

In [3]: *### Niraj Neupane*

```
"""
=====
Project 6 – Automated Paper Trading & Execution Workflow
(ROBUST yfinance handling + risk controls + outputs)
=====

Install deps:
pip install pandas numpy matplotlib yfinance

Run:
python paper_trading_system.py
(or paste into Jupyter and run cells)

What it does:
- Downloads market data (yfinance)
- Builds SMA momentum signal (close > SMA)
- Executes paper trades (next-day open by default)
- Tracks positions/cash/equity/PnL/drawdown
- Applies risk halts (max daily loss, max drawdown)
- Saves CSV outputs + plots
"""

from __future__ import annotations

import os
from dataclasses import dataclass
from typing import Tuple

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

try:
    import yfinance as yf # type: ignore
    HAS_YFINANCE = True
except Exception:
    HAS_YFINANCE = False

# =====
# Config
# =====
@dataclass
class Config:
    ticker: str = "SPY"
    start: str = "2015-01-01"
    end: str = "2025-01-01"

    sma_window: int = 50
    target_leverage: float = 1.0

    commission_per_trade: float = 1.00
```

```

slippage_bps: float = 1.0
use_next_open_execution: bool = True

initial_cash: float = 100_000.0
max_position_leverage: float = 1.0
max_daily_loss_pct: float = 0.03
max_drawdown_pct: float = 0.15

outputs_dir: str = "outputs"
save_csv: bool = True
plot: bool = True

# =====
# Helpers
# =====
def bps_to_decimal(bps: float) -> float:
    return bps / 10_000.0

def _normalize_yf_columns(df: pd.DataFrame) -> pd.DataFrame:
    """
    yfinance sometimes returns MultiIndex columns (e.g., when using groups)
    or inconsistent column names. This normalizes to lower-case single level.
    """
    out = df.copy()

    # If MultiIndex, take the first level that matches OHLCV labels
    if isinstance(out.columns, pd.MultiIndex):
        # Try to map to OHLCV by taking the first level if it contains OHLC labels
        # Otherwise take the last level.
        lvl0 = [str(c).lower() for c in out.columns.get_level_values(0)]
        if any(x in {"open", "high", "low", "close", "adj close", "volume"} for x in lvl0):
            out.columns = out.columns.get_level_values(0)
        else:
            out.columns = out.columns.get_level_values(-1)

    # Normalize names
    out.columns = [str(c).strip().lower() for c in out.columns]
    out = out.rename(columns={"adj close": "adj_close", "adjclose": "adj_close"})

    return out

def download_prices(cfg: Config) -> pd.DataFrame:
    if not HAS_YFINANCE:
        raise ModuleNotFoundError("yfinance not installed. Run: pip install yfinance")

    raw = yf.download(cfg.ticker, start=cfg.start, end=cfg.end, auto_adjust=False,

    if raw is None or raw.empty:
        raise ValueError("No data downloaded. Check ticker/date range.")

    df = _normalize_yf_columns(raw)

    # Keep essential columns (prefer open/close; fall back to close if open missing

```

```

needed = ["close"]
if "open" in df.columns:
    needed = ["open", "close"]
else:
    # If open not present, create open = close (fallback)
    df["open"] = df["close"]
    needed = ["open", "close"]

df = df[needed].copy()
df.index = pd.to_datetime(df.index)
df = df.sort_index()

# Ensure numeric
for c in df.columns:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=["open", "close"])
if df.empty:
    raise ValueError("After cleaning, no usable rows remain (open/close missing)")

return df

def add_features(df: pd.DataFrame, cfg: Config) -> pd.DataFrame:
    out = df.copy()

    if "close" not in out.columns:
        raise ValueError(f"'close' column missing. Available columns: {list(out.columns)}")

    out["sma"] = out["close"].rolling(cfg.sma_window, min_periods=cfg.sma_window).mean()
    out["signal"] = (out["close"] > out["sma"]).astype(int)
    out["target_w"] = out["signal"] * float(cfg.target_leverage)

    return out

def get_execution_prices(df: pd.DataFrame, cfg: Config) -> pd.Series:
    # Decision at close(t), execute at open(t+1) by default
    if cfg.use_next_open_execution:
        return df["open"].shift(-1)
    return df["close"]

# =====
# Simulation
# =====

def simulate(df: pd.DataFrame, cfg: Config) -> Tuple[pd.DataFrame, pd.DataFrame]:
    os.makedirs(cfg.outputs_dir, exist_ok=True)

    exec_px = get_execution_prices(df, cfg)

    ledger = pd.DataFrame(index=df.index)
    ledger["open"] = df["open"]
    ledger["close"] = df["close"]
    ledger["sma"] = df["sma"]
    ledger["signal"] = df["signal"]

```

```

ledger["target_w"] = df["target_w"]
ledger["exec_price"] = exec_px

cash = cfg.initial_cash
shares = 0.0
prev_equity = cfg.initial_cash
peak_equity = cfg.initial_cash
stop_trading = False

slip = bps_to_decimal(cfg.slippage_bps)

# Output columns
for col in ["shares", "cash", "position_value", "equity", "daily_pnl", "drawdown":
    ledger[col] = 0.0
ledger["halted"] = False

trades = []

for dt in ledger.index:
    px_close = float(ledger.loc[dt, "close"])
    px_exec = ledger.loc[dt, "exec_price"]

    # Mark-to-market at close
    position_value = shares * px_close
    equity = cash + position_value
    daily_pnl = equity - prev_equity

    peak_equity = max(peak_equity, equity)
    dd = (equity / peak_equity) - 1.0 if peak_equity > 0 else 0.0

    # Risk halts
    if not stop_trading:
        if prev_equity > 0 and (daily_pnl / prev_equity) <= -cfg.max_daily_loss:
            stop_trading = True
        if dd <= -cfg.max_drawdown_pct:
            stop_trading = True

    # Determine target
    target_w = float(ledger.loc[dt, "target_w"])
    if stop_trading:
        target_w = 0.0 # flatten

    # If we can't execute (NaN on last date when using next open), skip trading
    if pd.isna(px_exec) or float(px_exec) <= 0:
        target_shares = shares
    else:
        target_value = target_w * equity
        target_shares = target_value / float(px_exec)

    # Cap Leverage
    max_val = cfg.max_position_leverage * equity
    if abs(target_value) > max_val:
        target_value = np.sign(target_value) * max_val
        target_shares = target_value / float(px_exec)

    delta = target_shares - shares

```

```

if abs(delta) > 1e-9 and (not pd.isna(px_exec)) and float(px_exec) > 0:
    side = "BUY" if delta > 0 else "SELL"
    qty = abs(delta)
    price = float(px_exec)
    notional = qty * price

    slippage_cost = notional * slip
    commission = cfg.commission_per_trade

    if side == "BUY":
        cash_change = -(notional + slippage_cost + commission)
        # no margin: scale down if insufficient cash
        if cash + cash_change < 0:
            avail = max(cash - commission, 0.0)
            max_notional = avail / (1.0 + slip) if (1.0 + slip) > 0 else 0.
            qty = max_notional / price if price > 0 else 0.0
            if qty <= 0:
                cash_change = 0.0
                delta = 0.0
            else:
                notional = qty * price
                slippage_cost = notional * slip
                cash_change = -(notional + slippage_cost + commission)
                delta = qty # positive
        else:
            cash_change = +(notional - slippage_cost - commission)

    if abs(delta) > 1e-9:
        cash += cash_change
        shares += np.sign(delta) * qty

    trades.append(
        {
            "date": dt,
            "side": side,
            "qty": qty,
            "price": price,
            "notional": notional,
            "commission": commission,
            "slippage_cost": slippage_cost,
            "reason": "risk_halt_flatten" if stop_trading else "signal_
        }
    )

# Recompute end-of-day
position_value = shares * px_close
equity = cash + position_value
daily_pnl = equity - prev_equity
peak_equity = max(peak_equity, equity)
dd = (equity / peak_equity) - 1.0 if peak_equity > 0 else 0.0

ledger.loc[dt, "shares"] = shares
ledger.loc[dt, "cash"] = cash
ledger.loc[dt, "position_value"] = position_value
ledger.loc[dt, "equity"] = equity

```

```

        ledger.loc[dt, "daily_pnl"] = daily_pnl
        ledger.loc[dt, "drawdown"] = dd
        ledger.loc[dt, "halted"] = stop_trading

    prev_equity = equity

    trades_df = pd.DataFrame(trades)
    if not trades_df.empty:
        trades_df["date"] = pd.to_datetime(trades_df["date"])
        trades_df = trades_df.set_index("date").sort_index()

    return ledger, trades_df

def perf_stats(equity: pd.Series) -> pd.Series:
    eq = equity.dropna()
    rets = eq.pct_change().dropna()

    total_ret = (eq.iloc[-1] / eq.iloc[0]) - 1.0
    ann_ret = (1.0 + total_ret) ** (252 / max(len(rets), 1)) - 1.0 if len(rets) > 0
    ann_vol = rets.std() * np.sqrt(252) if len(rets) > 1 else np.nan
    sharpe = ann_ret / ann_vol if (ann_vol is not np.nan and ann_vol and ann_vol >

    dd = (eq / eq.cummax()) - 1.0
    max_dd = dd.min() if len(dd) else np.nan

    return pd.Series(
        {
            "Total Return": total_ret,
            "Annualized Return": ann_ret,
            "Annualized Volatility": ann_vol,
            "Sharpe (rf=0)": sharpe,
            "Max Drawdown": max_dd,
            "Start Equity": float(eq.iloc[0]),
            "End Equity": float(eq.iloc[-1]),
            "Num Days": int(len(eq)),
        }
    )

def plot_results(ledger: pd.DataFrame, cfg: Config) -> None:
    plt.figure()
    ledger["equity"].plot(title=f"Equity Curve - {cfg.ticker}")
    plt.xlabel("Date")
    plt.ylabel("Equity ($)")
    plt.show()

    plt.figure()
    ledger["drawdown"].plot(title="Drawdown")
    plt.xlabel("Date")
    plt.ylabel("Drawdown")
    plt.show()

def main() -> None:
    cfg = Config()

```

```

print("Loading market data...")
df = download_prices(cfg)

print("Adding features...")
df = add_features(df, cfg)

# Drop early rows where SMA isn't ready
df = df.dropna(subset=["sma"]).copy()
if df.empty:
    raise ValueError("After SMA warmup, no rows remain. Try smaller sma_window")

print("Running simulation...")
ledger, trades = simulate(df, cfg)

stats = perf_stats(ledger["equity"])
print("\nPerformance Summary:")
print(stats.round(4))

if cfg.save_csv:
    os.makedirs(cfg.outputs_dir, exist_ok=True)
    ledger_path = os.path.join(cfg.outputs_dir, f"{cfg.ticker.lower()}_ledger.csv")
    trades_path = os.path.join(cfg.outputs_dir, f"{cfg.ticker.lower()}_trades.csv")
    stats_path = os.path.join(cfg.outputs_dir, f"{cfg.ticker.lower()}_stats.csv")

    ledger.to_csv(ledger_path)
    if not trades.empty:
        trades.to_csv(trades_path)
    stats.to_frame("value").to_csv(stats_path)

    print(f"\nSaved outputs in '{cfg.outputs_dir}':")
    print(f"- {ledger_path}")
    if not trades.empty:
        print(f"- {trades_path}")
    print(f"- {stats_path}")

if cfg.plot:
    plot_results(ledger, cfg)

if __name__ == "__main__":
    main()

```

Loading market data...

Adding features...

Running simulation...

Performance Summary:

Total Return	-0.1387
Annualized Return	-0.0151
Annualized Volatility	0.0342
Sharpe (rf=0)	-0.4434
Max Drawdown	-0.1464
Start Equity	100417.4769
End Equity	86485.0884
Num Days	2467.0000

dtype: float64

Saved outputs in 'outputs/':

- outputs\spy_ledger.csv
- outputs\spy_trades.csv
- outputs\spy_stats.csv



