app.py analyzed_stock_data.csv # from your existing pipeline requirements.txt

streamlit pandas numpy yfinance plotly

```
In [10]: %%writefile app.py
# ---------- Block A: Imports & Config ----------
import os
import ast
import numpy as np
import pandas as pd
import plotly.express as px
import streamlit as st
import yfinance as yf
from pathlib import Path

# Safe in Streamlit runtime; harmless if imported elsewhere
try:
    st.set_page_config(page_title="Reddit Sentiment & Backtest", layout="wic
except Exception:
    # Avoid notebook “ScriptRunContext” warning if accidentally executed her
    pass

DATA_FILE = "analyzed_stock_data.csv"   # must exist in the same folder
BENCHMARK = "SPY"
ENTRY = 0.05
EXIT  = -0.05
COST  = 0.001   # 0.1%
TRADING_DAYS = 252
DEFAULT_YEARS = 2  # backtest horizon
```

```
Writing app.py
```

```
In [11]: %%writefile -a app.py
# ---------- Block B: Data loaders ----------
@st.cache_data(show_spinner=False)
def load_reddit(filepath: str) -> pd.DataFrame:
    df = pd.read_csv(filepath)

    # Date handling
    if "date_only" in df.columns:
        df["date_only"] = pd.to_datetime(df["date_only"])
    elif "created_utc" in df.columns:
        df["date_only"] = pd.to_datetime(df["created_utc"]).dt.normalize()
    else:
        raise ValueError("No date column found. Expected 'date_only' or 'cre

    # Parse tickers list
    def parse_tickers(x):
        if isinstance(x, list): return x
        if isinstance(x, str):
            s = x.strip()
            if s.startswith("[") and s.endswith("]"):
                try: return ast.literal_eval(s)
                except Exception: return []
            return [t.strip().upper() for t in s.replace(",", " ").split() i
        return []
    df["tickers"] = df["tickers"].apply(parse_tickers) if "tickers" in df.co

    if "sentiment" not in df.columns:
        raise ValueError("Expected 'sentiment' column in CSV.")
    if "sentiment_label" not in df.columns:
        df["sentiment_label"] = df["sentiment"].apply(lambda s: "Positive" i

    for col in ["title", "selftext", "url", "subreddit", "permalink"]:
        if col not in df.columns: df[col] = ""

    # Make Streamlit cache happy
    df["tickers"] = df["tickers"].apply(tuple)
    return df


@st.cache_data(show_spinner=False)
def daily_signals(df: pd.DataFrame) -> pd.DataFrame:
    """Average sentiment per (date, ticker)."""
    e = df.explode("tickers").dropna(subset=["tickers"])
    return (e.groupby(["date_only", "tickers"], as_index=False)["sentiment"]
              .mean()
              .rename(columns={"tickers": "ticker"}))
```

```
Appending to app.py
```

In [12]:
```
%%writefile -a app.py
# ---------- Block C: Price history ----------
@st.cache_data(show_spinner=False)
def get_price_history(tickers: list[str], start: pd.Timestamp, end: pd.Times
    df = yf.download(
        tickers,
        start=start.date(),
        end=end.date(),
        auto_adjust=True,   # adjusted Close → use "Close"
        progress=False,
        group_by="column",
        threads=True,
    )

    if isinstance(df, pd.Series):
        df = df.to_frame()

    if isinstance(df.columns, pd.MultiIndex):
        if "Close" in set(df.columns.get_level_values(0)):
            prices = df.xs("Close", axis=1, level=0)
        else:
            prices = df.stack(0).groupby(level=[0,1]).last().unstack(1)
    else:
        if "Close" in df.columns and len(tickers) == 1:
            prices = df[["Close"]].rename(columns={"Close": tickers[0]})
        else:
            prices = df.copy()

    if isinstance(prices, pd.Series):
        prices = prices.to_frame()

    prices.index.name = "date_only"
    prices = prices.reindex(columns=[t for t in tickers if t in prices.colum
    return prices
```

Appending to app.py

In [13]:
```
%%writefile -a app.py
# ---------- Block D: Strategy helpers & metrics ----------
def build_daily_weights(signals_wide: pd.DataFrame) -> pd.DataFrame:
    raw_long = (signals_wide > ENTRY).astype(int)
    raw_exit = (signals_wide < EXIT).astype(int)
    desired_pos = raw_long.shift(1).fillna(0).astype(int)
    desired_pos = desired_pos.mask(raw_exit.shift(1) == 1, 0).fillna(0).asty
    weights = desired_pos.div(desired_pos.sum(axis=1).replace(0, np.nan), ax
    return weights

def portfolio_returns(weights: pd.DataFrame, rets: pd.DataFrame, cost: float
    gross = (weights * rets).sum(axis=1)
    turnover = weights.diff().abs().sum(axis=1).fillna(0.0)
    net = gross - cost * turnover
    return net

def perf_metrics(returns: pd.Series, benchmark_returns: pd.Series | None = N
    r = returns.dropna()
    if r.empty: return {}
    equity = (1 + r).cumprod()
    cum_return = equity.iloc[-1] - 1
    ann_return = r.mean() * trading_days
    ann_vol = r.std(ddof=0) * np.sqrt(trading_days)
    sharpe = ann_return / ann_vol if ann_vol > 0 else np.nan
    dd = (equity / equity.cummax() - 1).min()
    downside = r.where(r < 0, 0)
    sortino = ann_return / (downside.std(ddof=0) * np.sqrt(trading_days) or
    out = {
        "Cumulative Return": cum_return,
        "Annualized Return": ann_return,
        "Annualized Volatility": ann_vol,
        "Sharpe (rf=0)": sharpe,
        "Sortino (rf=0)": sortino,
        "Max Drawdown": dd,
    }
    if benchmark_returns is not None and not benchmark_returns.dropna().empt
        te = (r - benchmark_returns.reindex_like(r)).std(ddof=0) * np.sqrt(t
        info = ((ann_return - benchmark_returns.mean() * trading_days) / te)
        out["Information Ratio vs SPY"] = info
        out["Excess Ann Return vs SPY"] = ann_return - benchmark_returns.mea
    return out
```

Appending to app.py

In [14]:
```
%%writefile -a app.py
# ---------- Block E: UI ----------
st.title("📈 Reddit Sentiment Dashboard & 2Y Strategy Backtest")

if not Path(DATA_FILE).exists():
    st.error(f"Could not find `{DATA_FILE}`. Put it next to app.py (or updat
    st.stop()

data = load_reddit(DATA_FILE)
signals = daily_signals(data)

# Date range
col1, col2 = st.columns(2)
end_date = pd.Timestamp.today().normalize()
start_date = end_date - pd.DateOffset(years=2)
start_date = pd.Timestamp(col1.date_input("Start date", start_date))
end_date   = pd.Timestamp(col2.date_input("End date",   end_date))

available_tickers = sorted({t for ts in data["tickers"] for t in ts})

# --- Single ticker search
st.subheader("□ Search a Ticker")
ticker_query = st.text_input("Type a ticker (e.g., NVDA, PLTR)", value=(avai
if ticker_query:
    ex = data.explode("tickers")
    df_t = ex[(ex["tickers"] == ticker_query) & (ex["date_only"] >= start_da
    total_posts = int(len(df_t))
    avg_sent = float(df_t["sentiment"].mean()) if total_posts > 0 else np.na
    dist = df_t["sentiment_label"].value_counts(dropna=False).reindex(["Posi
    c1,c2,c3 = st.columns(3)
    c1.metric("Total posts", f"{total_posts:,}")
    c2.metric("Average sentiment", "N/A" if np.isnan(avg_sent) else f"{avg_s
    c3.metric("Range", f"{start_date.date()} → {end_date.date()}")
    st.plotly_chart(px.bar(pd.DataFrame({"Sentiment":dist.index,"Count":dist
    st.markdown(f"### □ Top Discussions mentioning **{ticker_query}**")
    sort_cols = [c for c in ["score","num_comments"] if c in df_t.columns]
    top_posts = (df_t.sort_values(by=sort_cols, ascending=False) if sort_col
    if top_posts.empty:
        st.info("No posts found in the selected range.")
    else:
        for _, row in top_posts[["date_only","title","url","subreddit"]].fil
            d = pd.to_datetime(row["date_only"]).date() if row["date_only"]
            title = row["title"] or "(no title)"
            sub = row["subreddit"]; url = row["url"]
            bullet = f"- **{d}** — {title}"
            if sub: bullet += f" _(r/{sub})_"
            if url: bullet += f" — [link]({url})"
            st.markdown(bullet)

st.markdown("---")

# --- Portfolio backtest
st.subheader("□ Portfolio Backtest (2 Years) vs SPY")
tickers_input = st.text_input("Enter up to 5 tickers (comma-separated):", va
sel = [t.strip().upper() for t in tickers_input.split(",") if t.strip()]
sel = [t for t in sel if t in available_tickers][:5]
show_per_stock = st.checkbox("Show individual stock strategy curves vs SPY",

if sel:
    sig_wide = (signals.pivot(index="date_only", columns="ticker", values="s
                      .reindex(pd.date_range(start_date, end_date, freq="B")
                       .fillna(0.0))
    sel = [t for t in sel if t in sig_wide.columns]
    if sel:
        sig_wide = sig_wide[sel]
        prices = get_price_history(sel + [BENCHMARK], start=start_date, end=
        if BENCHMARK not in prices.columns:
            st.error("Could not fetch SPY from yfinance."); st.stop()
        common = [t for t in sel if t in prices.columns]
        if not common:
            st.error("No overlap between selected tickers and available pric
        sel = common; sig_wide = sig_wide[sel]
        rets = prices[sel].pct_change().fillna(0.0)
        spy_rets = prices[BENCHMARK].pct_change().fillna(0.0)
        w = build_daily_weights(sig_wide).reindex(columns=sel).fillna(0.0)
        port_net = portfolio_returns(w, rets, cost=COST)
        m = perf_metrics(port_net, spy_rets)
        st.markdown("#### Performance (Net of 0.1% costs)")
        st.dataframe(pd.DataFrame({k:[v] for k,v in m.items()}, index=["Port
        eq_df = pd.DataFrame({"Portfolio (Net)": (1+port_net).cumprod(), "SP
        st.plotly_chart(px.line(eq_df, title="Equity Curve — Portfolio vs SP
        roll = pd.DataFrame({
            "Portfolio 1Y Sharpe": port_net.rolling(TRADING_DAYS).mean()/por
            "SPY 1Y Sharpe": spy_rets.rolling(TRADING_DAYS).mean()/spy_rets.
        }).dropna()
        if not roll.empty:
            st.plotly_chart(px.line(roll, title="Rolling 1-Year Sharpe"), us
        if show_per_stock:
            st.markdown("#### Per-Stock Strategy (Net) vs SPY")
            for t in sel:
                pos  = (sig_wide[t].shift(1) > ENTRY).astype(int)
                exi  = (sig_wide[t].shift(1) < EXIT).astype(int)
                pos  = pos.mask(exi==1, 0).fillna(0).astype(int)
                to   = pos.diff().abs().fillna(0.0)
                net  = pos * rets[t].fillna(0.0) - COST*to
                df_t = pd.DataFrame({f"{t} Strategy (Net)": (1+net).cumprod(
                st.plotly_chart(px.line(df_t, title=f"Equity Curve — {t} vs
    else:
        st.info("Selected tickers not present in your dataset's signals.")
else:
    st.info("Enter up to 5 tickers to run the 2-year backtest vs SPY.")
```

Appending to app.py