**🧩 OVERVIEW — HOW TO DESCRIBE IT**

* Built a **full-stack equity valuation and analytics dashboard** in Python replicating the workflow of a professional buy-side or sell-side analyst.
* Integrated **fundamental valuation, technical indicators, and analyst consensus** into one interface to deliver fair value, upside potential, and market context.
* Developed from scratch with a focus on **accuracy, transparency, and usability**, balancing rigorous quantitative methods with clear visual design.
* Entirely modular architecture, enabling each section (valuation, technicals, analyst, forecast) to run independently but feed a unified “recommendation” output.

**⚙️ TECHNICAL & QUANTITATIVE COMPONENTS**

* Implemented a **discounted cash flow (DCF) valuation engine** computing enterprise value, equity value, and price per share from projected free cash flows.
* Built **operating projections** linking revenue → EBITDA → EBIT → NOPAT → FCF, with dynamic CapEx, depreciation, tax, and working-capital flows.
* Added **WACC decomposition** (risk-free rate, market premium, beta) and **terminal value** calculation using both the Gordon Growth and exit-multiple approaches.
* Developed **scenario simulations (Bear, Base, Bull)** that adjust growth, margins, and discount rates to reflect different economic and company conditions.
* Integrated a **sensitivity matrix** showing price per share under multiple WACC and terminal-growth combinations, similar to professional valuation sheets.
* Combined results into a **triangulated fair-value system** comparing intrinsic DCF value, analyst target, and market price to generate buy/hold/sell suggestions.
* Integrated **technical indicators** (RSI, Bollinger Bands, ATR, MACD, moving averages) to reflect market momentum and volatility context.
* Fetched **analyst consensus data** (median target, number of analysts, average rating) to benchmark model accuracy and highlight deviations.
* Automated **data retrieval** through APIs (e.g., yfinance, WRDS, or Finnhub), with validation checks for missing or stale data.
* Engineered **reactive data pipelines** that recompute valuations automatically when parameters or tickers change.
* Implemented **financial-grade number formatting** (billions, no cents), rounding consistency, and clear axis labels for professional readability.
* Used **Streamlit** for the interface and **Plotly** for interactive visualization, ensuring responsiveness and clean layout across screen sizes.

**💰 FINANCIAL THINKING & MARKET LOGIC**

* Designed the workflow to mimic how analysts **triangulate between fundamentals, technicals, and sentiment** before investment decisions.
* Focused on **transparency of assumptions**, displaying all core inputs (WACC, g, exit multiple) alongside results for auditability.
* Built a structure that allows **analyst-style stress-testing**, comparing outputs across multiple economic assumptions in real time.
* Ensured each DCF scenario outputs **enterprise value, equity value, and target price**, making it possible to reconcile with current market capitalization.
* Added **data provenance** (source labels and last-updated timestamps) to increase credibility and facilitate future integration with professional data feeds.
* Incorporated **risk metrics** (Beta, ATR, volatility range) to contextualize valuation upside relative to risk exposure.
* Designed metrics and visual cues (colors, icons, layout) to emulate how portfolio managers read institutional dashboards — quick, clear, actionable.

**💻 SOFTWARE & SYSTEM DESIGN**

* Structured backend as modular Python classes/functions: separate logic for valuation, technical analysis, analyst data, and visualization.
* Encapsulated reusable helper functions (to\_num, fdiv, fsub, format\_billions, etc.) to standardize computations and handle API inconsistencies.
* Implemented a **clean caching mechanism** (via st.session\_state) for faster recalculation and persistence across sessions.
* Designed **export tools** allowing CSV downloads of DCF projections, sensitivity matrices, and valuation comparisons.
* Ensured **robust error handling** and visual warnings for missing data or unrealistic results (e.g., negative WACC, invalid growth assumptions).
* Maintained a **light but professional UI design** using Streamlit CSS blocks (cards, metrics, colors), matching the look of Bloomberg or Refinitiv widgets.

**🎨 UI / UX & PRESENTATION DESIGN**

* Created a **2×2 quadrant layout**:
  + Top left → Stock price chart with Bollinger Bands & ATR.
  + Top right → DCF valuation and metrics.
  + Bottom left → Company info & analyst estimates.
  + Bottom right → Forecast & final recommendation.
* Added icons and color logic for readability (💰 DCF, 📈 Price, 🏢 Company Info, 🔮 Forecast).
* Used consistent **visual hierarchy** (headers, metric cards, legends) to make the dashboard professional yet intuitive.
* Centered metric summaries (EV, Equity Value, PPS) above tables for clear hierarchy.
* Integrated **download buttons, expanders, and hover charts** to increase interactivity.
* Added **explicit units and axis titles** (“DCF Projection (in $B)”, “Fiscal Year”, “Value ($B)”) for publication-grade visuals.
* Designed for **CV / portfolio presentation** — readable screenshots, consistent color scheme, and clear framing of financial metrics.

**🧠 INSIGHT & PURPOSE**

* The goal was not to “visualize data” but to **replicate a real equity-research decision framework** inside a live tool.
* The DCF is used to anchor intrinsic value; technicals show market behavior; analysts provide sentiment and benchmarks — all synthesized into a single signal.
* Demonstrates the ability to **translate theory (valuation models) into automated, scalable computation**.
* Bridges **financial modeling, quantitative logic, and product design**, showing full-stack analytical thinking.

**📄 CV / PORTFOLIO WRITE-UP (short version)**

**Equity Valuation & Analytics Dashboard (Python, Streamlit)**  
• Built an end-to-end stock analysis platform combining discounted cash flow modeling, technical indicators, and analyst consensus data.  
• Implemented a modular DCF engine with multi-scenario simulation, WACC & terminal-growth sensitivity analysis, and automated fair-value computation.  
• Deployed a professional-grade dashboard integrating valuation projections, volatility metrics, and buy/sell signals through Streamlit and Plotly.

**🧭 INTERVIEW TALKING POINTS**

* “The motivation was to bridge theory and practice — I wanted to automate how a research analyst triangulates between fundamentals, technicals, and sentiment.”
* “The backend computes full DCF valuations with varying assumptions, compares them to analyst consensus, and merges technical indicators for signal validation.”
* “I focused on accuracy (±20% target deviation) and clarity — each valuation is transparent, assumptions are visible, and results are exportable.”
* “The project taught me to think like a quant and build like an engineer — understanding valuation math, data structuring, and presentation simultaneously.”
* “It’s not just a model; it’s a framework for testing investment ideas under multiple scenarios with proper sensitivity analysis.”

**🪜 FUTURE EXTENSIONS (to show ambition)**

* Add **Monte Carlo simulations** for uncertainty in revenue growth, margins, and discount rates.
* Integrate **real-time data feeds** and portfolio backtesting to track performance of DCF-implied signals.
* Incorporate **sentiment and news NLP** to enrich the “analyst consensus” view.
* Enable **multi-asset coverage** (ETFs, commodities) with adapted valuation logic.
* Deploy as a **web-accessible application** with authentication, watchlists, and portfolio analytics.

**1) Repository blueprint**

equity-dcf-dashboard/

├─ app/

│ ├─ interface.py # your existing UI

│ ├─ dcf\_calculation.py # improved calc (yours)

│ ├─ dcf\_calibration.py # improved calib (yours)

│ ├─ tech\_indicators.py # indicator helpers (if used)

│ ├─ data\_io.py # yfinance/WRDS/Finnhub wrappers

│ ├─ utils.py # formatting, safe ops

│ └─ assets/

│ ├─ styles.css # optional extra CSS

│ └─ logo.png # optional branding

├─ examples/

│ ├─ demo\_screenshots/

│ │ ├─ main\_aapl.png

│ │ ├─ dcf\_table.png

│ │ └─ forecast\_commentary.png

│ └─ sample\_outputs/

│ ├─ dcf\_projection\_aapl.csv

│ └─ sensitivity\_grid\_aapl.csv

├─ tests/

│ └─ test\_dcf\_sanity.py

├─ .github/

│ └─ workflows/

│ └─ ci.yml # GitHub Actions: lint + tests

├─ .env.example

├─ .gitignore

├─ LICENSE

├─ Makefile # quick commands (optional but nice)

├─ pyproject.toml # ruff/black config (optional)

├─ README.md # ⭐ your sales page (full below)

├─ requirements.txt

└─ streamlit\_app.py # tiny launcher

**2) README.md (paste-ready, detailed)**

# Equity DCF Dashboard

A \*\*Python/Streamlit\*\* dashboard that triangulates \*\*DCF valuation\*\*, \*\*technical indicators\*\*, and \*\*analyst consensus\*\* to produce investment insights with \*\*transparent assumptions\*\*, \*\*scenario & sensitivity analysis\*\*, and \*\*exportable tables\*\*.

<p align="center">

<img src="examples/demo\_screenshots/main\_aapl.png" width="860" />

</p>

---

## ✨ Highlights

- \*\*DCF engine (professional-grade):\*\*

- Multi-year projections with \*\*decaying revenue growth\*\* and \*\*margin convergence\*\* to industry targets.

- Clean FCF construction: `NOPAT + Depreciation − CapEx − ΔNWC`.

- \*\*Terminal value discipline\*\* (g < WACC − 1.5%), TV discounted once; TV share flagged if high.

- EV → Equity → \*\*Per-Share\*\* outputs, plus audit fields (FCF series, PV series, TV share, growth curve).

- \*\*Calibration module (realistic):\*\*

- \*\*WACC\*\* from country, size, beta, and rating; adaptive clamps by size bucket.

- Country risk premia, size premia, credit spreads, inflation anchors.

- Industry defaults for \*\*capex\*\*, \*\*NWC\*\*, \*\*margin targets\*\*, long-term growth caps.

- \*\*Interface that feels sell-side:\*\*

- Clear quadrants: Price/Technicals, DCF Valuation, Company/Analyst Info, Forecast/Recommendation.

- \*\*DCF Projection table\*\*: fiscal years across columns, metrics down rows; auto-units ($B/$M); CSV export.

- \*\*Analyst Commentary (expandable)\*\* explaining \*\*DCF vs Market\*\* divergences—perfect for viewers.

- \*\*Engineering craft:\*\*

- Modular backend, unit-safe helpers, error handling, and data source tagging.

- CI (GitHub Actions) + minimal tests for credibility.

- Ready to deploy to \*\*Streamlit Community Cloud\*\* or your own host.

---

## 🧭 Quick Start

```bash

# 1) clone & enter

git clone https://github.com/<your-username>/equity-dcf-dashboard.git

cd equity-dcf-dashboard

# 2) create env (choose one)

python -m venv .venv && source .venv/bin/activate # macOS/Linux

# or: conda create -n dcf python=3.11 -y && conda activate dcf

# 3) install deps

pip install -r requirements.txt

# 4) optional: set keys

cp .env.example .env # then edit with your keys if using Finnhub/WRDS

# 5) run

streamlit run streamlit\_app.py

**Default data source**: yfinance (for prices & basic info).  
Optionally plug in **Finnhub/WRDS** by adding keys in .env and enabling in app/data\_io.py.

**🧩 Project Structure**

app/

interface.py # UI (Streamlit layout & cards)

dcf\_calculation.py # cash flows, PVs, TV, EV→Equity, PPS; audit fields

dcf\_calibration.py # WACC + terminal growth calibration; industry/country/size aware

tech\_indicators.py # Bollinger Bands, ATR, MAs (used by price chart)

data\_io.py # clean fetchers & source tags

utils.py # formatting helpers ($B/$M), safe math, clamps

examples/

demo\_screenshots/ # images used in README and portfolio

sample\_outputs/ # CSV exports from the app (DCF table, etc.)

tests/

test\_dcf\_sanity.py # small but meaningful sanity checks

**🧮 How the DCF works (10-point summary)**

1. **Revenue growth** follows a **decaying curve**: high near-term, fades toward steady-state.
2. **Margins** converge toward **industry targets** using a smooth blend from current → steady.
3. **CapEx** and **ΔNWC** scale with revenue (industry-adjusted), with small cyclicality control.
4. **Depreciation** linked conservatively to CapEx; capped to avoid non-sense vs EBIT.
5. **NOPAT** computed as EBIT × (1 − tax\_rate) with a realistic tax floor and region effects.
6. **FCF** = NOPAT + Depreciation − CapEx − ΔNWC (stored per year for auditing).
7. Each FCF is discounted: PV\_FCF\_t = FCF\_t / (1+WACC)^t.
8. **Terminal Value**:
   * Perpetuity: TV = FCF\_final × (1+g)/(WACC−g) with **g < WACC−1.5%** enforced.
   * Exit multiple (optional): TV = EBITDA\_final × multiple.
   * TV discounted **once**: PV\_TV = TV / (1+WACC)^N.
9. **Enterprise Value** = ΣPV\_FCF + PV\_TV.
10. **Equity Value** = EV − Net Debt (+ Cash if not netted) → **Price per Share** using current shares.

**Diagnostics included:** TV\_share, growth\_curve\_used, FCF\_series, PV\_FCF\_series, TV\_method\_used.

**🧭 Why DCF can differ from the market**

Markets price **sentiment**, **liquidity**, **macro cycles**, and **growth optionality** beyond modeled cash flows.  
The interface includes an expandable commentary in the **Forecast/Recommendation** panel explaining this (great for reviewers).

**🔧 Configuration**

* **.env.example**
* FINNHUB\_API\_KEY=YOUR\_KEY
* WRDS\_USER=your\_wrds\_user
* WRDS\_PASS=your\_wrds\_password
* DATA\_PROVIDER=yfinance # yfinance|finnhub|wrds (handled in data\_io.py)
* **Switching data source**: edit app/data\_io.py to read DATA\_PROVIDER and route fetchers.
* **Units**: Internally normalized to **USD millions**; UI auto-formats to **$B/$M**.

**🧪 Tests (sanity, but meaningful)**

Run:

pytest -q

tests/test\_dcf\_sanity.py:

import math

from app.dcf\_calculation import run\_dcf # adapt to your entry function

def test\_tv\_share\_reasonable():

out = run\_dcf("AAPL", years=5)

tv\_share = out.get("TV\_share")

assert tv\_share is not None

assert 0.30 <= tv\_share <= 0.80

def test\_pv\_series\_length():

out = run\_dcf("AAPL", years=5)

pv = out.get("PV\_FCF\_series")

assert isinstance(pv, (list, tuple))

assert len(pv) == 5

def test\_pps\_positive():

out = run\_dcf("AAPL", years=5)

assert out["price\_per\_share"] is None or out["price\_per\_share"] > 0

*(Adjust run\_dcf import to your actual function that returns the valuation dict.)*

**🟩 CI (GitHub Actions)**

.github/workflows/ci.yml

name: CI

on:

push:

pull\_request:

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-python@v5

with:

python-version: '3.11'

- run: python -m pip install --upgrade pip

- run: pip install -r requirements.txt

- run: pip install pytest

- run: pytest -q

**🧰 Developer UX**

* **Makefile** (optional):
* venv:
* python -m venv .venv && . .venv/bin/activate && pip install -r requirements.txt
* run:
* streamlit run streamlit\_app.py
* test:
* pytest -q
* fmt:
* ruff check . --fix || true
* **pyproject.toml** (optional formatting with ruff/black):
* [tool.ruff]
* line-length = 100
* target-version = "py311"
* select = ["E","F","I","UP"]

**🚀 Deployment (Streamlit Cloud)**

1. Push repo to GitHub.
2. Go to **share.streamlit.io** → **New app** → select repo & streamlit\_app.py.
3. Add **secrets** if needed (Settings → Secrets):
4. FINNHUB\_API\_KEY="YOUR\_KEY"
5. DATA\_PROVIDER="yfinance"
6. Deploy. Add the link to your README badge:
7. [![Streamlit App](https://img.shields.io/badge/Streamlit-live-red)](YOUR\_DEPLOY\_URL)

**🧯 Troubleshooting**

* **Valuations all high/low:**
  + Check WACC dispersion (size, beta, country) — the calibration clamps may be too tight.
  + Ensure g < WACC − 1.5% is enforced; TV share should be ≤ ~75%.
* **Per-share looks wrong:**
  + Confirm **shares outstanding** are from the same snapshot as your market cap.
  + Verify unit consistency: internal **USD millions**, UI formats to **$B/$M**.
* **Tables show constant numbers:**
  + Ensure you’re passing the **actual projection DataFrame** into the display block (no placeholder).
  + Confirm formatting is applied **after** computing numeric values.
* **App is too big on screen:**
  + Use the built-in **UI density toggle** (you added in interface.py), or reduce font via CSS block.

**🗺️ Roadmap**

* Add **sensitivity heatmap** (WACC × g) with tooltip showing implied PPS.
* Add **Monte Carlo** on growth/margins/WACC to produce a **distribution of intrinsic values**.
* Backtest: link a **simple rule** (e.g., buy when DCF > price by 20%) and show realized returns.
* Add **news/sentiment** overlay as a 4th “context” signal.
* Export a **one-page PDF** summary per ticker (valuation + commentary).

**📝 License**

MIT — see LICENSE.

**🙌 Acknowledgements**

* Public market data via **yfinance**.
* Streamlit & Plotly for fast, clean UI.

---

# 3) requirements.txt (pin versions)

```txt

streamlit==1.37.1

pandas==2.2.2

numpy==1.26.4

yfinance==0.2.43

plotly==5.22.0

python-dotenv==1.0.1

requests>=2.31

# optional (uncomment if you use them)

# wrds==3.1.6

# finnhub-python==2.4.21

pytest==8.2.0

ruff==0.5.7

**4) .gitignore**

# env & caches

.venv/

\_\_pycache\_\_/

\*.pyc

.DS\_Store

# notebooks/checkpoints

.ipynb\_checkpoints/

# secrets

.env

.streamlit/

secrets.toml

**5) .env.example**

FINNHUB\_API\_KEY=

WRDS\_USER=

WRDS\_PASS=

DATA\_PROVIDER=yfinance

**6) LICENSE (MIT)**

MIT License

Copyright (c) 2025 <Your Name>

Permission is hereby granted, free of charge, to any person obtaining a copy

...

(standard MIT text — keep as-is)

**7) streamlit\_app.py (launcher)**

import os

import streamlit as st

# Optionally load .env for local dev

try:

from dotenv import load\_dotenv

load\_dotenv()

except Exception:

pass

st.set\_page\_config(page\_title="Equity DCF Dashboard", page\_icon="📈", layout="wide")

# Import your app entry

from app.interface import main as run\_app # adjust if your entry name differs

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

run\_app()

*(If your interface.py exposes main(), this works out-of-the-box; if not, export a main() wrapper there.)*

**8) tests/test\_dcf\_sanity.py**

(Already shown in README; include that file verbatim.)

**9) GitHub push steps (quick)**

cd ~/Desktop

mkdir equity-dcf-dashboard && cd equity-dcf-dashboard

# place all files/folders above here

git init

git add .

git commit -m "feat: initial release of equity DCF dashboard"

git branch -M main

git remote add origin git@github.com:<your-username>/equity-dcf-dashboard.git

git push -u origin main

**10) Screenshots to include**

* examples/demo\_screenshots/main\_aapl.png: full 2×2 layout with compact fonts.
* examples/demo\_screenshots/dcf\_table.png: the formatted DCF table (years across columns).
* examples/demo\_screenshots/forecast\_commentary.png: the Recommendation quadrant with the **DCF vs Market** expander visible.