

**can we do a tree and code workflow to give to my programer. I dont know how all this are implemented**

Yes, a **workflow tree** and annotated code workflow diagram can be provided for implementing your financial data visualization suite. Below is an organized **workflow tree** followed by a step-by-step **code workflow** that you can share with your programmer, showing how data flows and how chart scripts interface with the raw CSV files and sector logic.[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7)[[8]](#fn8)[[9]](#fn9)

**Financial Data Visualization Workflow Tree**

1. Data Ingestion  
 └─ Load main financial CSVs  
 ├─ basic\_calculation\_daily\_0\_20250829.csv  
 └─ tradingview\_universe.csv  
2. Data Preparation  
 ├─ Define sector/ETF groupings  
 ├─ Create derived columns (market cap category, signals)  
 └─ Filter/select relevant tickers and indicators  
3. Analysis and Aggregation  
 ├─ Sector performance (avg returns, volatility, RSI)  
 ├─ Top/bottom performers analysis  
 ├─ Market cap return distribution (group by cap category)  
 ├─ Technical signal extraction (RSI/MACD/momentum)  
 └─ Correlation matrix construction  
4. Visualization Script Execution  
 ├─ chart\_script.py (Price vs Year Change Scatter Plot)  
 ├─ chart\_script\_1.py (Correlation Heatmap)  
 ├─ chart\_script\_2.py (Sector Performance Bubble Chart)  
 ├─ chart\_script\_3.py (RSI vs Momentum Scatter)  
 ├─ chart\_script\_4.py (Market Cap Violin Plot)  
 ├─ chart\_script\_5.py (Technical Analysis Radar Chart)  
 └─ chart\_script\_6.py (Risk vs Return Scatter Plot)  
5. Output Generation  
 ├─ Export figures as .png or .jpg  
 ├─ Optionally integrate charts into dashboard/web (Plotly/FastAPI/Flask/Dash)  
 └─ Save analysis tables as .csv/.json as needed

**Annotated Code Workflow Step-by-Step**

**1. Data Ingestion**

* **Read main financial CSV**

df\_calc = pd.read\_csv('basic\_calculation\_daily\_0\_20250829.csv')  
df\_univ = pd.read\_csv('tradingview\_universe.csv')

**2. Data Preparation**

* **Define sectors/ETF tickers**
  + Use hardcoded ticker lists per sector (see script\_1.py logic for reference).
* **Engineer columns**
  + Example: market cap via price bins, as in script\_3.py:

df\_calc['MarketCap\_Category'] = pd.cut(df\_calc['current\_price'], bins=[0, 100, 300, 600, float('inf')], labels=['Small Cap', 'Mid Cap', 'Large Cap', 'Mega Cap'])

* + Extract technical indicators (RSI, MACD, ATR, etc).

**3. Analysis & Aggregation**

* **Sector Performance**
  + Loop through sector tickers, aggregate average metrics.
* **Top/Bottom Performers**
  + Use nlargest() and nsmallest() methods.
* **Signal Extraction**
  + Categorize RSI, MACD (see script\_4.py).
* **Correlation Matrix**
  + Use df.corr() on selected columns.

**4. Visualization**

* **Run chart scripts**
  + Each script imports pandas/plotly, loads the prepped data, and builds a figure, e.g.:

fig = go.Figure()  
# Add traces, update layout  
fig.write\_image('filename.png')

* + Scripts include:
    - price\_vs\_year\_change\_scatter.py
    - correlation\_heatmap.py
    - sector\_performance\_bubble\_chart.py
    - rsi\_momentum\_scatter.py
    - violin\_plot\_returns.py
    - radar\_chart.py
    - risk\_return\_scatter.py

**5. Output**

* **Save charts**
  + Each chart is saved as PNG (or SVG/JPG if needed) via Plotly's write\_image() function.
* **Export summary tables or intermediate datasets** as needed for additional tools or dashboard integration.

**Diagram Example for Your Programmer**

CSV Data ──► DataFrame (pandas)  
 │  
 ├─► Sector & Ticker Selection (script\_1.py)  
 │  
 ├─► Technical Signal Engineering (script\_4.py)  
 │  
 └─► Analysis (top/bottom, correlation, grouping)  
 │  
 └─► Run Chart Scripts  
 ├─► creates chart .png/.svg files  
 └─► Dashboard or reporting output

**Notes for Implementation**

* **Python libraries required:**  
  pandas, numpy, plotly (and optionally dash, flask, fastapi for dashboards).
* **Data input:**  
  Place CSV files in project directory.
* **Chart scripts:**  
  Run independently or in batch, each script expects data columns as shown above.

This workflow tree and annotated code structure allows your programmer to understand the architecture needed for automating, batching, or deploying the full financial visualization suite using your attached scripts and files.[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7)[[8]](#fn8)[[9]](#fn9)[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)

⁂

1. [script.py](http://script.py)

1. script\_1.py

1. chart\_script.py

1. chart\_script\_1.py

1. chart\_script\_2.py

1. chart\_script\_3.py

1. chart\_script\_4.py

1. chart\_script\_5.py

1. chart\_script\_6.py

1. script\_2.py

1. script\_3.py

1. script\_4.py

1. Comprehensive-Financial-Data-Visualization-Analysi.docx