**CAPSTONE PROJECT**

**Stock Price Detection and Visualization**

**HARIKRISHNAN K - 192124217**

**SHAIK BURAN BASHA –192124214**

**COURSE NAME: “Data Handling and Visualization for Large Dataset”**

**COURSE CODE: DSA0619**

**SUPERVISOR :**

**Dr . RAMESH KUMAR Date of Submission :**

**Department of Next-Gen Computing 25/02/2025**

**SIMATS Engineering**

|  |  |
| --- | --- |
| **Table of Contents:** | |
| 1. **Introduction and Abstraction** | **1** |
| 1. **Objectives** | **2** |
| 1. **Tools and Technologies** | **9** |
| 1. **Program 1: Nvidia Stock Price Prediction**  * **Functionality Overview** * **Key Functions** * **Visualization and Outputs** | **10** |
| 1. **Prediction model** | **13** |
| 1. **Future Development** | **16** |
| 1. **Conclusion** | **16** |

**STOCK PRICE DETECTION AND VISUALIZATION**

**Introduction:**

* Briefly explain the purpose of the project. Highlight how stock prediction and visualization are useful for financial analysis and how this project showcases data fetching, predictive modeling, and visualization using Python.

**Abstract :**

* This project explores stock price prediction and visualization using Python and the Alpha Vantage API. The project is divided into two key components. The first component involves predicting Nvidia’s future stock prices using historical data and linear regression. The data is visualized through interactive plots that highlight trends before and after a reference date, as well as predicted future prices. The second component focuses on visualizing the stock prices and user counts of several AI-driven companies, including META, Microsoft, Google, Amazon, and Tesla. Real-time stock prices are fetched, and the data is presented using bar plots and tree maps for better insights into each company’s user engagement and market performance. This project demonstrates the integration of predictive modeling and interactive visualization, providing valuable insights into stock trends and company performance in the AI sector. Future improvements could include advanced prediction models and real-time analytics to enhance decision-making capabilities.

**Objective:**Build an interactive web dashboard that displays Nvidia stock data. The dashboard fetches data from the Alpha Vantage API and allows users to visualize the stock's performance, view key statistics, and examine detailed metrics for specific dates.

**Key Features:**

* **Dynamic Charting:**Users can choose from multiple chart types (Line, Bar, Candlestick) to view the stock data.
* **Date-Based Highlighting:**Data is segmented and highlighted based on a specific date (e.g., data after January 24, 2025 is highlighted in red).
* **Statistical Summaries:**The dashboard computes and displays statistics such as mean, median, and a simple predicted next closing price.
* **Detailed Daily Metrics:**A dropdown menu lets users select a specific date to view detailed metrics (open, high, low, close) for that day.
* **Modern UI Theme:**The entire dashboard has a sleek, modern design with a black background and contrasting white and red text.

**2. Data Acquisition and Preprocessing**

**A. Data Fetching**

* **API Call:**The project uses the Alpha Vantage API to retrieve daily stock data for Nvidia (NVDA).
* **Implementation:**The fetch\_stock\_data function constructs an HTTP GET request with the required parameters (function type, stock symbol, API key) using the requests library. The JSON response is parsed, and the daily time series data is extracted.

**B. Data Filtering and Sorting**

* **Filtering by Date:**Only data from a specified start date (e.g., January 1, 2024) is used. This ensures that outdated information is excluded.
* **Sorting:**The filtered data is sorted chronologically to maintain a natural time flow, which is essential for accurate charting and analysis.
* **Output:**Two lists are returned: one for the sorted dates and another for their corresponding closing prices.

**C. Prediction Calculation**

* **Simple Predictive Model:**The predict\_next\_day\_price function calculates the next closing price by:
  + Computing the differences between consecutive closing prices.
  + Averaging these differences.
  + Adding the average difference to the most recent closing price.
* **Purpose:**This gives users a rough forecast of what the next closing price might be based on historical trends.

**3. User Interface and Layout Design**

**A. Layout Structure**

* **Overall Theme:**The entire dashboard uses a black background with white text for a modern, high-contrast look. Red is used for highlighting important data.
* **Header:**A large header (using <h1>) displays the project title. It is centrally aligned and styled with padding, a black background, and white text.
* **Control Elements:**
  + **Radio Buttons:**These allow users to select the type of chart they want to view (Line, Bar, or Candlestick). They are rendered with a white text style to be clearly visible.
  + **Dropdown Menu:**A dropdown lists all available dates from the dataset. When a date is selected, detailed stock metrics for that date are displayed.
* **Interactive Graph:**The main graph (dcc.Graph) is used to visualize the stock data interactively. The graph updates dynamically based on user input (chart type selection).
* **Statistics Section:**Additional <p> elements are used to display calculated metrics like mean price, median price, and the predicted next closing price.
* **Data Table:**An HTML table is included to display key metrics such as the ending price and the predicted next price. The table is styled to match the overall theme.

**B. Component Placement**

* The header and control elements (radio buttons and dropdown) are placed at the top for easy access.
* The interactive graph is centrally located.
* Below the graph, statistics and the data table are displayed.
* Finally, detailed metrics for a selected date are shown in a dedicated area beneath the dropdown.

**4. Dash Callbacks and Interactivity**

**A. Graph and Statistics Callback**

* **Trigger:**The callback is triggered when the user changes the selected chart type via the radio buttons.
* **Functionality:**
  1. **Statistical Computations:**Calculates the mean, median, and predicted next price using the processed data.
  2. Data Segmentation:  
     **Splits the data into two groups:**
     + Data before the highlight date (e.g., before January 24, 2025).
     + Data from the highlight date onward.
  3. **Graph Creation:  
     Depending on the selected chart type:**
     + **Line Chart:**Two separate traces are created (one for each data segment) with different colors (blue for before, red for after).
     + **Bar Chart:**Similar segmentation is applied using bar charts.
     + **Candlestick Chart**: A single candlestick chart is generated with an added vertical highlight (using a translucent red rectangle or a dashed red line).
  4. **Layout Updates:**The graph layout is updated with titles, axis labels, and the overall theme (black background, white font).
  5. **Data Table Construction:**An HTML table is constructed to display the ending price and the predicted next price.

**B. Date-Specific Metrics Callback**

* **Trigger:**When the user selects a date from the dropdown menu.
* **Functionality:**
  1. **Data Retrieval:**The callback retrieves the stock data for the selected date from the pre-fetched data.
  2. **Metric Extraction:**Extracts metrics such as the opening price, highest price, lowest price, and closing price.
  3. **Display:**Returns a formatted block of text (styled in red) that displays these metrics, ensuring that important details are highlighted.

**5. Application Execution and Deployment**

**A. Running the Application Locally**

* **Development Server:**The script uses app.run\_server(debug=True) to start the Dash development server. Users can run the script (e.g., python app.py) and then access the application via a local URL (typically http://127.0.0.1:8050).

**B. Deployment Readiness**

* **Flask Server Exposure:**The underlying Flask server is exposed via server = app.server. This makes it straightforward to deploy the app on platforms like Heroku, AWS, or any other cloud hosting service**.**
* **Scaling Considerations:**While the current implementation is ideal for a development environment, further optimization and error handling might be required for production deployments.

**6. Future Enhancements and Potential Extensions**

* **Advanced Forecasting:**Integrate machine learning models to provide more accurate stock price predictions.
* **Additional Metrics:**Include more financial metrics like volume, moving averages, RSI (Relative Strength Index), etc.
* **User Customization:**Allow users to select custom date ranges, compare multiple stocks, or overlay technical indicators on the graphs.
* **Improved UI/UX:**Enhance the styling with more responsive design elements, additional tooltips, and interactive filters.
* **Real-Time Updates:**Implement periodic data refreshes to provide near-real-time stock updates.

**TOOLS and TECHNIQUES:**

**The project utilizes the following tools and techniques to achieve its objectives:**

1. **Programming Language:**
   * Python: The primary language used for data fetching, processing, prediction, and visualization.
2. **APIs:**
   * Alpha Vantage API: Used for fetching both historical and real-time stock prices.
3. **Data Manipulation and Analysis:**
   * **Pandas:** Used for data loading, transformation, and management.
   * **NumPy:** Utilized for numerical operations and date range creation.
4. **Machine Learning:**
   * **Scikit-Learn**: Used to implement linear regression for stock price prediction.
5. **Visualization:**
   * **Plotly**: Used for creating interactive visualizations, including line plots, bar charts, and tree maps.
6. **Data Splitting and Comparison:**
   * Date-based data splitting to compare trends before and after a specified date**.**

**Program 1:**

* **Stock Price Prediction for Nvidia**

Functionality Overview: This program focuses on fetching daily stock prices for Nvidia (NVDA), performing linear regression on the historical data, predicting future stock prices, and visualizing the trends through interactive plots.

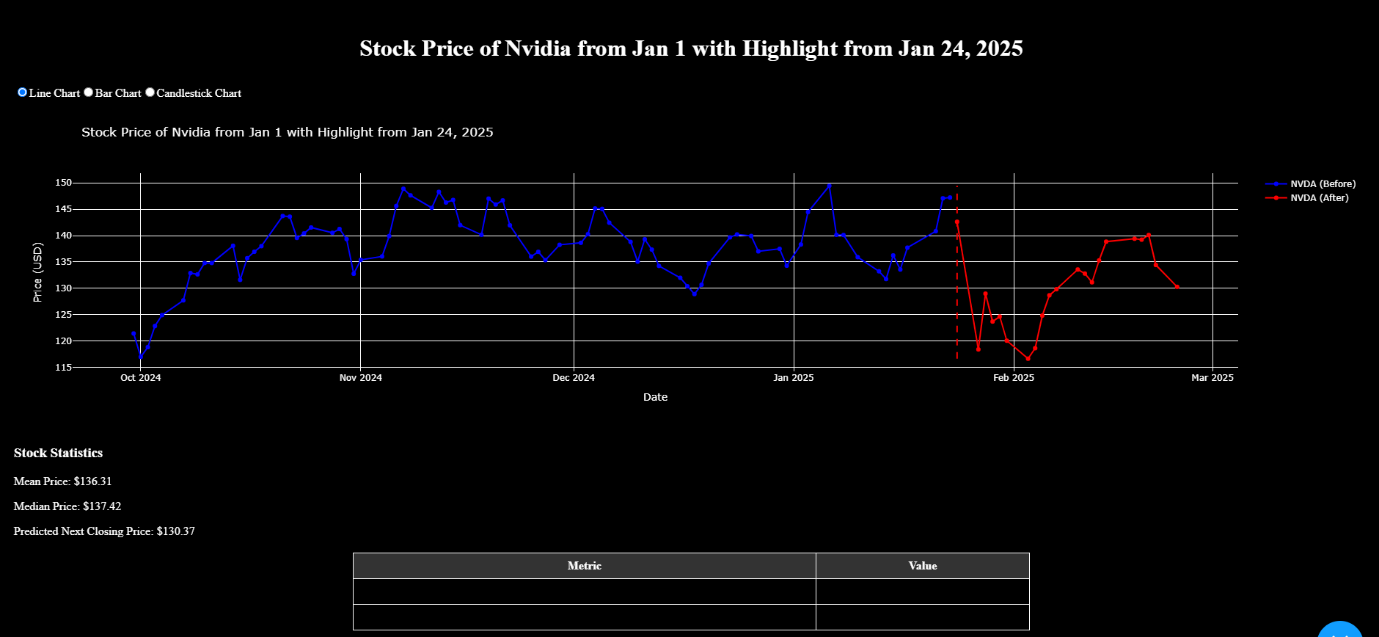
**Key Functions:**

1. **get\_stock\_data(symbol, api\_key):**
   * Fetches daily historical stock prices for a given symbol using the Alpha Vantage API.
   * Transforms the data into a Pandas DataFrame.
2. **predict\_stock\_prices(df):**
   * Uses linear regression to predict stock prices for the next 5 days.
   * Generates future dates corresponding to the predicted prices.
3. **Visualization:**
   * Splits the historical data into two time periods (before and after January 21, 2025).
   * Creates an interactive Plotly chart with separate lines for each time period and predicted prices.

**Code Flow:**

* Fetch Nvidia's stock data.
* Split the data based on a reference date.
* Train a linear regression model and predict future prices.

**LINE PLOT:**



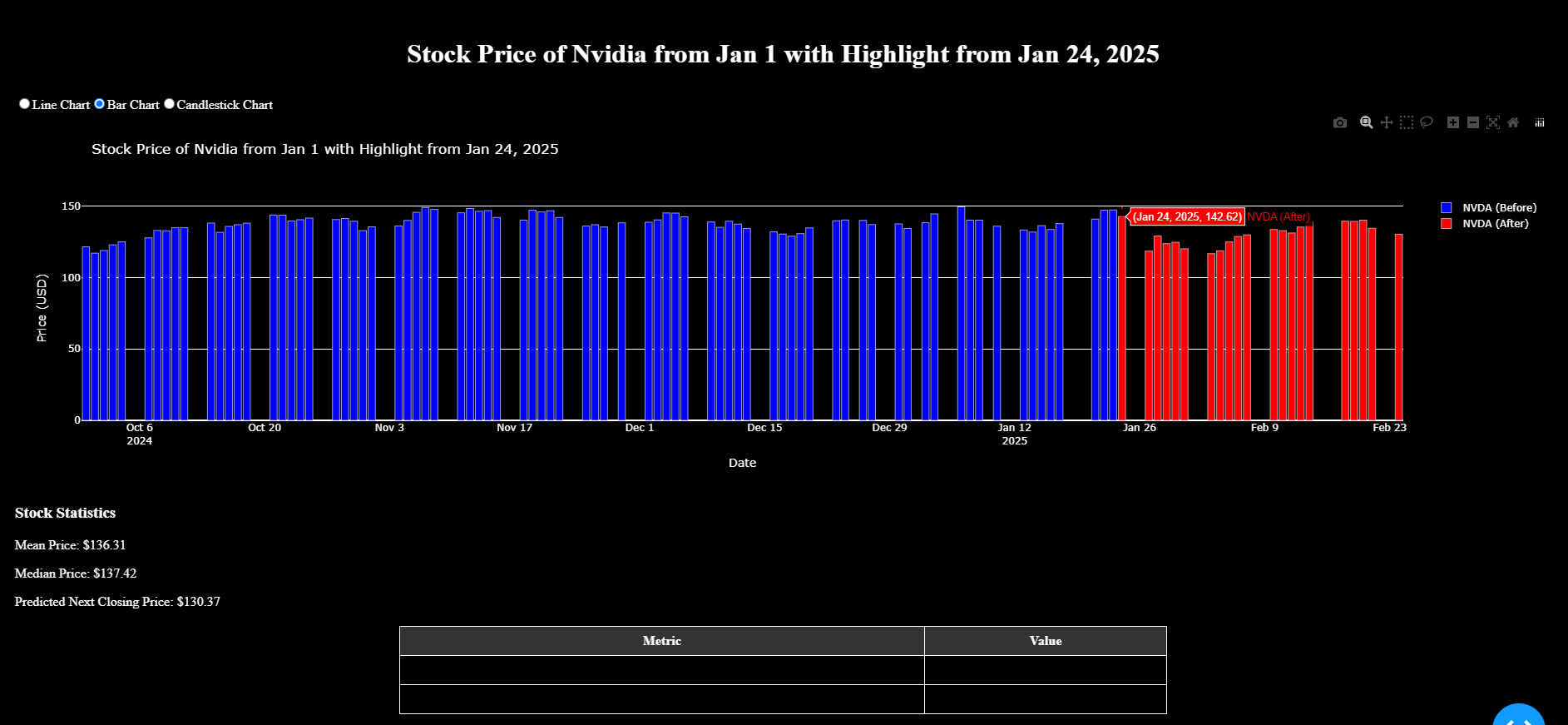
**Two Distinct Line Segments:**

* + **Blue Line (Before January 24, 2025):** This segment plots Nvidia's closing prices from January 1, 2024, up to—but not including—January 24, 2025.
  + **Red Line (After January 24, 2025):** This segment displays the closing prices from January 24, 2025, onward. The red color visually highlights this period.

**BAR PLOT:**

**Two Segmented Bar Groups:**

* + **Blue Bars (Before January 24, 2025**): These bars represent Nvidia's closing prices from January 1, 2024, up until (but not including) January 24, 2025.
  + **Red Bars (After January 24, 2025):**These bars display the closing prices from January 24, 2025, onward, using red to highlight this period.



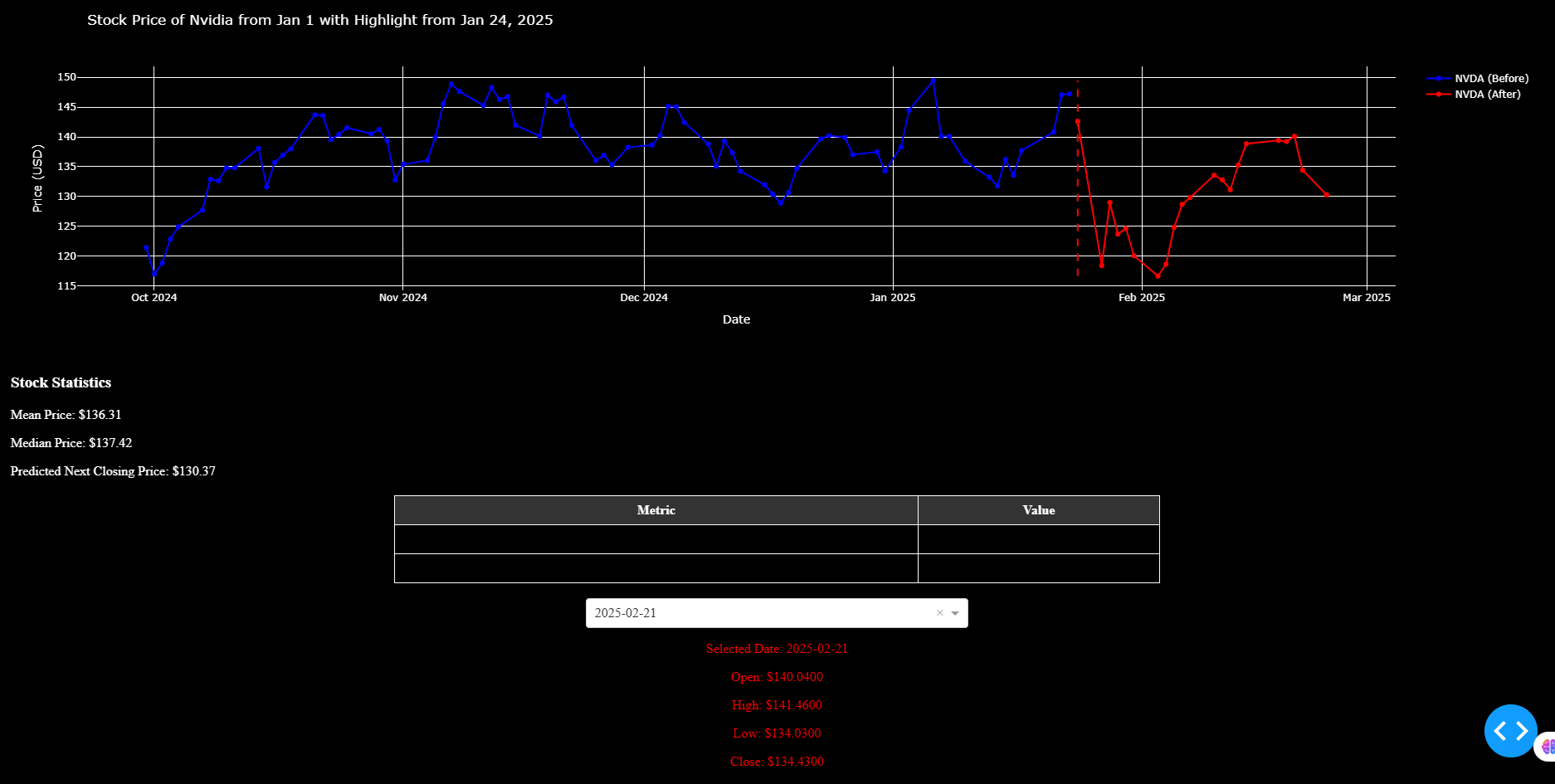
**CANDLE STICK:**

When you select the Candlestick Chart option, you'll see a detailed financial chart that represents daily price movements in a traditional candlestick format.

**PREDICTION MODEL:**

The prediction model in this project is a simple linear extrapolation based on historical closing prices. It calculates the average change between consecutive days and then adds this average daily change to the most recent closing price to forecast the next day's value. This straightforward method assumes that the stock will continue to move at a rate similar to its recent average, providing a basic estimate rather than a detailed forecast that accounts for market volatility or external factors.

So we can also collect the values of the Stock price of a particular date by selecting the drop down button and selecting the particular date.



**Future Enhancements and Potential Extensions**

* **Advanced Forecasting:**Integrate machine learning models to provide more accurate stock price predictions.
* **Additional Metrics:**Include more financial metrics like volume, moving averages, RSI (Relative Strength Index), etc.
* **User Customization:**Allow users to select custom date ranges, compare multiple stocks, or overlay technical indicators on the graphs.
* **Improved UI/UX:**Enhance the styling with more responsive design elements, additional tooltips, and interactive filters**.**
* **Real-Time Updates:**Implement periodic data refreshes to provide near-real-time stock updates.

**Conclusion**:

An interactive plot displaying the historical stock prices and predicted future prices, with distinct colors for different time frames and dashed lines for predictions**.**

**CODE:**

**import requests**

**import plotly.graph\_objects as go**

**import dash**

**from dash import dcc, html**

**from dash.dependencies import Input, Output**

**import numpy as np**

**API\_KEY = "0580ZZYG5OQTVS3V"  # Replace with your API key**

**API\_URL = "https://www.alphavantage.co/query"**

**SYMBOL = "NVDA"**

**START\_DATE = "2024-01-01"       # Starting data from January 1**

**HIGHLIGHT\_DATE = "2025-01-24"   # Highlight data from January 24, 2025 onward**

**def fetch\_stock\_data(symbol):**

**params = {**

**"function": "TIME\_SERIES\_DAILY",**

**"symbol": symbol,**

**"apikey": API\_KEY**

**}**

**response = requests.get(API\_URL, params=params)**

**data = response.json()**

**return data.get("Time Series (Daily)", {})**

**def filter\_stock\_data(stock\_data):**

**# Filter for dates greater than or equal to START\_DATE**

**filtered\_data = {**

**date: float(values["4. close"])**

**for date, values in stock\_data.items()**

**if date >= START\_DATE**

**}**

**sorted\_dates = sorted(filtered\_data.keys())**

**return sorted\_dates, [filtered\_data[date] for date in sorted\_dates]**

**stock\_data = fetch\_stock\_data(SYMBOL)**

**dates, prices = filter\_stock\_data(stock\_data)**

**def predict\_next\_day\_price(prices):**

**# A simple prediction using the average daily difference**

**return round(prices[-1] + (np.mean(np.diff(prices))), 2)**

**app = dash.Dash(\_\_name\_\_)**

**server = app.server  # Expose the server for deployment if needed**

**app.layout = html.Div([**

**html.H1(**

**"Stock Price of Nvidia from Jan 1 with Highlight from Jan 24, 2025",**

**style={'textAlign': 'center', 'backgroundColor': 'black', 'color': 'white', 'padding': '10px'}**

**),**

**dcc.RadioItems(**

**id="graph-type",**

**options=[**

**{"label": "Line Chart", "value": "line"},**

**{"label": "Bar Chart", "value": "bar"},**

**{"label": "Candlestick Chart", "value": "candlestick"}**

**],**

**value="line",**

**inline=True,**

**style={'color': 'white'}**

**),**

**dcc.Graph(id="stock-chart"),**

**html.H3("Stock Statistics", style={'margin-top': '20px', 'color': 'white'}),**

**html.P(id="stock-mean", style={'color': 'white'}),**

**html.P(id="stock-median", style={'color': 'white'}),**

**html.P(id="stock-prediction", style={'color': 'white'}),**

**html.Div(id="stock-table"),**

**html.Br(),**

**dcc.Dropdown(**

**id="date-dropdown",**

**options=[{'label': d, 'value': d} for d in dates],**

**placeholder="Select a date",**

**style={'width': '50%', 'margin': 'auto'}**

**),**

**html.Div(id="selected-metrics")**

**], style={'backgroundColor': 'black', 'padding': '20px'})**

**@app.callback(**

**[Output("stock-chart", "figure"),**

**Output("stock-mean", "children"),**

**Output("stock-median", "children"),**

**Output("stock-prediction", "children"),**

**Output("stock-table", "children")],**

**[Input("graph-type", "value")]**

**)**

**def update\_graph(graph\_type):**

**mean\_price = round(np.mean(prices), 2)**

**median\_price = round(np.median(prices), 2)**

**predicted\_price = predict\_next\_day\_price(prices)**

**# Split data into before and after the highlight date**

**dates\_before = [d for d in dates if d < HIGHLIGHT\_DATE]**

**prices\_before = [prices[i] for i, d in enumerate(dates) if d < HIGHLIGHT\_DATE]**

**dates\_after = [d for d in dates if d >= HIGHLIGHT\_DATE]**

**prices\_after = [prices[i] for i, d in enumerate(dates) if d >= HIGHLIGHT\_DATE]**

**if graph\_type == "line":**

**trace1 = go.Scatter(**

**x=dates\_before, y=prices\_before, mode="lines+markers",**

**name="NVDA (Before)", line=dict(color='blue')**

**)**

**trace2 = go.Scatter(**

**x=dates\_after, y=prices\_after, mode="lines+markers",**

**name="NVDA (After)", line=dict(color='red')**

**)**

**figure = go.Figure(data=[trace1, trace2])**

**elif graph\_type == "bar":**

**trace1 = go.Bar(**

**x=dates\_before, y=prices\_before,**

**name="NVDA (Before)", marker\_color='blue'**

**)**

**trace2 = go.Bar(**

**x=dates\_after, y=prices\_after,**

**name="NVDA (After)", marker\_color='red'**

**)**

**figure = go.Figure(data=[trace1, trace2])**

**elif graph\_type == "candlestick":**

**figure = go.Figure(**

**data=[go.Candlestick(**

**x=dates,**

**open=[float(stock\_data[date]["1. open"]) for date in dates],**

**high=[float(stock\_data[date]["2. high"]) for date in dates],**

**low=[float(stock\_data[date]["3. low"]) for date in dates],**

**close=prices,**

**name="NVDA Candlestick"**

**)]**

**)**

**# Add a vertical rectangle to highlight the period from HIGHLIGHT\_DATE to the last date**

**figure.add\_vrect(**

**x0=HIGHLIGHT\_DATE, x1=dates[-1],**

**fillcolor="rgba(255,0,0,0.2)",**

**line\_width=0,**

**annotation\_text="Highlighted",**

**annotation\_position="top left"**

**)**

**figure.update\_layout(**

**title={"text": "Stock Price of Nvidia from Jan 1 with Highlight from Jan 24, 2025", "font": {"color": "white"}},**

**xaxis\_title="Date",**

**yaxis\_title="Price (USD)",**

**paper\_bgcolor="black",**

**plot\_bgcolor="black",**

**font\_color="white",**

**shapes=[**

**# For non-candlestick charts, add a vertical dashed line at the highlight date**

**{**

**"type": "line",**

**"x0": HIGHLIGHT\_DATE,**

**"x1": HIGHLIGHT\_DATE,**

**"y0": min(prices),**

**"y1": max(prices),**

**"line": {"color": "red", "width": 2, "dash": "dash"},**

**}**

**]**

**)**

**# Prepare table data: End Price and Predicted Price**

**end\_price = prices[-1] if prices else "N/A"**

**table = html.Table(**

**children=[**

**html.Thead(**

**html.Tr([**

**html.Th("Metric", style={'padding': '8px', 'border': '1px solid white'}),**

**html.Th("Value", style={'padding': '8px', 'border': '1px solid white'})**

**]),**

**style={'backgroundColor': '#333', 'color': 'white'}**

**),**

**html.Tbody([**

**html.Tr([**

**html.Td("End Price", style={'padding': '8px', 'border': '1px solid white'}),**

**html.Td(f"${end\_price}", style={'padding': '8px', 'border': '1px solid white'})**

**]),**

**html.Tr([**

**html.Td("Predicted Next Price", style={'padding': '8px', 'border': '1px solid white'}),**

**html.Td(f"${predicted\_price}", style={'padding': '8px', 'border': '1px solid white'})**

**])**

**])**

**],**

**style={'width': '50%', 'margin': 'auto', 'marginTop': '20px', 'borderCollapse': 'collapse'}**

**)**

**return (**

**figure,**

**f"Mean Price: ${mean\_price}",**

**f"Median Price: ${median\_price}",**

**f"Predicted Next Closing Price: ${predicted\_price}",**

**table**

**)**

**@app.callback(**

**Output("selected-metrics", "children"),**

**[Input("date-dropdown", "value")]**

**)**

**def update\_selected\_metrics(selected\_date):**

**if selected\_date is None:**

**return ""**

**data = stock\_data.get(selected\_date)**

**if not data:**

**return f"No data available for {selected\_date}"**

**open\_price = data.get("1. open", "N/A")**

**high\_price = data.get("2. high", "N/A")**

**low\_price = data.get("3. low", "N/A")**

**close\_price = data.get("4. close", "N/A")**

**return html.Div([**

**html.P(f"Selected Date: {selected\_date}"),**

**html.P(f"Open: ${open\_price}"),**

**html.P(f"High: ${high\_price}"),**

**html.P(f"Low: ${low\_price}"),**

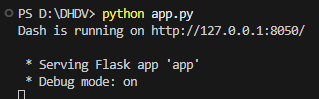
**html.P(f"Close: ${close\_price}")**

**], style={'color': 'red', 'textAlign': 'center'})**

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

**app.run\_server(debug=True)**

To RUN the File :



**Using the Link or IP Address we can access the Dash Out of the Above Code**

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