Project Documentation Roadmap: Stock Price Predictor

1. Project Planning & Management

- **Project Overview:**
  - **Objective:** Develop a multi-horizon stock forecasting tool using Deep Learning and provide qualitative financial analysis using Generative AI.
  - **AI Problem Type:** * **Deep Learning (DL):** Time-series forecasting (Regression) using LSTM.
    - **Generative AI (GenAI):** Financial advisory and text generation using LLMs (Gemini).
- **Key Performance Indicators (KPIs):**
  - **Regression Metrics:** Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE).
  - **GenAI Metrics:** Latency (time to generate advice), Relevance (qualitative assessment of advice against market trends).
- **Risk Assessment:**
  - **Market Risk:** Disclaimer that AI predictions are not financial guarantees.
  - **Hallucination Risk:** Gemini providing incorrect financial facts or "making up" news.
  - **Data Latency:** Reliance on `yfinance` (delayed data vs. real-time).

2. Requirements Analysis

Define what the system does based on your `app.py` and `stock_predictor.py`.

- **Functional Requirements:**
  - **Data Ingestion:** Fetch historical stock data via Yahoo Finance (`yfinance`) for a user-selected ticker.
  - **Forecasting:** Predict stock prices for 4 specific horizons: 1, 30, 90, and 180 days.
  - **Advisory:** Generate textual investment advice based on current price vs. predicted price using Google Gemini 2.5 Flash.

    - **Visualization:** Interactive plotting of training loss, historical prices, and prediction comparisons using Plotly.
- **Non-Functional Requirements:**
    - **Reproducibility:** Ensuring consistent model training results via seed setting (`set_seeds(42)`).
    - **Usability:** Simple web interface via Streamlit.
    - **Portability:** Dockerized deployment capabilities.

3. Data Management & Preparation

Document how `stock_predictor.py` handles data.

- **Data Sources:**
    - **Primary Source:** Yahoo Finance API (`yfinance`).
    - **Data Type:** Public historical market data (Open, High, Low, Close, Volume).
- **Data Preprocessing Pipeline:**
    - **Transformation:** Logarithmic transformation of Close prices (`np.log`).
    - **Feature Engineering:** Calculation of returns (`diff()`).
    - **Normalization:** `StandardScaler` applied to returns to achieve zero mean and unit variance.
    - **Sequence Generation:** Sliding window approach (`_build_supervised`) to create `(Past, Future)` pairs.
        - *Lookback Period:* 60 days (default).
        - *Target:* Sum of returns for specific horizons.
- **Data Splitting:**
    - **Method:** Chronological split (Time Series Split).
    - **Ratio:** 80% Training, 20% Testing (implied by `train_ratio=0.8`).

4. System Design & Modeling

Detail the architecture found in `stock_predictor.py` and `gemini_advisor.py`.

4.1. Deep Learning Model (LSTM)

- **Architecture:**
  - Input Layer: Shape `(Lookback, 1)`
  - Hidden Layer 1: LSTM (64 units, `return_sequences=True`)
  - Dropout Layer: 0.2 rate
  - Hidden Layer 2: LSTM (64 units)
  - Dropout Layer: 0.2 rate
  - Dense Layer: 64 units (Linear activation)
  - Output Layer: 4 units (Linear activation, one for each horizon).
- **Loss Function:** Custom Asymmetric Pinball Loss (`pinball_loss`) with `tau=0.6` (penalizes underestimation differently than overestimation).
- **Optimizer:** Adam (`learning_rate=1e-3`).

4.2. Generative AI (Gemini)

- **Model:** `gemini-2.5-flash`.
- **Integration:** `google-genai` SDK.
- **Prompt Engineering:**
  - **Context:** Acts as an "expert financial advisor".
  - **Inputs:** Ticker, Current Price, Predicted Prices (1d, 30d, 90d, 180d), History Summary.
  - **Outputs:** Short-term outlook, Long-term strategy, Risk factors, Buy/Sell/Hold recommendation.

5. Implementation Details

Document the tech stack and reproducibility steps.

- **Technology Stack:**
  - **Frontend:** Streamlit.
  - **ML Framework:** TensorFlow/Keras.
  - **Containerization:** Docker & Docker Compose.
- **Source Code Structure:**
  - `app.py`: UI entry point.
  - `stock_predictor.py`: Core logic class.
  - `gemini_advisor.py`: GenAI wrapper.
- **Deployment:**
  - Docker Build: `docker-compose up --build`.
  - Port Mapping: 8501.

6. Testing & Evaluation

Document how the model's success is measured.

- **Evaluation Strategy:**
  - **Validation:** 10% validation split during training (`val_split=0.1`).
  - **Bias Correction:** Post-training bias correction using the last 10% of training data residuals.
- **Metrics:**
  - `metrics['mse']`: Mean Squared Error.
  - `metrics['mae']`: Mean Absolute Error.
  - `metrics['mape']`: Mean Absolute Percentage Error (Crucial for financial accuracy).

7. User Manual (Draft)

Instructions for the end-user.

1. **Setup:** Input API Key for Gemini (currently hardcoded for demo, but advise on security).
2. **Configuration:** Select a Ticker (e.g., AAPL) from the sidebar.
3. **Training:** Click "Train Model". Wait for the progress bar.
4. **Analysis:**
   - Review the "Model Performance Metrics" cards.
   - Check the "Multi-Horizon Forecast" tab for future price targets.
5. **Advisory:** Click "Get AI Trading Advice" to generate a Gemini report.
6. **Export:** Use the "Download CSV" button to save predictions.

Action Items for Documentation Completion

To fully satisfy the "AI Project Documentation Guidelines", you should add the following files or sections to your repo:

1. **ARCHITECTURE.md**: A diagram or text description of how the LSTM flows into the Gemini Prompt.
2. **ETHICS_POLICY.md**: A brief statement about AI bias in finance and a disclaimer that the tool is for educational purposes only.
3. **CHANGELOG.md**: Track versions (e.g., "v1.0: Added Gemini 2.5 Flash support").