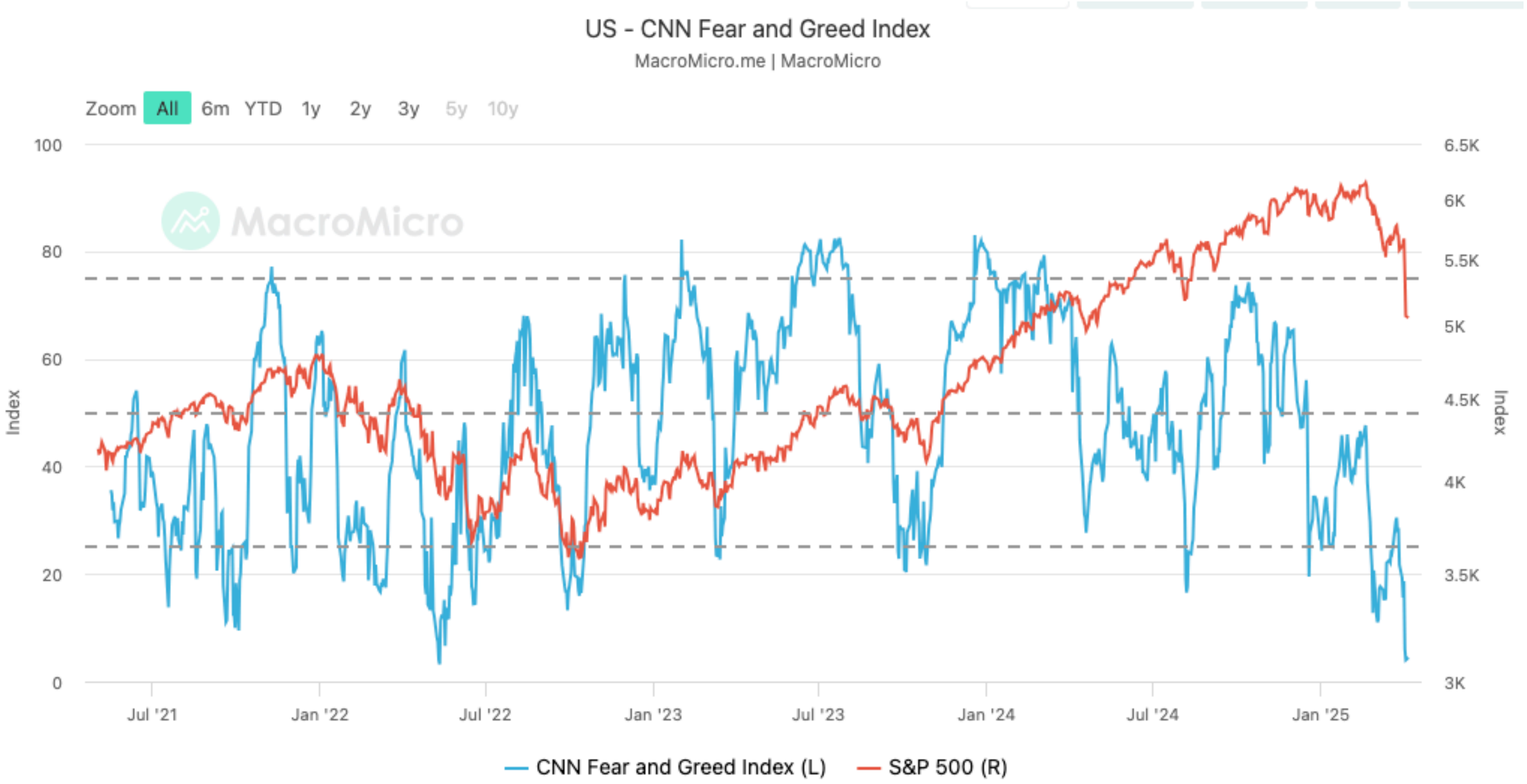
구분	세부 지표	설명
📈 시장 모멘텀	S&P500 vs 125일 평균	현재 주가 수준
📊 시장 강도	52주 고가 vs 저가 종목 비율	시장 내 강세/약세 비율
🧩 옵션시장 심리	Put/Call Ratio	투자자들의 베팅 방향
⚠️ 시장 변동성	VIX (공포지수)	시장의 불안정성
🏦 안전자산 수요	국채 수익률 스프레드	위험자산 vs 안전자산 수요
💰 안전자산 가격	국채 가격	안전자산 매수 강도
📉 시장 수급	상승 vs 하락 종목 수, 거래량	시장 참여자 동향

Fear & Greed Index

점수 해석

점수	상태	시장 해석
0~25	Extreme Fear (극단적 공포)	투매, 저가 매수 기회
25~50	Fear (공포)	약세장 지속 가능성
50	Neutral (중립)	심리적 균형
50~75	Greed (탐욕)	상승 기대감
75~100	Extreme Greed (극단적 탐욕)	거품 형성 가능성

Fear & Greed Index



Fear & Greed Index

라이브러리 설치

```
# ------------------------
# 1. 필요한 라이브러리 설치
# ------------------------

print("🚀 라이브러리 설치 시작!")
!pip install yfinance pandas matplotlib duckdb streamlit fredapi pyngrok requests --quiet
print("✅ 라이브러리 설치 완료!")
```

Fear & Greed Index

데이터 수집

```
# ------------------------
# 2. 데이터 수집 및 DuckDB 저장
# ------------------------

print("\n🚀 데이터 수집 및 DuckDB 저장 시작!")

import requests
import pandas as pd
import duckdb
import datetime
import matplotlib.pyplot as plt
import os
import threading
import time
from pyngrok import ngrok
import yfinance as yf

# DuckDB 연결
con = duckdb.connect(database='buffett_indicator.duckdb', read_only=False)

# ✅ 2-1. Fear & Greed Index 수집
print("📡 Fear & Greed Index API 호출 중...")
url = 'https://api.alternative.me/fng/?limit=0&format=json&date_format=us'
response = requests.get(url)
data = response.json()
print("✅ API 데이터 수신 완료!")

fng_data = pd.DataFrame(data['data'])
print(f"📊 원본 데이터 행 수: {len(fng_data)}")
columns={'timestamp': 'Date',
         'value': 'FearGreedValue',
         'value_classification': 'Classification'}
fng_data['timestamp'] = pd.to_datetime(fng_data['timestamp'])
fng_data = fng_data.rename(columns=columns)
fng_data['FearGreedValue'] = fng_data['FearGreedValue'].astype(int)
fng_data = fng_data[['Date', 'FearGreedValue', 'Classification']]
fng_data = fng_data.sort_values('Date')
```

```
print("📦 Fear & Greed Index 데이터 DuckDB 저장 중...")
con.execute("CREATE OR REPLACE TABLE fear_greed_index_data AS SELECT * FROM fng_data")
print("✅ 저장 완료! Fear & Greed Index 데이터 미리보기:")
print(con.execute("SELECT * FROM fear_greed_index_data LIMIT 5").fetchdf())

# ✅ 2-2. S&P 500 데이터 수집
print("📩 S&P500 데이터 수집 중...")
sp500 = yf.Ticker("^GSPC")
sp500_df = sp500.history(period="max").reset_index()
sp500_df = sp500_df.rename(columns={'Date': 'Date', 'Close': 'SP500_Close'})
sp500_df['Date'] = pd.to_datetime(sp500_df['Date'])
print(f"📊 S&P500 데이터 행 수: {len(sp500_df)}")

print("📦 S&P500 데이터 DuckDB 저장 중...")
con.execute("CREATE OR REPLACE TABLE sp500_data AS SELECT * FROM sp500_df")
print("✅ 저장 완료! S&P500 데이터 미리보기:")
print(con.execute("SELECT * FROM sp500_data LIMIT 5").fetchdf())
con.close()
```

Fear & Greed Index

시각화

```
# ------------------------
# 3. 시각화: Buffett Indicator + Wilshire 5000 / Fear & Greed + S&P500
# ------------------------

print("\n🚀 시각화 시작!")
con = duckdb.connect(database='buffett_indicator.duckdb', read_only=False)
# ✅ 3-1. Buffett Indicator 데이터 로드
print("📥 DuckDB 에서 Buffett Indicator 데이터 로딩 중...")
buffett_df = con.execute("SELECT * FROM buffett_indicator_data").fetchdf()
print("✅ 로드 완료! Buffett Indicator 데이터:")
print(buffett_df.head())

# ✅ 3-2. Fear & Greed Index 데이터 로드
print("📥 DuckDB 에서 Fear & Greed Index 데이터 로딩 중...")
fng_df = con.execute("SELECT * FROM fear_greed_index_data").fetchdf()
print("✅ 로드 완료! Fear & Greed Index 데이터:")
print(fng_df.head())

fng_df['Date'] = pd.to_datetime(fng_df['Date']).dt.date
# Fear & Greed Index 가 가지고 있는 날짜만 추출
available_dates = fng_df['Date']

# S&P500 도 같은 날짜로 필터링
filtered_sp500_df = sp500_df[sp500_df['Date'].isin(available_dates)]

# Fear & Greed Index 는 원본 유지
print(f"✅ Fear & Greed Index 가 있는 날짜 수: {len(available_dates)}")
print(f"✅ 보간 완료! Fear & Greed Index 데이터 행 수: {len(fng_df)}")

# ✅ 3-3. S&P500 데이터 로드
print("📥 DuckDB 에서 S&P500 데이터 로딩 중...")
sp500_df = con.execute("SELECT * FROM sp500_data").fetchdf()
sp500_df['Date'] = pd.to_datetime(sp500_df['Date']).dt.date
print("✅ 로드 완료! S&P500 데이터:")
print(sp500_df.head())

con.close()
```

Fear & Greed Index

버핏인디케이터 차트

```
# ------------------
# 첫 번째 차트: Buffett Indicator + Wilshire 5000
# ------------------
print("\n📊 Buffett Indicator + Wilshire 5000 시각화 중...")
plt.figure(figsize=(14, 7))
ax1 = plt.gca()
ax1.plot(buffett_df['Date'], buffett_df['Buffett_Indicator'], color='blue', linewidth=1, label='Buffett Indicator (%)')
ax1.fill_between(buffett_df['Date'], 100, 120, color='orange', alpha=0.1)
ax1.fill_between(buffett_df['Date'], 120, buffett_df['Buffett_Indicator'].max(), color='red', alpha=0.1)
ax1.axhline(y=100, color='gray', linestyle='--', linewidth=1)
ax1.axhline(y=120, color='red', linestyle='--', linewidth=1)
ax1.set_xlabel('Date')
ax1.set_ylabel('Buffett Indicator (%)', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

ax2 = ax1.twinx()
ax2.plot(buffett_df['Date'], buffett_df['Market_Cap'], color='purple', linewidth=1, alpha=0.6, label='Wilshire 5000 (Market Cap)')
ax2.set_ylabel('Wilshire 5000 Market Cap', color='purple')
ax2.tick_params(axis='y', labelcolor='purple')

plt.title('Buffett Indicator & Wilshire 5000 Market Cap')
fig = plt.gcf()
fig.tight_layout()
plt.show()
print("✅ 첫 번째 차트 완료!")
```

Fear & Greed Index

공포탐욕지수 차트

```
# --------------------
# 두 번째 차트: Fear & Greed Index + S&P 500
# --------------------
print("\n📊 Fear & Greed Index + S&P 500 시각화 중...")
# Fear & Greed Index 기간에 맞춰 S&P500 데이터 필터링
print("🗓️ Fear & Greed Index 데이터 범위로 S&P500 데이터 필터링 중...")
date_min = fng_df['Date'].min()
date_max = fng_df['Date'].max()
filtered_sp500_df = sp500_df[(sp500_df['Date'] >= date_min) & (sp500_df['Date'] <= date_max)]

merged_sp500 = pd.merge(fng_df, filtered_sp500_df, on='Date', how='inner')
print(f"🔗 병합 완료! 총 {len(merged_sp500)} 행")

plt.figure(figsize=(14, 5))
ax1 = plt.gca()
ax1.plot(merged_sp500['Date'], merged_sp500['FearGreedValue'], color='red', linewidth=0.8, alpha=0.8, label='Fear & Greed Index')
ax1.set_xlabel('Date')
ax1.set_ylabel('Fear & Greed Index', color='green')
ax1.tick_params(axis='y', labelcolor='green')
ax1.set_ylim(0, 100)  # Fear & Greed Index 범위 고정

# 공포/탐욕 구간 강조
ax1.axhspan(0, 20, color='red', alpha=0.1)
ax1.axhspan(20, 40, color='orange', alpha=0.1)
ax1.axhspan(60, 80, color='lightgreen', alpha=0.1)
ax1.axhspan(80, 100, color='green', alpha=0.1)

ax2 = ax1.twinx()
ax2.plot(merged_sp500['Date'], merged_sp500['SP500_Close'], color='blue', linewidth=0.6, alpha=0.6, label='S&P 500 Close')
ax2.set_ylabel('S&P 500 Index', color='blue')
ax2.tick_params(axis='y', labelcolor='blue')

plt.title('Fear & Greed Index & S&P 500 Index')
fig = plt.gcf()
fig.tight_layout()
plt.show()

print("✅ 두 번째 차트 완료!")
```

Fear & Greed Index

대시보드

```
# --------------------------
# 4. Streamlit 대시보드 코드 저장 및 실행
# --------------------------

print("💾 Streamlit 대시보드 코드 저장 중...")
streamlit_dashboard_code = """
import streamlit as st
import pandas as pd
import duckdb
import matplotlib.pyplot as plt

st.set_page_config(page_title="시장 심리 대시보드", layout="wide")
st.title("📈 시장 심리 대시보드: Buffett Indicator & Fear & Greed Index")
st.markdown("미국 시장의 심리적 온도를 두 가지 핵심 지표로 한눈에 파악하세요.")

# DuckDB 연결
con = duckdb.connect(database='buffett_indicator.duckdb', read_only=True)

# Buffett Indicator 데이터 로드
st.subheader("📊 Buffett Indicator & Wilshire 5000 Market Cap")
buffett_df = con.execute("SELECT * FROM buffett_indicator_data").fetchdf()

fig1, ax1 = plt.subplots(figsize=(14, 4))
ax1.plot(buffett_df['Date'], buffett_df['Buffett_Indicator'], color='blue', linewidth=1, label='Buffett Indicator (%)')
ax1.fill_between(buffett_df['Date'], 100, 120, color='orange', alpha=0.1)
ax1.fill_between(buffett_df['Date'], 120, buffett_df['Buffett_Indicator'].max(), color='red', alpha=0.1)
ax1.axhline(y=100, color='gray', linestyle='--', linewidth=1)
ax1.axhline(y=120, color='red', linestyle='--', linewidth=1)
ax1.set_xlabel('Date')
ax1.set_ylabel('Buffett Indicator (%)', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

ax2 = ax1.twinx()
ax2.plot(buffett_df['Date'], buffett_df['Market_Cap'], color='purple', linewidth=1, alpha=0.6, label='Wilshire 5000 Market Cap')
ax2.set_ylabel('Wilshire 5000 Market Cap', color='purple')
ax2.tick_params(axis='y', labelcolor='purple')

fig1.tight_layout()
st.pyplot(fig1)
```

Fear & Greed Index

대시보드

```
# Fear & Greed Index + S&P500 데이터 로드
st.subheader("📊 Fear & Greed Index & S&P 500 Index")
fng_df = con.execute("SELECT * FROM fear_greed_index_data").fetchdf()
sp500_df = con.execute("SELECT * FROM sp500_data").fetchdf()

# 전처리
fng_df['Date'] = pd.to_datetime(fng_df['Date']).dt.date
sp500_df['Date'] = pd.to_datetime(sp500_df['Date']).dt.date

# 병합
available_dates = fng_df['Date']
filtered_sp500_df = sp500_df[sp500_df['Date'].isin(available_dates)]
merged_sp500 = pd.merge(fng_df, filtered_sp500_df, on='Date', how='inner')

fig2, ax1 = plt.subplots(figsize=(14, 4))
ax1.plot(merged_sp500['Date'], merged_sp500['FearGreedValue'], color='green', linewidth=1, alpha=0.6, label='Fear & Greed Index')
ax1.set_xlabel('Date')
ax1.set_ylabel('Fear & Greed Index', color='green')
ax1.tick_params(axis='y', labelcolor='green')
ax1.set_ylim(0, 100)

# 공포/탐욕 구간 강조
ax1.axhspan(0, 20, color='red', alpha=0.1)
ax1.axhspan(20, 40, color='orange', alpha=0.1)
ax1.axhspan(60, 80, color='lightgreen', alpha=0.1)
ax1.axhspan(80, 100, color='green', alpha=0.1)

ax2 = ax1.twinx()
ax2.plot(merged_sp500['Date'], merged_sp500['SP500_Close'], color='orange', linewidth=1, alpha=0.6, label='S&P 500 Close')
ax2.set_ylabel('S&P 500 Index', color='orange')
ax2.tick_params(axis='y', labelcolor='orange')

fig2.tight_layout()
st.pyplot(fig2)

st.success("✅ 대시보드가 성공적으로 로드되었습니다!")
"""

with open("app.py", "w") as file:
    file.write(streamlit_dashboard_code)

print("✅ Streamlit 대시보드 코드 저장 완료!")
```

Fear & Greed Index

서비스 실행

```
# ngrok으로 공개 URL 생성 (Authtoken 필요)
from pyngrok import ngrok

# 여기에 발급받은 ngrok authtoken 입력하세요!
NGROK_AUTH_TOKEN = "여기에 발급받은 ngrok authtoken 입력하세요!"
ngrok.set_auth_token(NGROK_AUTH_TOKEN)

print("[INFO] ngrok 터널 생성 중...")
public_url = ngrok.connect(addr="8501", proto="http")
print(f"[INFO] Streamlit 앱 Public URL: {public_url}")

# Streamlit 앱 실행
!streamlit run app.py --server.port 8501 --server.enableCORS false

print("[INFO] Streamlit 웹 서비스가 실행되었습니다. 위의 Public URL을 클릭하여 접속하세요!")
```