Education Screener — Code Bundle (Dated 2025-09-15)

This document snapshots Lynn's educational stock screener code skeleton for provenance and dating.

# README.md

# Education Stock Screener — Code Pack (Dated 2025-09-15)  
  
This archive snapshots the current \*\*design and skeleton code\*\* for Lynn's educational stock screener.  
It is intended for \*\*idea dating / provenance\*\* and will be followed by runnable versions.  
  
## Files  
- `.env.template` — API keys and DB connection (rename to `.env`).  
- `docker-compose.yml` — MySQL service.  
- `schema.sql` — Database schema.  
- `ingest\_prices.py` — Tiingo prices ingestion.  
- `ingest\_fundamentals.py` — Tiingo fundamentals ingestion.  
- `compute\_snapshot.py` — Factor computations and traffic lights.  
  
## Core Factors  
- Fundamentals: \*\*ROE > 10%\*\*, \*\*FCF > 0\*\*, \*\*NetDebt/EBITDA < 3\*\*  
- Valuation: \*\*PE/PS below 5-year 20th percentile\*\*  
- Technical: \*\*6M drawdown ≤ -40%\*\*, \*\*RSI(14) < 35\*\*, \*\*10D Volume Up\*\*  
  
## Notice  
- This version is a \*\*skeleton\*\*. Endpoints/fields may require adjustment to match your specific Tiingo plan and data format.  
- Use this as a dated record of authorship and intent.

# .env.template

# Created: 2025-09-15  
# Rename this file to .env in your project root.  
TIINGO\_API\_KEY=YOUR\_TIINGO\_KEY\_HERE  
DB\_HOST=localhost  
DB\_PORT=3306  
DB\_USER=lynn  
DB\_PASSWORD=strongpass  
DB\_NAME=stocks

# docker-compose.yml

# Created: 2025-09-15  
version: '3.8'  
services:  
 db:  
 image: mysql:8.0  
 container\_name: stocks-mysql  
 restart: always  
 environment:  
 MYSQL\_ROOT\_PASSWORD: rootpass  
 MYSQL\_DATABASE: stocks  
 MYSQL\_USER: lynn  
 MYSQL\_PASSWORD: strongpass  
 ports:  
 - "3306:3306"  
 volumes:  
 - ./mysql\_data:/var/lib/mysql

# schema.sql

-- Created: 2025-09-15  
-- Minimal schema for prices, fundamentals, and snapshots  
CREATE TABLE IF NOT EXISTS prices (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 ticker VARCHAR(16) NOT NULL,  
 date DATE NOT NULL,  
 open DECIMAL(18,6),  
 high DECIMAL(18,6),  
 low DECIMAL(18,6),  
 close DECIMAL(18,6),  
 adj\_close DECIMAL(18,6),  
 volume BIGINT,  
 UNIQUE KEY uq\_prices (ticker, date)  
);  
  
CREATE TABLE IF NOT EXISTS fundamentals\_q (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 ticker VARCHAR(16) NOT NULL,  
 period\_end DATE NOT NULL,  
 revenue DECIMAL(20,2),  
 net\_income DECIMAL(20,2),  
 operating\_cf DECIMAL(20,2),  
 capex DECIMAL(20,2),  
 total\_equity DECIMAL(20,2),  
 total\_debt DECIMAL(20,2),  
 cash\_and\_equivalents DECIMAL(20,2),  
 shares\_outstanding DECIMAL(20,2),  
 UNIQUE KEY uq\_fq (ticker, period\_end)  
);  
  
CREATE TABLE IF NOT EXISTS factor\_snapshot (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 ticker VARCHAR(16) NOT NULL,  
 snapshot\_date DATE NOT NULL,  
 roe DECIMAL(10,4),  
 fcf DECIMAL(20,2),  
 net\_debt\_to\_ebitda DECIMAL(10,4),  
 pe\_ttm DECIMAL(10,4),  
 pe\_percentile\_5y DECIMAL(10,4),  
 ps\_ttm DECIMAL(10,4),  
 ps\_percentile\_5y DECIMAL(10,4),  
 dd\_6m DECIMAL(10,4),  
 rsi\_14 DECIMAL(10,4),  
 vol\_trend\_10d DECIMAL(10,4),  
 traffic\_fundamental VARCHAR(8),  
 traffic\_valuation VARCHAR(8),  
 traffic\_technical VARCHAR(8),  
 notes TEXT,  
 UNIQUE KEY uq\_snap (ticker, snapshot\_date)  
);

# ingest\_prices.py

"""  
ingest\_prices.py — Created: 2025-09-15  
Purpose: Pull daily OHLCV (and adj\_close) from Tiingo for a list of tickers and store into MySQL.  
Status: Educational skeleton; adjust field names/schema per your Tiingo plan and pandas versions.  
"""  
import os  
import time  
import requests  
import pandas as pd  
import sqlalchemy as sa  
  
TIINGO\_API\_KEY = os.getenv("TIINGO\_API\_KEY", "YOUR\_TIINGO\_KEY\_HERE")  
DB\_HOST = os.getenv("DB\_HOST", "localhost")  
DB\_PORT = os.getenv("DB\_PORT", "3306")  
DB\_USER = os.getenv("DB\_USER", "lynn")  
DB\_PASSWORD = os.getenv("DB\_PASSWORD", "strongpass")  
DB\_NAME = os.getenv("DB\_NAME", "stocks")  
  
ENGINE = sa.create\_engine(f"mysql+pymysql://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}")  
  
def fetch\_prices(ticker: str, start='2015-01-01', end=None) -> pd.DataFrame:  
 base = "https://api.tiingo.com/tiingo/daily/{ticker}/prices"  
 params = {  
 "startDate": start,  
 \*\*({"endDate": end} if end else {}),  
 "token": TIINGO\_API\_KEY  
 }  
 url = base.format(ticker=ticker)  
 r = requests.get(url, params=params, timeout=60)  
 r.raise\_for\_status()  
 df = pd.DataFrame(r.json())  
 if df.empty:  
 return df  
 # Standardize columns expected by schema  
 df.rename(columns={  
 "adjClose": "adj\_close",  
 "adjHigh": "adj\_high",  
 "adjLow": "adj\_low",  
 "adjOpen": "adj\_open",  
 "adjVolume": "adj\_volume"  
 }, inplace=True, errors="ignore")  
 df["ticker"] = ticker.upper()  
 df["date"] = pd.to\_datetime(df["date"]).dt.date  
 out = df[[  
 "ticker","date","open","high","low","close","adjClose","volume"  
 ]].copy()  
 out.rename(columns={"adjClose":"adj\_close"}, inplace=True)  
 return out  
  
def upsert\_prices(df: pd.DataFrame):  
 if df.empty:  
 return  
 with ENGINE.begin() as conn:  
 df.to\_sql("prices", conn, if\_exists="append", index=False)  
 # For true upsert, you'd run a post-insert dedup/merge step (implementation varies).  
  
def run(tickers):  
 for tk in tickers:  
 try:  
 df = fetch\_prices(tk)  
 upsert\_prices(df)  
 time.sleep(1)  
 print(f"[OK] {tk}: {len(df)} rows")  
 except Exception as e:  
 print(f"[ERROR] {tk}: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 tickers = sys.argv[1:] or ["META","AAPL","TSLA","SOFI","SAIA","UNH"]  
 run(tickers)

# ingest\_fundamentals.py

"""  
ingest\_fundamentals.py — Created: 2025-09-15  
Purpose: Pull quarterly fundamentals (revenue, net income, cash flows, debt, equity, etc.) and store in MySQL.  
Status: Educational skeleton — field names may require alignment with your Tiingo endpoints.  
"""  
import os  
import time  
import requests  
import pandas as pd  
import sqlalchemy as sa  
  
TIINGO\_API\_KEY = os.getenv("TIINGO\_API\_KEY", "YOUR\_TIINGO\_KEY\_HERE")  
DB\_HOST = os.getenv("DB\_HOST", "localhost")  
DB\_PORT = os.getenv("DB\_PORT", "3306")  
DB\_USER = os.getenv("DB\_USER", "lynn")  
DB\_PASSWORD = os.getenv("DB\_PASSWORD", "strongpass")  
DB\_NAME = os.getenv("DB\_NAME", "stocks")  
  
ENGINE = sa.create\_engine(f"mysql+pymysql://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}")  
  
def fetch\_quarterly(ticker: str) -> pd.DataFrame:  
 # Note: Adjust to correct Tiingo fundamentals endpoint & json path per your plan  
 url = f"https://api.tiingo.com/tiingo/fundamentals/{{ticker}}/statements"  
 params = {"format":"json","token": TIINGO\_API\_KEY}  
 r = requests.get(url.format(ticker=ticker), params=params, timeout=90)  
 r.raise\_for\_status()  
 data = r.json()  
 # Expecting list of statements; normalize as needed  
 records = []  
 for item in data:  
 period\_end = item.get("periodEndDate")  
 # These keys are illustrative; align to actual response keys  
 records.append({  
 "ticker": ticker.upper(),  
 "period\_end": period\_end,  
 "revenue": item.get("revenue"),  
 "net\_income": item.get("netIncome"),  
 "operating\_cf": item.get("netCashFromOperations"),  
 "capex": item.get("capitalExpenditures"),  
 "total\_equity": item.get("totalShareholderEquity"),  
 "total\_debt": item.get("totalDebt"),  
 "cash\_and\_equivalents": item.get("cashAndShortTermInvestments"),  
 "shares\_outstanding": item.get("sharesBasic")  
 })  
 df = pd.DataFrame.from\_records(records)  
 if not df.empty:  
 df["period\_end"] = pd.to\_datetime(df["period\_end"]).dt.date  
 return df  
  
def upsert\_fundamentals(df: pd.DataFrame):  
 if df.empty:  
 return  
 with ENGINE.begin() as conn:  
 df.to\_sql("fundamentals\_q", conn, if\_exists="append", index=False)  
  
def run(tickers):  
 for tk in tickers:  
 try:  
 df = fetch\_quarterly(tk)  
 upsert\_fundamentals(df)  
 time.sleep(1)  
 print(f"[OK] {tk}: {len(df)} quarters")  
 except Exception as e:  
 print(f"[ERROR] {tk}: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 tickers = sys.argv[1:] or ["META","AAPL","TSLA","SOFI","SAIA","UNH"]  
 run(tickers)

# compute\_snapshot.py

"""  
compute\_snapshot.py — Created: 2025-09-15  
Purpose: Compute factors (ROE, FCF, NetDebt/EBITDA, PE/PS percentiles, 6M drawdown, RSI, volume trend)  
and write summarized results into factor\_snapshot.  
Status: Educational skeleton; replace placeholders with actual joins/TTM calculations.  
"""  
import os  
import pandas as pd  
import numpy as np  
import sqlalchemy as sa  
from datetime import date, timedelta  
  
DB\_HOST = os.getenv("DB\_HOST", "localhost")  
DB\_PORT = os.getenv("DB\_PORT", "3306")  
DB\_USER = os.getenv("DB\_USER", "lynn")  
DB\_PASSWORD = os.getenv("DB\_PASSWORD", "strongpass")  
DB\_NAME = os.getenv("DB\_NAME", "stocks")  
  
ENGINE = sa.create\_engine(f"mysql+pymysql://{DB\_USER}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}")  
  
def rsi(series: pd.Series, period: int = 14) -> pd.Series:  
 delta = series.diff()  
 gain = np.where(delta > 0, delta, 0.0)  
 loss = np.where(delta < 0, -delta, 0.0)  
 roll\_up = pd.Series(gain).rolling(period).mean()  
 roll\_down = pd.Series(loss).rolling(period).mean()  
 rs = roll\_up / (roll\_down.replace(0, np.nan))  
 rsi\_val = 100.0 - (100.0 / (1.0 + rs))  
 return pd.Series(rsi\_val, index=series.index)  
  
def six\_month\_drawdown(close: pd.Series) -> float:  
 if len(close) < 126:  
 return np.nan  
 recent = close.iloc[-1]  
 back\_6m = close.iloc[-126]  
 return (recent - back\_6m) / back\_6m  
  
def percentile\_rank(series: pd.Series, value: float) -> float:  
 if series.dropna().empty or pd.isna(value):  
 return np.nan  
 return (series < value).mean()  
  
def compute\_for\_ticker(ticker: str) -> pd.DataFrame:  
 with ENGINE.begin() as conn:  
 px = pd.read\_sql(sa.text("SELECT date, adj\_close, volume FROM prices WHERE ticker=:t ORDER BY date"), conn, params={"t": ticker})  
 fq = pd.read\_sql(sa.text("SELECT \* FROM fundamentals\_q WHERE ticker=:t ORDER BY period\_end"), conn, params={"t": ticker})  
 if px.empty or fq.empty:  
 return pd.DataFrame()  
  
 px["adj\_close"] = pd.to\_numeric(px["adj\_close"], errors="coerce")  
 px["volume"] = pd.to\_numeric(px["volume"], errors="coerce")  
  
 # Compute RSI and 6M drawdown  
 px["rsi14"] = rsi(px["adj\_close"], 14)  
 dd6m = six\_month\_drawdown(px["adj\_close"])  
  
 # Volume trend (simple): pct change of 10-day avg vs prior 10-day avg  
 px["vol\_ma10"] = px["volume"].rolling(10).mean()  
 vol\_trend = (px["vol\_ma10"].iloc[-1] - px["vol\_ma10"].iloc[-11]) / px["vol\_ma10"].iloc[-11] if len(px) > 20 and px["vol\_ma10"].iloc[-11] else np.nan  
  
 # Simple TTM approximations (placeholder)  
 fq = fq.sort\_values("period\_end").tail(5) # last 5 quarters as proxy for TTM approximations  
 rev\_ttm = fq["revenue"].astype(float).tail(4).sum()  
 ni\_ttm = fq["net\_income"].astype(float).tail(4).sum()  
 ocf\_ttm = fq["operating\_cf"].astype(float).tail(4).sum()  
 capex\_ttm = fq["capex"].astype(float).tail(4).sum()  
 fcf\_ttm = ocf\_ttm - capex\_ttm  
 avg\_equity = fq["total\_equity"].astype(float).tail(4).mean()  
 roe = ni\_ttm / avg\_equity if avg\_equity not in (0, None, np.nan) else np.nan  
  
 # Net Debt / EBITDA (approx)  
 latest = fq.tail(1).iloc[0]  
 net\_debt = float(latest.get("total\_debt") or 0) - float(latest.get("cash\_and\_equivalents") or 0)  
 ebitda\_approx = ni\_ttm + 0 # placeholder — ideally add interest+tax+DA if available  
 nde = (net\_debt / ebitda\_approx) if ebitda\_approx else np.nan  
  
 # PE / PS (requires shares outstanding)  
 last\_price = px["adj\_close"].iloc[-1]  
 shares = float(latest.get("shares\_outstanding") or np.nan)  
 eps\_ttm = ni\_ttm / shares if shares else np.nan  
 sales\_ps = rev\_ttm / shares if shares else np.nan  
 pe\_ttm = (last\_price / eps\_ttm) if eps\_ttm else np.nan  
 ps\_ttm = (last\_price / (sales\_ps / 1.0)) if sales\_ps else np.nan  
  
 # Historical percentiles over 5y — placeholder  
 window\_5y = px.tail(252\*5).copy()  
 if not window\_5y.empty and shares:  
 window\_5y["pe\_proxy"] = window\_5y["adj\_close"] / eps\_ttm if eps\_ttm else np.nan  
 window\_5y["ps\_proxy"] = window\_5y["adj\_close"] / (sales\_ps / 1.0) if sales\_ps else np.nan  
 pe\_pct = percentile\_rank(window\_5y["pe\_proxy"].dropna(), pe\_ttm) if eps\_ttm else np.nan  
 ps\_pct = percentile\_rank(window\_5y["ps\_proxy"].dropna(), ps\_ttm) if sales\_ps else np.nan  
 else:  
 pe\_pct = np.nan  
 ps\_pct = np.nan  
  
 # Traffic lights  
 traffic\_f = "GREEN" if (roe is not np.nan and roe > 0.10) and (fcf\_ttm > 0) and (nde is not np.nan and nde < 3) else "RED"  
 traffic\_v = "GREEN" if (pe\_pct is not np.nan and pe\_pct <= 0.20) or (ps\_pct is not np.nan and ps\_pct <= 0.20) else "RED"  
 traffic\_t = "GREEN" if (dd6m is not np.nan and dd6m <= -0.40) and (px["rsi14"].iloc[-1] < 35) and (vol\_trend is not np.nan and vol\_trend > 0) else "RED"  
  
 snap = pd.DataFrame([{  
 "ticker": ticker,  
 "snapshot\_date": pd.Timestamp.today().date(),  
 "roe": roe,  
 "fcf": fcf\_ttm,  
 "net\_debt\_to\_ebitda": nde,  
 "pe\_ttm": pe\_ttm,  
 "pe\_percentile\_5y": pe\_pct,  
 "ps\_ttm": ps\_ttm,  
 "ps\_percentile\_5y": ps\_pct,  
 "dd\_6m": dd6m,  
 "rsi\_14": float(px["rsi14"].iloc[-1]) if not pd.isna(px["rsi14"].iloc[-1]) else np.nan,  
 "vol\_trend\_10d": vol\_trend,  
 "traffic\_fundamental": traffic\_f,  
 "traffic\_valuation": traffic\_v,  
 "traffic\_technical": traffic\_t,  
 "notes": "Educational skeleton snapshot"  
 }])  
 return snap  
  
def write\_snapshot(df: pd.DataFrame):  
 if df.empty:  
 return  
 with ENGINE.begin() as conn:  
 df.to\_sql("factor\_snapshot", conn, if\_exists="append", index=False)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 tickers = sys.argv[1:] or ["META","AAPL","TSLA","SOFI","SAIA","UNH"]  
 out\_list = []  
 for tk in tickers:  
 try:  
 snap = compute\_for\_ticker(tk)  
 write\_snapshot(snap)  
 print(f"[OK] {tk} snapshot rows: {len(snap)}")  
 except Exception as e:  
 print(f"[ERROR] {tk}: {e}")