In [2]:

```
### Niraj Neupane

# ============================================================
# Project 5: Machine Learning for Trading Signal Evaluation (FIXED)
# ============================================================
# - Downloads SPY data (robust to yfinance column formats)
# - Builds features (no lookahead)
# - Walk-forward ML evaluation (rolling train, forward test)
# - Converts probabilities to a simple long/flat strategy
# - Saves CSV outputs and plots equity curves
#
# Install:
#   pip install numpy pandas matplotlib yfinance scikit-learn
# ============================================================

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

try:
    import yfinance as yf
except ModuleNotFoundError as e:
    raise ModuleNotFoundError("Missing yfinance. Install with: pip install yfinance

try:
    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
except ModuleNotFoundError as e:
    raise ModuleNotFoundError("Missing scikit-learn. Install with: pip install scik

plt.style.use("seaborn-v0_8")

# ----------------------------
# Config
# ----------------------------
OUTPUT_DIR = "outputs"
os.makedirs(OUTPUT_DIR, exist_ok=True)

TICKER = "SPY"
START = "2005-01-01"
END = None

TRAIN_YEARS = 5
TEST_MONTHS = 3
MIN_TRAIN_SAMPLES = 500

PROB_THRESHOLD = 0.52
TCOST_BPS = 2.0
RANDOM_STATE = 42
```

```python
# ----------------------------
# Robust download helper
# ----------------------------
def download_close_series(ticker: str, start: str, end=None) -> pd.Series:
    """
    Robustly return a 1-D Series of Close prices for a single ticker.
    Handles yfinance DataFrame, Series, and MultiIndex columns.
    """
    df = yf.download(ticker, start=start, end=end, auto_adjust=True, progress=False

    if df is None or len(df) == 0:
        raise ValueError("No data returned from yfinance. Check ticker or internet

    # Case 1: typical DataFrame with 'Close'
    if isinstance(df, pd.DataFrame):
        # MultiIndex columns sometimes appear, e.g. ('Close', 'SPY')
        if isinstance(df.columns, pd.MultiIndex):
            # Find the 'Close' level
            if "Close" in df.columns.get_level_values(0):
                close = df["Close"]
                # If still DataFrame (because second level), select the ticker colu
                if isinstance(close, pd.DataFrame):
                    if ticker in close.columns:
                        close = close[ticker]
                    else:
                        # fallback: take first column
                        close = close.iloc[:, 0]
            else:
                raise ValueError("Downloaded data has MultiIndex columns but no 'Cl
        else:
            if "Close" not in df.columns:
                raise ValueError("Downloaded data does not contain a 'Close' column
            close = df["Close"]

    # Case 2: Series already
    elif isinstance(df, pd.Series):
        close = df
    else:
        raise ValueError("Unexpected data type returned from yfinance.")

    close = close.dropna()
    close.name = ticker
    return close


# ----------------------------
# Feature engineering
# ----------------------------
def make_features(close: pd.Series) -> pd.DataFrame:
    """
    Build features using information available at time t to predict y_{t+1}.
    """
    df = pd.DataFrame(index=close.index)
    df["close"] = close
```

```
    df["ret_1d"] = df["close"].pct_change()
    df["ret_5d"] = df["close"].pct_change(5)
    df["ret_21d"] = df["close"].pct_change(21)

    df["mom_5d"] = df["ret_1d"].rolling(5).mean()
    df["mom_21d"] = df["ret_1d"].rolling(21).mean()
    df["mom_63d"] = df["ret_1d"].rolling(63).mean()

    df["vol_21d"] = df["ret_1d"].rolling(21).std()
    df["vol_63d"] = df["ret_1d"].rolling(63).std()

    df["ma_20"] = df["close"].rolling(20).mean()
    df["ma_50"] = df["close"].rolling(50).mean()
    df["ma_200"] = df["close"].rolling(200).mean()

    df["trend_20"] = (df["close"] / df["ma_20"]) - 1
    df["trend_50"] = (df["close"] / df["ma_50"]) - 1
    df["trend_200"] = (df["close"] / df["ma_200"]) - 1

    up = df["ret_1d"].clip(lower=0)
    down = (-df["ret_1d"]).clip(lower=0)
    rs = up.rolling(14).mean() / (down.rolling(14).mean() + 1e-12)
    df["rsi_14"] = 100 - (100 / (1 + rs))

    # Target: next-day return and direction
    df["fwd_ret_1d"] = df["ret_1d"].shift(-1)
    df["y"] = (df["fwd_ret_1d"] > 0).astype(int)

    df = df.dropna()
    return df


# ------------------------------
# Walk-forward splits
# ------------------------------
def walk_forward_splits(index: pd.DatetimeIndex, train_years: int, test_months: int
    start_date = index.min()
    end_date = index.max()
    test_start = start_date + pd.DateOffset(years=train_years)

    while test_start < end_date:
        train_start = test_start - pd.DateOffset(years=train_years)
        train_end = test_start - pd.DateOffset(days=1)

        test_end = test_start + pd.DateOffset(months=test_months) - pd.DateOffset(d
        yield train_start, train_end, test_start, min(test_end, end_date)

        test_start = test_start + pd.DateOffset(months=test_months)


# ------------------------------
# Backtest utilities
# ------------------------------
def perf_metrics(returns: pd.Series) -> dict:
    r = returns.dropna()
    if len(r) < 50:
```

```
        return {"n": int(len(r))}

    ann_ret = (1 + r).prod() ** (252 / len(r)) - 1
    ann_vol = r.std() * np.sqrt(252)
    sharpe = np.nan if ann_vol == 0 else ann_ret / ann_vol

    eq = (1 + r).cumprod()
    dd = eq / eq.cummax() - 1
    mdd = dd.min()

    win_rate = (r > 0).mean()

    return {
        "n": int(len(r)),
        "ann_return": float(ann_ret),
        "ann_vol": float(ann_vol),
        "sharpe": float(sharpe) if np.isfinite(sharpe) else np.nan,
        "max_drawdown": float(mdd),
        "win_rate": float(win_rate),
    }


def apply_transaction_costs(position: pd.Series, returns: pd.Series, tcost_bps: flo
    tcost = tcost_bps / 10000.0
    turnover = (position - position.shift(1)).abs().fillna(0.0)
    cost = turnover * tcost
    strat = position.shift(1).fillna(0.0) * returns - cost
    return strat


# ------------------------------
# Main
# ------------------------------
def main():
    print("Downloading data...")
    close = download_close_series(TICKER, START, END)

    data = make_features(close)
    feature_cols = [
        "ret_1d", "ret_5d", "ret_21d",
        "mom_5d", "mom_21d", "mom_63d",
        "vol_21d", "vol_63d",
        "trend_20", "trend_50", "trend_200",
        "rsi_14",
    ]

    X_all = data[feature_cols].copy()
    y_all = data["y"].copy()
    fwd_ret = data["fwd_ret_1d"].copy()
    idx = data.index

    logit = Pipeline([
        ("scaler", StandardScaler()),
        ("clf", LogisticRegression(max_iter=2000, random_state=RANDOM_STATE))
    ])
```

```
    rf = RandomForestClassifier(
        n_estimators=400,
        max_depth=6,
        min_samples_leaf=50,
        random_state=RANDOM_STATE,
        n_jobs=-1
    )

    rows = []

    for train_start, train_end, test_start, test_end in walk_forward_splits(idx, TR
        train_mask = (idx >= train_start) & (idx <= train_end)
        test_mask = (idx >= test_start) & (idx <= test_end)

        X_train, y_train = X_all.loc[train_mask], y_all.loc[train_mask]
        X_test, y_test = X_all.loc[test_mask], y_all.loc[test_mask]

        if len(X_train) < MIN_TRAIN_SAMPLES or len(X_test) < 20:
            continue

        logit.fit(X_train, y_train)
        rf.fit(X_train, y_train)

        p_logit = logit.predict_proba(X_test)[:, 1]
        p_rf = rf.predict_proba(X_test)[:, 1]

        tmp = pd.DataFrame(index=X_test.index)
        tmp["y_true"] = y_test.values
        tmp["fwd_ret"] = fwd_ret.loc[X_test.index].values
        tmp["p_logit"] = p_logit
        tmp["p_rf"] = p_rf
        rows.append(tmp)

    pred = pd.concat(rows).sort_index()

    if pred.empty:
        raise ValueError("No walk-forward predictions were produced. Try reducing T

    pred["pred_logit"] = (pred["p_logit"] >= 0.5).astype(int)
    pred["pred_rf"] = (pred["p_rf"] >= 0.5).astype(int)

    pred["pos_logit"] = (pred["p_logit"] >= PROB_THRESHOLD).astype(float)
    pred["pos_rf"] = (pred["p_rf"] >= PROB_THRESHOLD).astype(float)

    pred["ret_logit"] = apply_transaction_costs(pred["pos_logit"], pred["fwd_ret"],
    pred["ret_rf"] = apply_transaction_costs(pred["pos_rf"], pred["fwd_ret"], TCOST

    pred["ret_buy_hold"] = pred["fwd_ret"]

    mom_signal = (data["ret_21d"].reindex(pred.index) > 0).astype(float)
    pred["ret_momentum"] = apply_transaction_costs(mom_signal, pred["fwd_ret"], TCO

    pred["eq_logit"] = (1 + pred["ret_logit"].fillna(0)).cumprod()
    pred["eq_rf"] = (1 + pred["ret_rf"].fillna(0)).cumprod()
    pred["eq_buy_hold"] = (1 + pred["ret_buy_hold"].fillna(0)).cumprod()
    pred["eq_momentum"] = (1 + pred["ret_momentum"].fillna(0)).cumprod()
```

```
    def cls_metrics(y_true, prob, pred_label):
        out = {
            "accuracy": accuracy_score(y_true, pred_label),
            "precision": precision_score(y_true, pred_label, zero_division=0),
            "recall": recall_score(y_true, pred_label, zero_division=0),
            "f1": f1_score(y_true, pred_label, zero_division=0),
        }
        try:
            out["roc_auc"] = roc_auc_score(y_true, prob)
        except Exception:
            out["roc_auc"] = np.nan
        return out

    m_logit = cls_metrics(pred["y_true"], pred["p_logit"], pred["pred_logit"])
    m_rf = cls_metrics(pred["y_true"], pred["p_rf"], pred["pred_rf"])

    t_logit = perf_metrics(pred["ret_logit"])
    t_rf = perf_metrics(pred["ret_rf"])
    t_bh = perf_metrics(pred["ret_buy_hold"])
    t_mom = perf_metrics(pred["ret_momentum"])

    metrics_df = pd.DataFrame([
        {"model": "LogisticRegression", **m_logit, **{f"trade_{k}": v for k, v in t
        {"model": "RandomForest", **m_rf, **{f"trade_{k}": v for k, v in t_rf.items
        {"model": "BuyHold", **{f"trade_{k}": v for k, v in t_bh.items()}},
        {"model": "Momentum_21d", **{f"trade_{k}": v for k, v in t_mom.items()}},
    ])

    print("\n===== OUT-OF-SAMPLE METRICS =====")
    print(metrics_df)

    pred.to_csv(os.path.join(OUTPUT_DIR, "ml_predictions.csv"))
    pred[["eq_logit", "eq_rf", "eq_buy_hold", "eq_momentum"]].to_csv(os.path.join(O
    metrics_df.to_csv(os.path.join(OUTPUT_DIR, "ml_metrics.csv"), index=False)

    print("\nSaved outputs to ./outputs")

    plt.figure(figsize=(11, 6))
    pred["eq_buy_hold"].plot(label="Buy & Hold")
    pred["eq_momentum"].plot(label="Momentum (21d)")
    pred["eq_logit"].plot(label=f"Logit (thr={PROB_THRESHOLD})")
    pred["eq_rf"].plot(label=f"RF (thr={PROB_THRESHOLD})")
    plt.title("Project 5 — ML Trading Signals (Walk-Forward OOS Equity Curves)")
    plt.ylabel("Growth of $1")
    plt.legend()
    plt.tight_layout()
    plt.show()


# In Jupyter, just run main()
# In .py file, it will run automatically
if __name__ == "__main__":
    main()
```

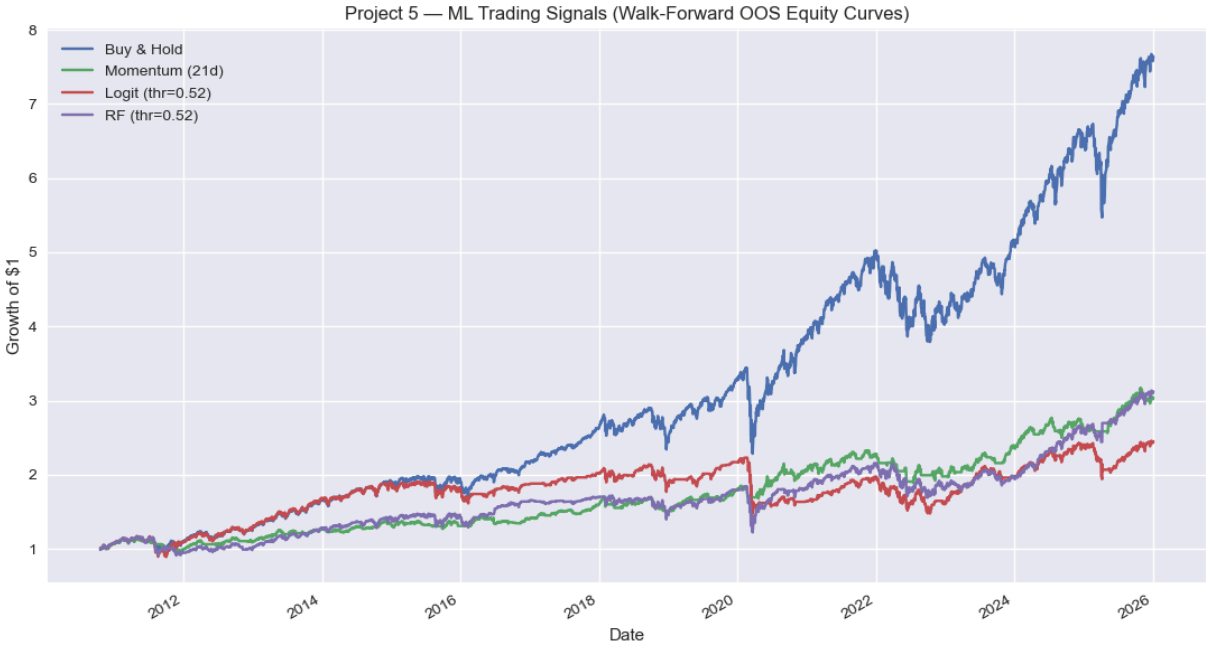
```
Downloading data...

===== OUT-OF-SAMPLE METRICS =====
                model  accuracy  precision    recall       f1   roc_auc  \
0  LogisticRegression  0.538160   0.555486  0.837418  0.66792  0.493419
1        RandomForest  0.542342   0.556400  0.862394  0.67640  0.490348
2             BuyHold       NaN        NaN       NaN      NaN       NaN
3        Momentum_21d       NaN        NaN       NaN      NaN       NaN

   trade_n  trade_ann_return  trade_ann_vol  trade_sharpe  trade_max_drawdown  \
0     3826          0.060734       0.152375      0.398585           -0.340487
1     3826          0.077849       0.154815      0.502850           -0.337173
2     3826          0.143317       0.170992      0.838151           -0.337173
3     3826          0.076024       0.106298      0.715193           -0.187479

   trade_win_rate
0        0.404861
1        0.405907
2        0.554626
3        0.385781

Saved outputs to ./outputs
```



In [ ]: