EDA Project: Nifty 50 (1-Year Data Analysis)

Objective

Analyze the daily performance of the Nifty 50 Index over the past year to uncover market trends, volatility, seasonality, and investor behavior patterns.

1. Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-v0_8-darkgrid')
import plotly.graph_objects as go


pd.set_option('display.max_columns', None)
```

2. Load the Dataset

```
import yfinance as yf
nifty = yf.download("^NSEI", period="1y")
nifty.reset_index(inplace=True)
nifty.head()
```

```
/tmp/ipython-input-577906451.py:2: FutureWarning:

YF.download() has changed argument auto_adjust default to True

[*********************100%***********************]  1 of 1 completed
```

Price	Date	Close	High	Low	Open	Volume
Ticker		^NSEI	^NSEI	^NSEI	^NSEI	^NSEI
0	2024-10-31	24205.349609	24372.449219	24172.599609	24349.849609	287000
1	2024-11-01	24304.349609	24368.250000	24280.199219	24302.750000	38800
2	2024-11-04	23995.349609	24316.750000	23816.150391	24315.750000	285500
3	2024-11-05	24213.300781	24229.050781	23842.750000	23916.500000	289500
4	2024-11-06	24484.050781	24537.599609	24204.050781	24308.750000	351100

Next steps: Generate code with `nifty` New interactive sheet

3. Data Cleaning

```
nifty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249 entries, 0 to 248
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   (Date, )        249 non-null    datetime64[ns]
 1   (Close, ^NSEI)  249 non-null    float64
 2   (High, ^NSEI)   249 non-null    float64
 3   (Low, ^NSEI)    249 non-null    float64
 4   (Open, ^NSEI)   249 non-null    float64
 5   (Volume, ^NSEI) 249 non-null    int64
```

```
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 11.8 KB
```

```
nifty.isnull().sum()
```

```
                  0
 Price  Ticker
 Date             0
 Close   ^NSEI    0
 High    ^NSEI    0
 Low     ^NSEI    0
 Open    ^NSEI    0
 Volume  ^NSEI    0

dtype: int64
```

```
nifty.duplicated().sum()
```

```
np.int64(0)
```

```
type(nifty['Date'])
```

```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None,
fastpath: bool | lib.NoDefault=lib.no_default) -> None

/usr/local/lib/python3.12/dist-packages/pandas/core/series.py
One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
```

Data Type Correction

```
pd.to_datetime(nifty['Date'])
```

```
           Date
  0  2024-10-31
  1  2024-11-01
  2  2024-11-04
  3  2024-11-05
  4  2024-11-06
...         ...
244  2025-10-27
245  2025-10-28
246  2025-10-29
247  2025-10-30
248  2025-10-31
249 rows × 1 columns

dtype: datetime64[ns]
```

Handling Missing Values

```
nifty.fillna(method='ffill', inplace=True)
```

```
/tmp/ipython-input-1974732243.py:1: FutureWarning:

DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
```

4. Feature Engineering

```
nifty['Daily_Return'] = nifty['Close'].pct_change() * 100
```

```
nifty['Volatility'] = (nifty['High'] - nifty['Low']) / nifty['Open'] * 100
```

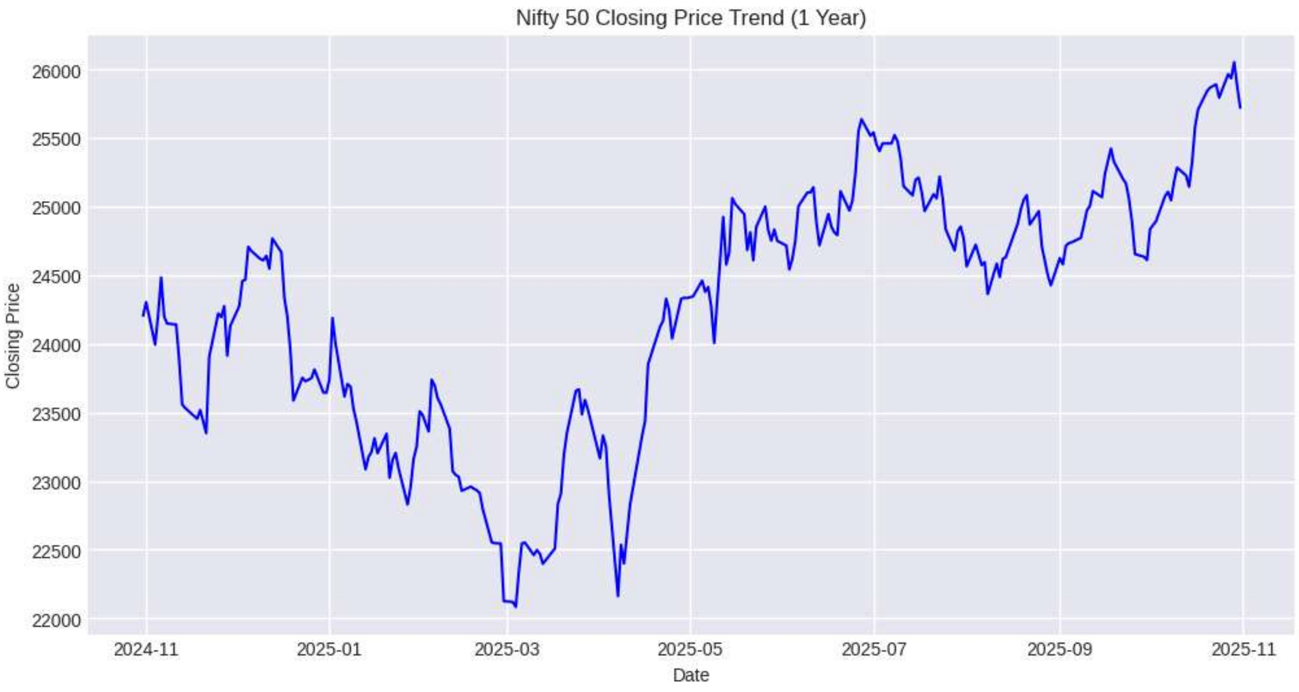
```
nifty['Month'] = nifty['Date'].dt.month_name()
```

```
nifty['Weekday'] = nifty['Date'].dt.day_name()
```

5. Exploratory Data Analysis (EDA)

Insight 1: Nifty 50 Trend Over the Year

```
plt.figure(figsize=(12,6))
plt.plot(nifty['Date'], nifty['Close'], color='blue')
plt.title("Nifty 50 Closing Price Trend (1 Year)")
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.show()
```

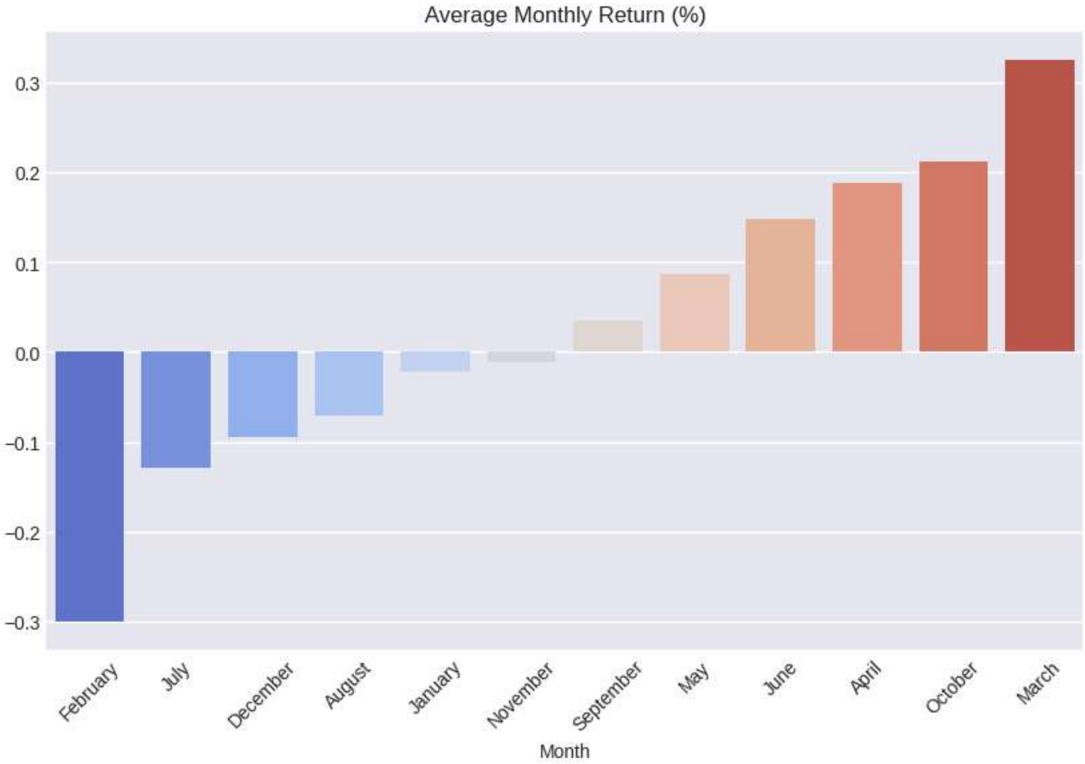


Insight 2: Monthly Performance Overview

Question: Which month performed best?

```
monthly_returns = nifty.groupby('Month')['Daily_Return'].mean().sort_values()
plt.figure(figsize=(10,6))
sns.barplot(x=monthly_returns.index, y=monthly_returns.values, palette='coolwarm')
plt.title("Average Monthly Return (%)")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-1749748106.py:3: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
```

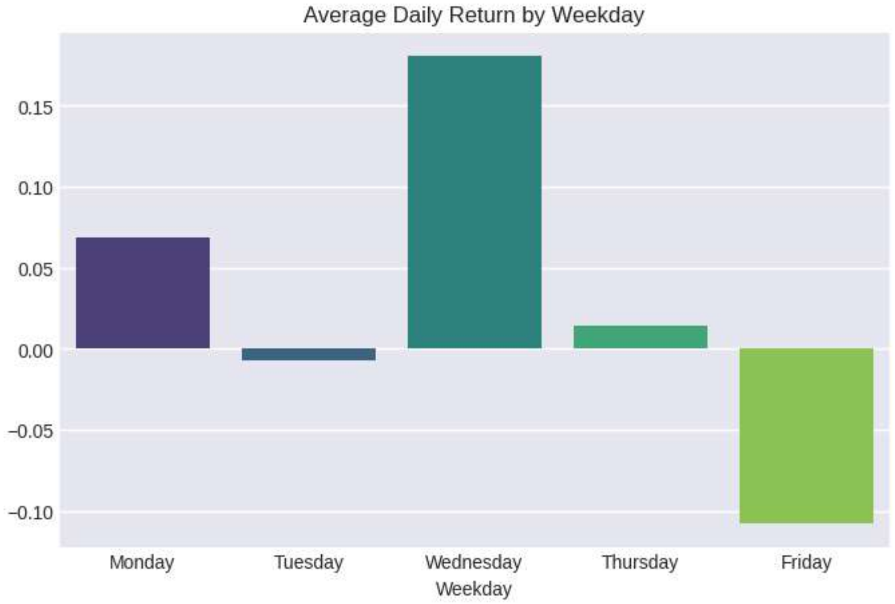


Insight 3: Weekday Effect

Question: Are some days more profitable

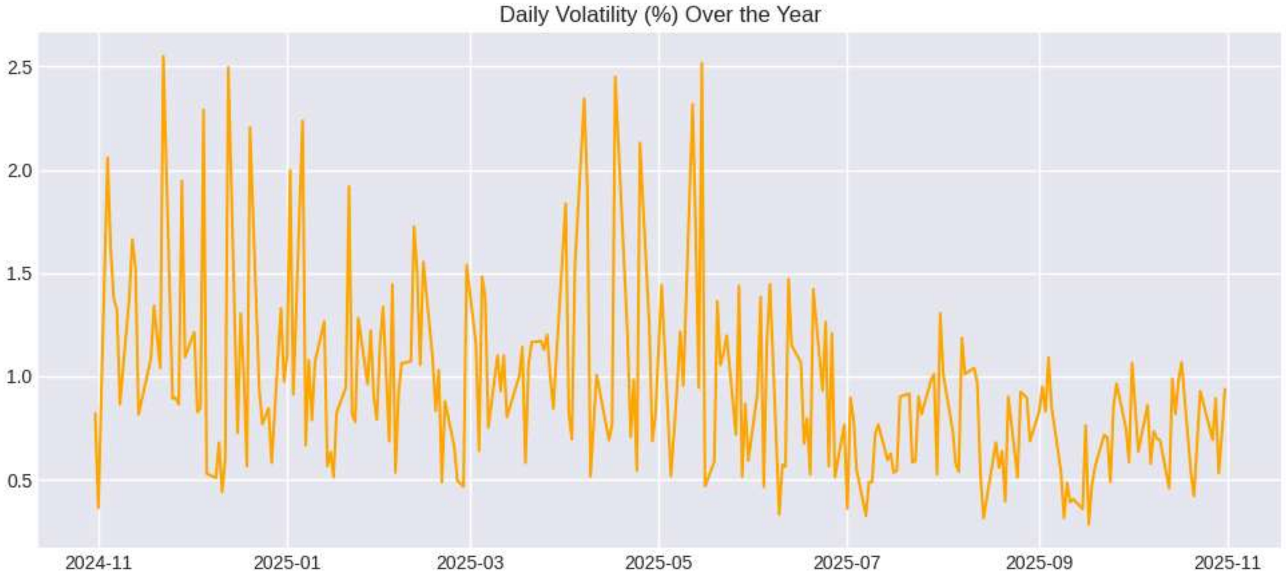
```
weekday_perf = nifty.groupby('Weekday')['Daily_Return'].mean().reindex(
    ['Monday','Tuesday','Wednesday','Thursday','Friday']
)
plt.figure(figsize=(8,5))
sns.barplot(x=weekday_perf.index, y=weekday_perf.values, palette='viridis')
plt.title("Average Daily Return by Weekday")
plt.show()
```

```
/tmp/ipython-input-2979614378.py:5: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
```



Insight 4: Daily Volatility Pattern

```
plt.figure(figsize=(12,5))
plt.plot(nifty['Date'], nifty['Volatility'], color='orange')
plt.title("Daily Volatility (%) Over the Year")
plt.show()
```

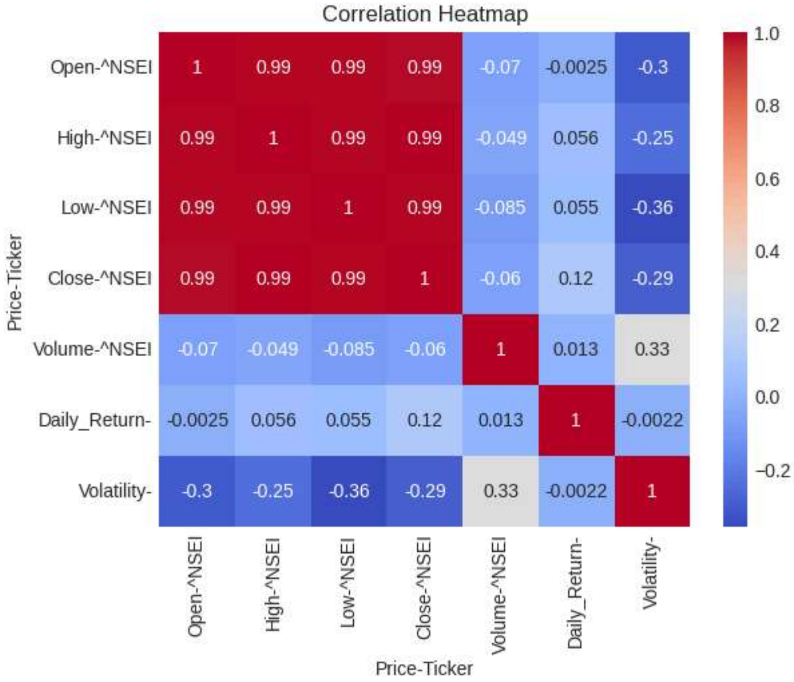


Detect high-volatility months — often due to budget sessions, RBI announcements, or global events.

Insight 5: Correlation Between Variables

```
sns.heatmap(nifty[['Open','High','Low','Close','Volume','Daily_Return','Volatility']].corr(),
            annot=True, cmap='coolwarm')
```

```
plt.title("Correlation Heatmap")
plt.show()
```

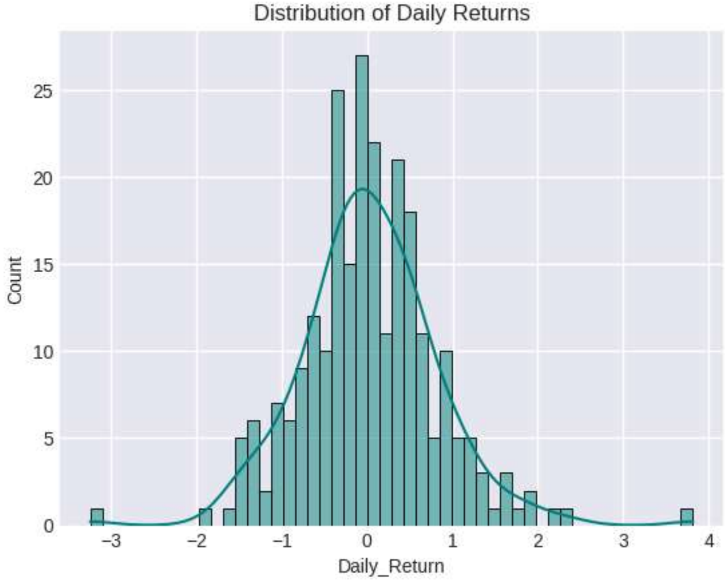


High, Low, and Close are highly correlated → expected.

Volume may have weak correlation → shows that trading volume alone doesn’t drive price.

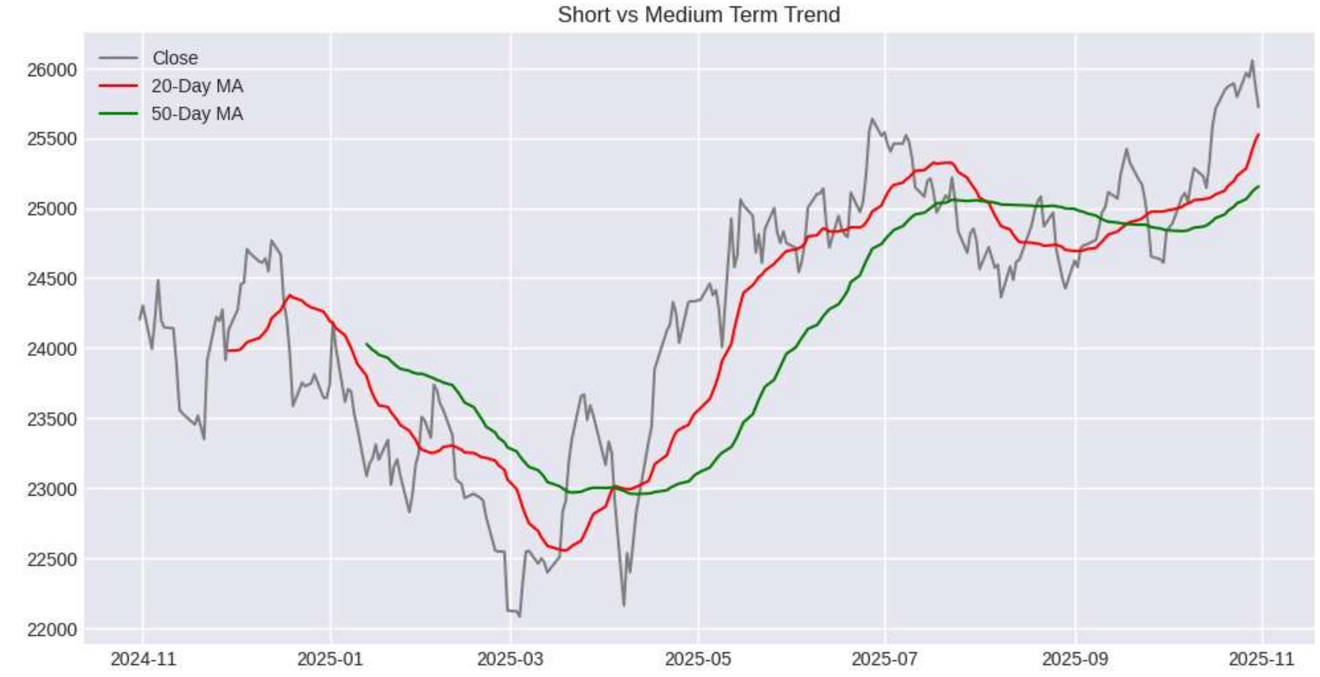
Insight 6: Distribution of Daily Returns

```
sns.histplot(nifty['Daily_Return'].dropna(), bins=50, kde=True, color='teal')
plt.title("Distribution of Daily Returns")
plt.show()
```



Insight 7: Rolling Mean & Trend Strength

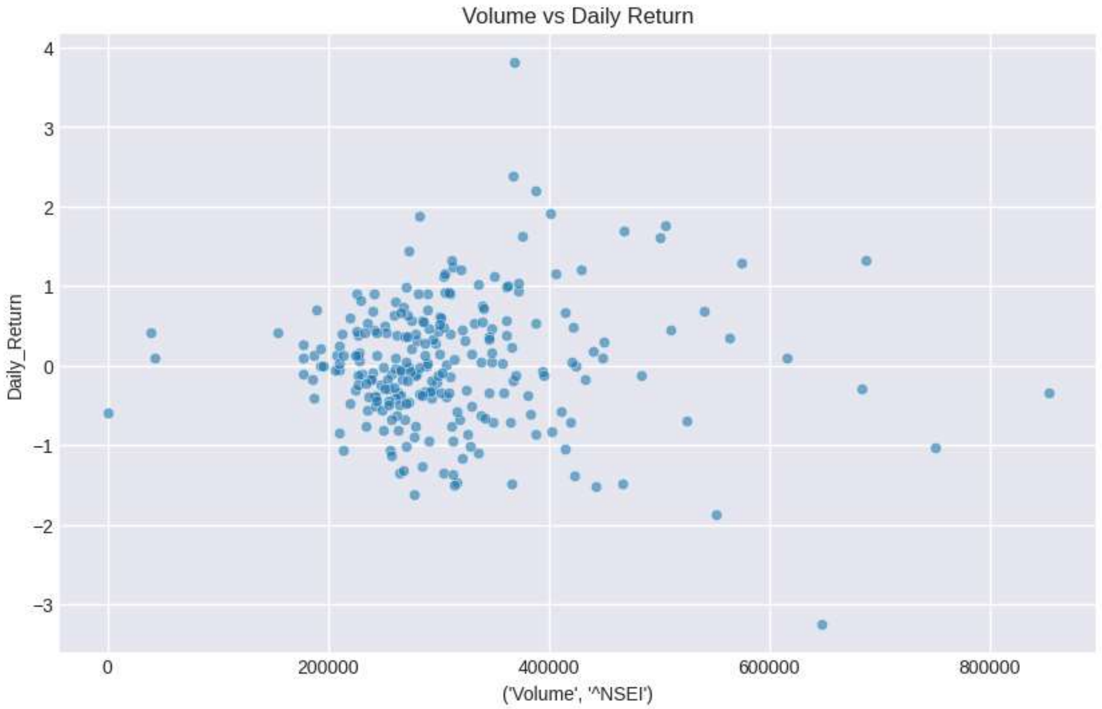
```
plt.figure(figsize=(12,6))
plt.plot(nifty['Date'], nifty['Close'], label='Close', color='gray')
plt.plot(nifty['Date'], nifty['Close'].rolling(window=20).mean(), label='20-Day MA', color='red')
plt.plot(nifty['Date'], nifty['Close'].rolling(window=50).mean(), label='50-Day MA', color='green')
plt.title("Short vs Medium Term Trend")
plt.legend()
plt.show()
```



Shows momentum and trend reversals — e.g., “Golden Cross” when 20-day MA crosses above 50-day.

Insight 8: Volume vs Price Movement

```
plt.figure(figsize=(10,6))
sns.scatterplot(x=nifty[('Volume', '^NSEI')], y=nifty['Daily_Return'], alpha=0.6)
plt.title("Volume vs Daily Return")
plt.show()
```



No strong correlation → high volume doesn’t always mean high return.

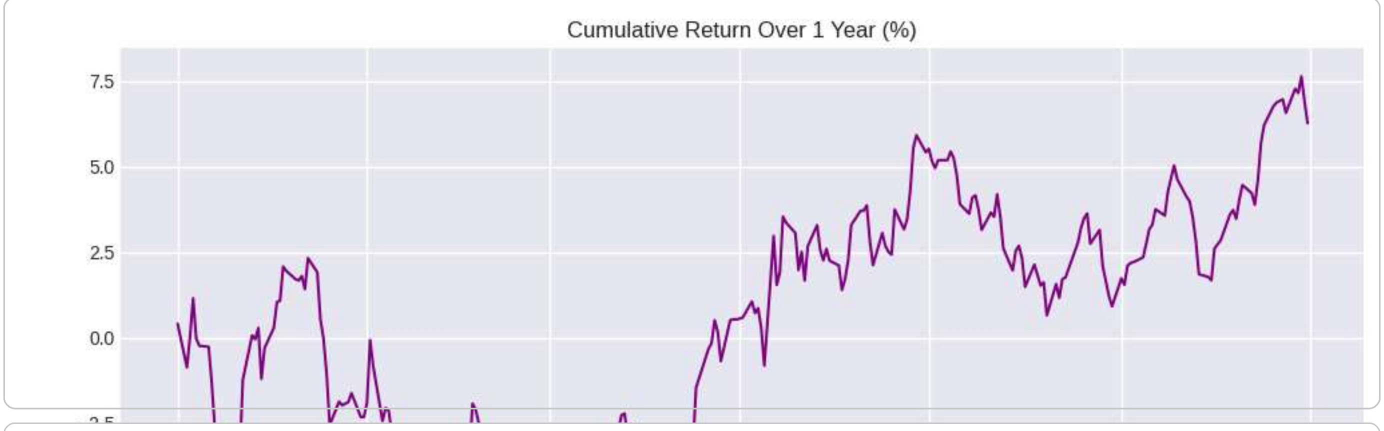
Insight 9: Identify Biggest Gain & Drop Days

```
max_gain = nifty.loc[nifty['Daily_Return'].idxmax()]
max_loss = nifty.loc[nifty['Daily_Return'].idxmin()]

print("Biggest Gain:", max_gain['Date'], round(max_gain['Daily_Return'],2), "%")
print("Biggest Drop:", max_loss['Date'], round(max_loss['Daily_Return'],2), "%")
```

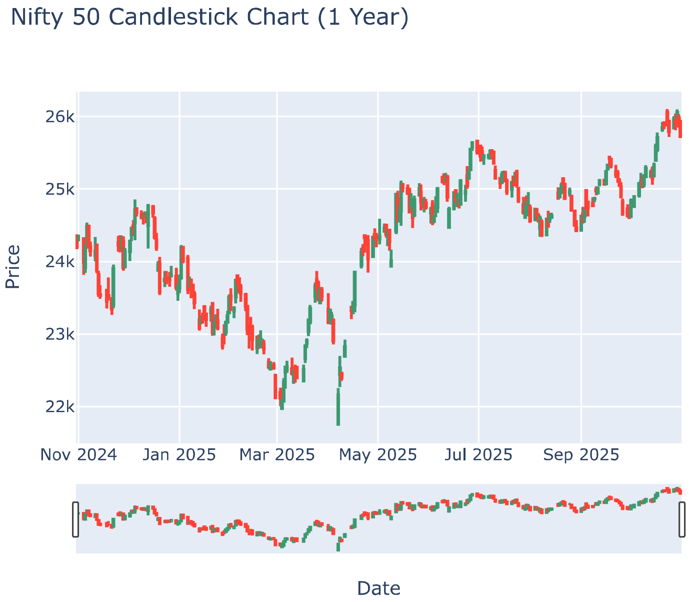
```
Biggest Gain: Ticker
   2025-05-12
Name: 128, dtype: datetime64[ns] Ticker
    3.818307
Name: 128, dtype: object %
Biggest Drop: Ticker
   2025-04-07
Name: 107, dtype: datetime64[ns] Ticker
   -3.243255
Name: 107, dtype: object %
```

Insight 10: Cumulative Return

```
nifty['Cumulative_Return'] = (1 + nifty['Daily_Return']/100).cumprod() - 1
plt.figure(figsize=(12,6))
plt.plot(nifty['Date'], nifty['Cumulative_Return']*100, color='purple')
plt.title("Cumulative Return Over 1 Year (%)")
plt.show()
```



```
fig = go.Figure(data=[go.Candlestick(x=nifty['Date'],
                open=nifty[('Open', '^NSEI')],
                high=nifty[('High', '^NSEI')],
                low=nifty[('Low', '^NSEI')],
                close=nifty[('Close', '^NSEI')])])

fig.update_layout(title='Nifty 50 Candlestick Chart (1 Year)',
                  xaxis_title='Date',
                  yaxis_title='Price')

fig.show()
```



```
plt.figure(figsize=(12, 6))
plt.plot(nifty['Date'], nifty['Daily_Return'], color='blue')
plt.title("Daily Returns Over Time")
plt.xlabel("Date")
plt.ylabel("Daily Return (%)")
plt.grid(True)
plt.show()
```