# Stock Predictor – Detailed Project Guide

An end-to-end framework for daily stock prediction, evaluation, backtesting, CLI, and Streamlit app.

## Overview

• Data ingestion via yfinance with on-disk caching.

• Feature engineering with technical indicators, returns, and leak-free lags.

• Time-aware train/validation/test splitting.

• Models: RandomForest (default) and optional XGBoost.

• Backtesting: threshold-based long-only with costs.

• CLI to run the pipeline and a Streamlit app for interactivity.

## Project structure

• src/stock_predictor/config.py – paths, constants, auto-dir creation

• src/stock_predictor/utils.py – logging and IO helpers

• src/stock_predictor/data/ingest.py – yfinance download + MultiIndex handling + parquet cache

• src/stock_predictor/features/engineering.py – indicators, returns, lags, target

• src/stock_predictor/models/train.py – train/evaluate/save model

• src/stock_predictor/models/evaluate.py – evaluate saved model on a window

• src/stock_predictor/backtest/backtest.py – simple long-only threshold backtest

• src/stock_predictor/cli.py – Typer commands

• streamlit_app.py – UI for training and backtesting

## Data ingestion

Function download_prices(ticker, start, end, interval='1d'): downloads OHLCV, normalizes timestamp, flattens Yahoo MultiIndex columns when needed, title-cases column names, and caches parquet under data/raw/.

## Feature engineering

• Trend: SMA(10/20), EMA(10/20)

• Momentum: RSI(14), MACD (line/signal/hist)

• Volatility: Bollinger bands (high/low/width)

• Returns: 1/5/10-day pct changes

• Leakage prevention: features lagged by 1 day; drop NaNs

• Target: next-day return (Close.pct_change().shift(-1))

## Modeling

• Split: ~70% train, 15% val, 15% test (by time).

• RandomForestRegressor as default (no special system deps).

• XGBoost optional: lazy import; macOS needs libomp (brew install libomp).

• Metrics: RMSE, MAE, $R^2$, MAPE (caution near zero).

• Persisted artifact: models/{TICKER}_{model_type}.joblib with model + feature names + metrics.

# Backtesting

• Signal: long if predicted next-day return > threshold, else cash.

• Transaction cost on signal changes (enter/exit).

• Equity vs Buy & Hold, metrics: final equity, annualized return, Sharpe, win rate.

• Intended as a baseline; customize sizing, shorting, slippage, multi-asset.

# CLI usage

• Help: python -m src.stock_predictor.cli --help

• Train: python -m src.stock_predictor.cli train AAPL --start 2018-01-01 --end 2024-12-31 --model-type rf

• Evaluate: python -m src.stock_predictor.cli evaluate AAPL --start 2019-01-01 --end 2024-12-31

• Backtest: python -m src.stock_predictor.cli run-backtest AAPL --start 2019-01-01 --end 2024-12-31 --threshold 0.0

• UI: streamlit run streamlit_app.py

# Real-time trading (ELI5)

Daily loop after market close: download latest → build features → load model → predict → decide → place order via broker API (e.g., Alpaca/IBKR).

• Start with paper trading (sandbox).

• Use a scheduler (cron/launchd) to run at a fixed time.

• Position sizing: fixed $ or % of cash; add max position; stop-loss.

• Keep logs of predictions and orders.

# Scope for modification

• Targets: classification (up/down), multi-step horizons, volatility forecasts.

• Models: linear models, gradient boosting, neural nets, ensembles; hyperparameter tuning with Optuna.

• Validation: walk-forward CV, expanding/rolling windows.

• Features: macro data, cross-asset signals, alternative data, regime features.

• Backtest: slippage model, borrow/short costs, portfolio allocation, risk overlays.

# Appendix – Tips

• MAPE on near-zero returns is misleading – prefer RMSE/MAE.

• Normalize/standardize features if using linear models or neural nets.

- Document your experiments; version your models and data windows.