"""

module\_01.py 📘 1단원 – 기본 정보 분석

==================================================

“종목이 무엇인지 몰라도 되는 AI 중심 구조” + “전략 전체의 출발점이자 핵심 피처 생성기”

주요 기능

---------

1. 종목·티커 식별 ➜ identify\_ticker()

2. 메타 데이터 수집 ➜ fetch\_meta\_info()

3. 특성·라벨 분류 ➜ classify\_stock()

4. 자연어 해석 생성 ➜ generate\_explanation()

5. 통합 분석 ➜ analyze\_stock\_basic()

외부 모듈 연계

-------------

- module\_02 ~ module\_24: Feature Vector 그대로 전달

- app.py(Streamlit): analyze\_stock\_basic() 호출 후 결과 시각화

"""

from \_\_future\_\_ import annotations

import os

import re

import json

import math

import time

import datetime as dt

from dataclasses import dataclass, asdict

from functools import lru\_cache

from typing import List, Dict, Any, Tuple

import requests

import numpy as np

import pandas as pd

import yfinance as yf

from tqdm.auto import tqdm

from sklearn.preprocessing import quantile\_transform

##############################################

# 환경 설정 & 로깅

##############################################

OPEN\_DART\_API\_KEY = (

os.getenv("DART\_API\_KEY")

or os.getenv("OPEN\_DART\_API\_KEY")

or os.environ.get("DART\_API\_KEY")

)

CACHE\_DIR = os.getenv("MSA\_CACHE\_DIR", ".msa\_cache")

os.makedirs(CACHE\_DIR, exist\_ok=True)

def \_log(msg: str) -> None:

now = dt.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

print(f"[module\_01 | {now}] {msg}")

##############################################

# 보조 유틸

##############################################

\_KRX\_TICKER\_URL = (

"https://kind.krx.co.kr/corpgeneral/corpList.do?method=download&searchType=13"

) # 한글 종목명 <-> 티커 매핑용

@lru\_cache(maxsize=1)

def \_load\_krx\_ticker\_table() -> pd.DataFrame:

"""KRX 종목 코드표 로딩 (캐싱)"""

path = os.path.join(CACHE\_DIR, "krx\_tickers.csv")

if os.path.exists(path) and (time.time() - os.path.getmtime(path) < 86400):

return pd.read\_csv(path, dtype=str)

\_log("Downloading KRX ticker table …")

df = pd.read\_html(\_KRX\_TICKER\_URL, header=0)[0]

df["종목코드"] = df["종목코드"].str.zfill(6)

df.to\_csv(path, index=False, encoding="utf-8-sig")

return df

def \_to\_ticker(user\_input: str) -> str | None:

"""한국·해외 상장 여부를 가리지 않고 티커(string)를 리턴. 실패 시 None."""

s = str(user\_input).strip()

# 1) 숫자 6자리 → KRX

if re.fullmatch(r"\d{6}", s):

return s

# 2) 영문 티커 그대로 시도

if re.fullmatch(r"[A-Za-z\-\.]{1,10}", s):

return s.upper()

# 3) 한글 종목명 매핑

df = \_load\_krx\_ticker\_table()

hit = df[df["회사명"] == s]

return hit.iloc[0]["종목코드"] if not hit.empty else None

def \_percentile\_rank(series: pd.Series, value: float) -> float:

"""series 내 value의 퍼센타일(0~100)를 반환"""

if series.empty or pd.isna(value):

return np.nan

rank = (series < value).mean() \* 100

return round(float(rank), 1)

##############################################

# 데이터 클래스

##############################################

@dataclass

class BasicFeatureVector:

ticker: str

name: str

listing\_date: str

market: str # KOSPI / KOSDAQ / NYSE / NASDAQ / ETF …

market\_cap: float # 원화 환산

market\_cap\_pct: float # 업종 내 시총 Percentile

liquidity\_grade: str # 대형/중형/소형

industry: str # 세분화 산업

style\_tags: List[str] # 성장/가치/배당/모멘텀/테마

inst\_preferred: bool

vol\_score: float # 0~1

relative\_metrics: Dict[str, float] # PER/ROE 등 Percentile

beginner\_summary: str

expert\_summary: str

def as\_dict(self) -> Dict[str, Any]:

return asdict(self)

##############################################

# 1) 종목·티커 식별

##############################################

def identify\_ticker(user\_input: str) -> str:

"""

사용자 입력(종목명/티커)을 표준화된 티커로 변환.

실패 시 ValueError.

"""

ticker = \_to\_ticker(user\_input)

if ticker is None:

raise ValueError(f"'{user\_input}'(으)로 티커를 식별할 수 없습니다.")

return ticker

##############################################

# 2) 메타 정보 수집

##############################################

def \_krx\_add\_suffix(ticker: str) -> str:

"""KRX 티커를 yfinance 형식(.KS / .KQ)으로 변환."""

df = \_load\_krx\_ticker\_table()

row = df[df["종목코드"] == ticker]

if row.empty:

return ticker

mkt = row.iloc[0]["시장구분"]

return ticker + (".KS" if mkt == "KOSPI" else ".KQ")

def \_fetch\_yf\_info(ticker: str) -> Tuple[dict, pd.DataFrame]:

"""yfinance 정보 + 최근 주가 DataFrame 리턴 (단일 티커용)"""

try:

yf\_ticker = ticker

# 국내 주식이면 접미사 붙이기

if re.fullmatch(r"\d{6}", ticker):

yf\_ticker = \_krx\_add\_suffix(ticker)

yf\_obj = yf.Ticker(yf\_ticker)

info = yf\_obj.info or {}

hist = yf\_obj.history(period="1y")

return info, hist

except Exception as e:

\_log(f"yfinance 오류: {e}")

return {}, pd.DataFrame()

@lru\_cache(maxsize=128)

def fetch\_meta\_info(ticker: str) -> Dict[str, Any]:

"""

시가총액, 상장일, 시장, 거래량 등 메타 정보 dict 반환.

- 시가총액: 원화 환산 (환율은 yfinance 'currency' 필드 사용; 미지정 시 원 단위 가정)

- listing\_date: yyyy-mm-dd

"""

info, hist = \_fetch\_yf\_info(ticker)

name = info.get("longName") or info.get("shortName") or info.get("symbol") or ticker

currency = info.get("currency", "KRW")

market\_cap\_local = info.get("marketCap", np.nan)

# 환율 보정 (단순, 실시간 필요시 환율 API 사용)

fx\_rates = {"USD": 1400, "EUR": 1500, "JPY": 9.0, "KRW": 1.0}

market\_cap\_krw = float(market\_cap\_local) \* fx\_rates.get(currency, 1.0)

market = info.get("exchange") or info.get("market", "KOSPI" if re.fullmatch(r"\d{6}", ticker) else "UNKNOWN")

listing\_date = info.get("ipoYear") or info.get("listing\_date") or ""

if listing\_date and isinstance(listing\_date, int):

listing\_date = f"{listing\_date}-01-01"

avg\_volume = info.get("averageVolume") or info.get("volume")

liquidity\_grade = (

"대형주" if market\_cap\_krw > 5e13 else

"중형주" if market\_cap\_krw > 1e13 else

"소형주"

)

# 변동성 스코어 (1y 표준편차 / 평균)

vol\_score = 0.0

if not hist.empty:

vol\_score = float(hist["Close"].pct\_change().std() \* math.sqrt(252))

vol\_score = round(min(max(vol\_score, 0), 1), 3)

meta = {

"ticker": ticker,

"name": name,

"listing\_date": listing\_date,

"market": market,

"market\_cap": market\_cap\_krw,

"avg\_volume": avg\_volume,

"liquidity\_grade": liquidity\_grade,

"vol\_score": vol\_score,

"currency": currency,

}

return meta

##############################################

# 3) 특성 분류 & 라벨링

##############################################

# 산업 분류 매핑 예시 (필요시 확장)

\_INDUSTRY\_KEYWORDS = {

"반도체": ["Semiconductor", "Semiconductors", "반도체"],

"메모리 반도체": ["Memory", "DRAM", "NAND"],

"스마트폰": ["Smartphone", "Mobile", "Wireless"],

"바이오": ["Biotechnology", "바이오", "제약"],

"전기차": ["EV", "Electric Vehicle"],

"재생에너지": ["Renewable", "Solar", "Wind"],

}

def \_classify\_industry(info: dict) -> str:

text = " ".join(str(v) for v in info.values()).lower()

for label, keys in \_INDUSTRY\_KEYWORDS.items():

if any(k.lower() in text for k in keys):

return label

return info.get("industry") or info.get("sector") or "기타"

def \_style\_tags(info: dict, hist: pd.DataFrame) -> List[str]:

"""성장/가치/배당/모멘텀/테마 등 성향 판정"""

tags = []

pe = info.get("trailingPE") or info.get("forwardPE") or np.nan

div\_yield = info.get("dividendYield") or 0

revenue\_growth = info.get("revenueGrowth") or 0

# 성장주

if revenue\_growth and revenue\_growth > 0.15:

tags.append("성장")

# 가치주

if not pd.isna(pe) and pe < 10:

tags.append("가치")

# 배당주

if div\_yield and div\_yield > 0.03:

tags.append("배당")

# 모멘텀

if not hist.empty and (hist["Close"].iloc[-1] / hist["Close"].iloc[0] > 1.2):

tags.append("모멘텀")

# 테마 (키워드 기반)

if any(k in info.get("longBusinessSummary", "").lower() for k in ["ai", "metaverse", "ev"]):

tags.append("테마")

return tags or ["일반"]

def \_instit\_preference(hist: pd.DataFrame) -> bool:

"""기관 선호 여부: 거래량 상위 + 상승 추세 여부 간단 판정"""

if hist.empty:

return False

price\_ratio = hist["Close"].iloc[-1] / hist["Close"].iloc[-60]

vol\_ratio = hist["Volume"].iloc[-60:].mean() / hist["Volume"].mean()

return (price\_ratio > 1.05) and (vol\_ratio > 1.2)

def \_relative\_metric\_percentiles(ticker: str, meta: Dict[str, Any], info: dict) -> Dict[str, float]:

"""

동일 업종 내에서 시가총액/PER/ROE 등의 퍼센타일 계산.

간단 구현: yfinance 산업 keyword 이용하여 peer set 추출 후 Percentile 산출.

"""

ind = \_classify\_industry(info)

symbol\_list = [] # peer 티커 모음 (동일 산업)

peer\_field = "industry"

try:

# 검색: 동일 인더스트리에 해당되는 전 세계 주요 티커 50개 최대

# (yfinance 'tickers' 엔드포인트 없음 → 임시로 KRX 테이블 활용)

df = \_load\_krx\_ticker\_table()

symbol\_list = df[df["업종"] == ind]["종목코드"].tolist()[:50]

except Exception:

pass

if ticker not in symbol\_list:

symbol\_list.append(ticker)

m\_caps, pes, roes = [], [], []

for tkr in symbol\_list:

inf, \_ = \_fetch\_yf\_info(tkr)

m\_caps.append(inf.get("marketCap", np.nan))

pes.append(inf.get("trailingPE") or np.nan)

roes.append(inf.get("returnOnEquity") or np.nan)

rel = {

"시총": \_percentile\_rank(pd.Series(m\_caps), meta["market\_cap"]),

"PER": \_percentile\_rank(pd.Series(pes), info.get("trailingPE") or np.nan),

"ROE": \_percentile\_rank(pd.Series(roes), info.get("returnOnEquity") or np.nan),

}

return rel

def classify\_stock(ticker: str) -> Dict[str, Any]:

"""

산업군, 성향 태그, 기관 선호 여부, 변동성 스코어 등 특성 정보 dict 반환

"""

info, hist = \_fetch\_yf\_info(ticker)

meta = fetch\_meta\_info(ticker)

industry = \_classify\_industry(info)

tags = \_style\_tags(info, hist)

inst\_pref = \_instit\_preference(hist)

rel\_metrics = \_relative\_metric\_percentiles(ticker, meta, info)

return {

"industry": industry,

"style\_tags": tags,

"inst\_preferred": inst\_pref,

"relative\_metrics": rel\_metrics,

}

##############################################

# 4) 자연어 해석 생성

##############################################

def \_template\_beginner(meta: Dict[str, Any], cls: Dict[str, Any]) -> str:

tmpl = (

"{name}는 {market}에 상장된 {industry} 분야 기업으로, "

"시가총액은 약 {cap:,.0f}원(국내 시장 기준 상위 {cap\_pct:.1f}%)입니다. "

"최근 1년간 변동성은 {vol:.2f} 수준이며, "

"{tags} 성향으로 분류됩니다."

)

return tmpl.format(

name=meta["name"],

market=meta["market"],

industry=cls["industry"],

cap=meta["market\_cap"],

cap\_pct=meta.get("market\_cap\_pct", np.nan),

vol=meta["vol\_score"],

tags="/".join(cls["style\_tags"]),

)

def \_template\_expert(meta: Dict[str, Any], cls: Dict[str, Any]) -> str:

r = cls["relative\_metrics"]

tmpl = (

"• 업종 대비 시가총액 Percentile {cap\_pct:.1f}%, PER {per\_pct:.1f}%, ROE {roe\_pct:.1f}%\n"

"• 기관 선호도: {inst}\n"

"• 유동성 등급: {liq}\n"

"• 변동성 스코어(1y): {vol:.3f}"

)

return tmpl.format(

cap\_pct=r.get("시총", np.nan),

per\_pct=r.get("PER", np.nan),

roe\_pct=r.get("ROE", np.nan),

inst="높음" if cls["inst\_preferred"] else "보통",

liq=meta["liquidity\_grade"],

vol=meta["vol\_score"],

)

def generate\_explanation(meta: Dict[str, Any], cls: Dict[str, Any]) -> Tuple[str, str]:

"""

초보자용·전문가용 설명 텍스트 반환. (rule-based 템플릿 + 수치 삽입)

"""

beginner = \_template\_beginner(meta, cls)

expert = \_template\_expert(meta, cls)

return beginner, expert

##############################################

# 5) 종합 분석 실행

##############################################

def \_attach\_percentile\_to\_meta(meta: Dict[str, Any], cls: Dict[str, Any]) -> None:

cap\_pct = cls["relative\_metrics"].get("시총")

meta["market\_cap\_pct"] = cap\_pct

def analyze\_stock\_basic(user\_input: str) -> BasicFeatureVector:

"""

Pipeline 전과정 실행 후 BasicFeatureVector 반환

"""

ticker = identify\_ticker(user\_input)

meta = fetch\_meta\_info(ticker)

cls = classify\_stock(ticker)

\_attach\_percentile\_to\_meta(meta, cls)

beg, exp = generate\_explanation(meta, cls)

fv = BasicFeatureVector(

ticker=ticker,

name=meta["name"],

listing\_date=meta["listing\_date"],

market=meta["market"],

market\_cap=meta["market\_cap"],

market\_cap\_pct=meta["market\_cap\_pct"],

liquidity\_grade=meta["liquidity\_grade"],

industry=cls["industry"],

style\_tags=cls["style\_tags"],

inst\_preferred=cls["inst\_preferred"],

vol\_score=meta["vol\_score"],

relative\_metrics=cls["relative\_metrics"],

beginner\_summary=beg,

expert\_summary=exp,

)

return fv

##############################################

# 6) 테스트 & CLI

##############################################

if \_\_name\_\_ == "\_\_main\_\_":

import argparse

parser = argparse.ArgumentParser(description="1단원 기본 정보 분석 모듈 테스트")

parser.add\_argument("query", help="종목명 또는 티커 (예: 삼성전자 또는 005930)")

args = parser.parse\_args()

try:

result = analyze\_stock\_basic(args.query)

print(json.dumps(result.as\_dict(), indent=2, ensure\_ascii=False))

except Exception as e:

\_log(f"오류: {e}")

raise