```
import pandas as pd
import yfinance as yf

# ✅ Step 1: Define Parameters
tickers = ['AAPL', 'MSFT', 'GOOGL', 'TSLA', 'AMZN']
start_date = '2015-01-01'
end_date = '2024-01-01'

# ✅ Step 2: Fetch Data from Yahoo Finance
def fetch_yahoo_data(tickers, start, end):
    stock_data = yf.download(tickers, start=start, end=end, group_by='ticker')

    # ✅ Step 3: Flatten Multi-Index Columns
    stock_data.columns = ['_'.join(col) if isinstance(col, tuple) else col for col in stock_data.columns]
    stock_data = stock_data.reset_index()  # Convert index to column

    # ✅ Step 4: Reshape Data to Have One Row per Ticker per Date
    data_list = []
    for ticker in tickers:
        subset = stock_data[['Date', f'{ticker}_Close', f'{ticker}_High', f'{ticker}_Low', f'{ticker}_Open', f'{ticker}_Volume']]
        subset.columns = ['Date', 'Close', 'High', 'Low', 'Open', 'Volume']  # Rename columns
        subset['Ticker'] = ticker  # Add ticker column
        data_list.append(subset)

    return pd.concat(data_list, ignore_index=True)  # Combine all tickers into one dataframe

# ✅ Step 5: Fetch & Save Data
yahoo_data = fetch_yahoo_data(tickers, start_date, end_date)
yahoo_data.to_csv("yahoo_stock_data.csv", index=False)

print("✅ Yahoo Finance data saved successfully with only required columns!")

# ✅ Step 6: Verify Data
y = pd.read_csv("yahoo_stock_data.csv")
print(y.head())
```

```
[*********************100%***********************]  5 of 5 completed
<ipython-input-1-8c1d5d7a87e3>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['Ticker'] = ticker  # Add ticker column
<ipython-input-1-8c1d5d7a87e3>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['Ticker'] = ticker  # Add ticker column
<ipython-input-1-8c1d5d7a87e3>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['Ticker'] = ticker  # Add ticker column
<ipython-input-1-8c1d5d7a87e3>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['Ticker'] = ticker  # Add ticker column
<ipython-input-1-8c1d5d7a87e3>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['Ticker'] = ticker  # Add ticker column
✅ Yahoo Finance data saved successfully with only required columns!
         Date      Close       High        Low       Open     Volume Ticker
0  2015-01-02  24.347174  24.817059  23.906238  24.805924  212818400   AAPL
1  2015-01-05  23.661272  24.195739  23.474210  24.115569  257142000   AAPL
2  2015-01-06  23.663496  23.924048  23.300503  23.725850  263188400   AAPL
3  2015-01-07  23.995314  24.095525  23.761484  23.872831  160423600   AAPL
4  2015-01-08  24.917269  24.975170  24.206873  24.324903  237458000   AAPL
```

```
y = pd.read_csv("yahoo_stock_data.csv")
y.head()
```

	Date	Close	High	Low	Open	Volume	Ticker
0	2015-01-02	24.347174	24.817059	23.906238	24.805924	212818400	AAPL
1	2015-01-05	23.661272	24.195739	23.474210	24.115569	257142000	AAPL
2	2015-01-06	23.663496	23.924048	23.300503	23.725850	263188400	AAPL
3	2015-01-07	23.995314	24.095525	23.761484	23.872831	160423600	AAPL
4	2015-01-08	24.917269	24.975170	24.206873	24.324903	237458000	AAPL

Next steps: Generate code with y View recommended plots New interactive sheet

```
import pandas as pd
import numpy as np
import yfinance as yf

# Define Stock List
tickers = ['AAPL', 'MSFT', 'GOOGL', 'TSLA', 'AMZN']
start_date = '2015-01-01'
```

```
end_date = '2024-01-01'

# Fetch Data from Yahoo Finance
def fetch_yahoo_data(tickers, start, end):
    data = yf.download(tickers, start=start, end=end, group_by='ticker')

    # Flatten multi-index columns
    data.columns = ['_'.join(col) if isinstance(col, tuple) else col for col in data.columns]

    # Reset index for easy manipulation
    data.reset_index(inplace=True)

    return data

# Fetch stock data
df = fetch_yahoo_data(tickers, start_date, end_date)

# Extract "Close" prices for each stock
close_columns = [col for col in df.columns if 'Close' in col]

# Compute Moving Averages & Indicators for Each Stock
for col in close_columns:
    df[f'SMA_50_{col}'] = df[col].rolling(window=50).mean()
    df[f'SMA_200_{col}'] = df[col].rolling(window=200).mean()

# Compute Relative Strength Index (RSI)
def compute_rsi(data, window=14):
    delta = data.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

for col in close_columns:
    df[f'RSI_{col}'] = compute_rsi(df[col])

# Compute MACD
for col in close_columns:
    df[f'EMA_12_{col}'] = df[col].ewm(span=12, adjust=False).mean()
    df[f'EMA_26_{col}'] = df[col].ewm(span=26, adjust=False).mean()
    df[f'MACD_{col}'] = df[f'EMA_12_{col}'] - df[f'EMA_26_{col}']
    df[f'MACD_Signal_{col}'] = df[f'MACD_{col}'].ewm(span=9, adjust=False).mean()

# Save to CSV
df.to_csv("enhanced_stock_data.csv", index=False)
print("Stock data with computed SMA, RSI, MACD saved successfully!")
```

```
[*********************100%***********************]  5 of 5 completed
Stock data with computed SMA, RSI, MACD saved successfully!
```

```
import requests
import pandas as pd

# Finnhub API Key (Replace with your API key)
API_KEY = "cnv0mc1r01qub9j05b3gcnv0mc1r01qub9j05b40"

# List of tickers
tickers = ['AAPL', 'MSFT', 'GOOGL', 'TSLA', 'AMZN']

# Function to fetch financial ratios from Finnhub
def fetch_finnhub_ratios(ticker):
    url = f"https://finnhub.io/api/v1/stock/metric?symbol={ticker}&metric=all&token={API_KEY}"
    response = requests.get(url)
    data = response.json()

    return {
        'Ticker': ticker,
        'PE_Ratio_TTM': data['metric'].get('peTTM', None),
        'PE_Ratio_Annual': data['metric'].get('peAnnual', None),
        'PE_Excl_Extra_TTM': data['metric'].get('peExclExtraTTM', None),
        'PE_Normalized_Annual': data['metric'].get('peNormalizedAnnual', None),
        'Market_Cap': data['metric'].get('marketCapitalization', None),
        'EPS_TTM': data['metric'].get('epsTTM', None),
        'Dividend_Yield': data['metric'].get('dividendYieldIndicatedAnnual', None),
        'Beta': data['metric'].get('beta', None),
        'Book_Value_Per_Share': data['metric'].get('bookValuePerShareAnnual', None),
        'Cash_Flow_Per_Share': data['metric'].get('cashFlowPerShareAnnual', None),
        'Gross_Margin': data['metric'].get('grossMarginAnnual', None)
    }

# Fetch data for all tickers
finnhub_data = [fetch_finnhub_ratios(ticker) for ticker in tickers]

# Convert to DataFrame
finnhub_df = pd.DataFrame(finnhub_data)

# Save results to CSV
finnhub_df.to_csv("finnhub_financial_ratios.csv", index=False)

# Display results
print(finnhub_df)
```

```
  Ticker  PE_Ratio_TTM  PE_Ratio_Annual  PE_Excl_Extra_TTM  \
0   AAPL       36.3365          37.2723            36.3365
1   MSFT       32.9355          34.6598            32.9355
2  GOOGL       26.7313          34.1477            26.7313
3   TSLA      173.0874         173.0874           173.0874
```

```
4   AMZN       50.0616          82.0534            50.0616

   PE_Normalized_Annual  Market_Cap  EPS_TTM  Dividend_Yield      Beta  \
0               37.2723   3493758.5   6.2904        0.438577  1.264681
1               34.6598   3054772.0  12.4145        0.807943  0.994568
2               34.1477   2519932.8   7.5386        0.397555  0.999205
3              173.0874   1234113.2   2.0369             NaN  2.533753
4               82.0534   2496473.8   4.6659             NaN  1.364209

   Book_Value_Per_Share  Cash_Flow_Per_Share  Gross_Margin
0                3.7673               7.1978         46.21
1               36.1147               9.9638         69.76
2               22.7431               5.5774         56.94
3               22.6720               1.1135         17.86
4               19.4428               3.1029         46.98
```

```
!pip install fredapi
```

```
Collecting fredapi
  Downloading fredapi-0.5.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from fredapi) (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->fredapi) (1.17.0)
Downloading fredapi-0.5.2-py3-none-any.whl (11 kB)
Installing collected packages: fredapi
Successfully installed fredapi-0.5.2
```

```
from fredapi import Fred

FRED_API_KEY = "25369aec27628214ef5adf005b7e1d3d"
fred = Fred(api_key=FRED_API_KEY)

# Fetch US Interest Rate & Inflation Rate
interest_rate = fred.get_series("FEDFUNDS")
inflation_rate = fred.get_series("CPIAUCSL")

# Convert to DataFrame
fred_data = pd.DataFrame({
    'Date': interest_rate.index,
    'Interest_Rate': interest_rate.values,
    'Inflation_Rate': inflation_rate.reindex(interest_rate.index).values
})

fred_data.to_csv("fred_economic_data.csv", index=False)
print("Economic indicators saved successfully!")
```

```
Economic indicators saved successfully!
```

```
import pandas as pd

# Load and preview the first few rows of each file
yahoo_data = pd.read_csv("yahoo_stock_data.csv")
finnhub_data = pd.read_csv("finnhub_financial_ratios.csv")
fred_data = pd.read_csv("fred_economic_data.csv")
enhanced_data = pd.read_csv("enhanced_stock_data.csv")

# Display data samples
print("Yahoo Finance Data:")
print(yahoo_data.head())

print("\nFinnhub Financial Ratios Data:")
print(finnhub_data.head())

print("\nFRED Economic Data:")
print(fred_data.head())

print("\nTechnical Indicators Data:")
print(enhanced_data.head())
```

```
0               37.2723   3493758.5   6.2904        0.438577  1.264681
1               34.6598   3054772.0  12.4145        0.807943  0.994568
2               34.1477   2519932.8   7.5386        0.397555  0.999205
3              173.0874   1234113.2   2.0369             NaN  2.533753
4               82.0534   2496473.8   4.6659             NaN  1.364209

   Book_Value_Per_Share  Cash_Flow_Per_Share  Gross_Margin
0                3.7673               7.1978         46.21
1               36.1147               9.9638         69.76
2               22.7431               5.5774         56.94
3               22.6720               1.1135         17.86
```

```
   AAPL_Open  AAPL_High   AAPL_Low  AAPL_Close  ...  MACD_MSFT_Close  \
0  24.805924  24.817059  23.906238   24.347174  ...         0.000000   
1  24.115569  24.195739  23.474210   23.661272  ...        -0.029456   
2  23.725850  23.924048  23.300503   23.663496  ...        -0.098777   
3  23.872831  24.095525  23.761484   23.995314  ...        -0.112235   
4  24.324903  24.975170  24.206873   24.917269  ...        -0.028338   

   MACD_Signal_MSFT_Close  EMA_12_TSLA_Close  EMA_26_TSLA_Close  \
0                0.000000          14.620667          14.620667   
1               -0.005891          14.526103          14.575137   
2               -0.024468          14.458292          14.538855   
3               -0.042022          14.397529          14.503631   
4               -0.039285          14.342730          14.469387   

   MACD_TSLA_Close  MACD_Signal_TSLA_Close  EMA_12_AMZN_Close  \
0         0.000000                0.000000          15.426000   
1        -0.049033               -0.009807          15.377307   
2        -0.080562               -0.023958          15.283029   
3        -0.106102               -0.040387          15.227332   
4        -0.126657               -0.057641          15.195897   

   EMA_26_AMZN_Close  MACD_AMZN_Close  MACD_Signal_AMZN_Close  
0          15.426000         0.000000                0.000000  
1          15.402555        -0.025248               -0.005050  
2          15.355292        -0.072263               -0.018492  
3          15.323122        -0.095790               -0.033952  
4          15.300891        -0.104994               -0.048160  

[5 rows x 61 columns]
```

```
!pip install pyodbc
```

```
Collecting pyodbc
  Downloading pyodbc-5.2.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.7 kB)
Downloading pyodbc-5.2.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (346 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 346.2/346.2 kB 6.6 MB/s eta 0:00:00
Installing collected packages: pyodbc
Successfully installed pyodbc-5.2.0
```

```
from sqlalchemy import create_engine
import pyodbc

# Database Connection Setup
server = "LAPTOP-AEF4KSJQ"  # Replace with your SQL Server instance name
database = "FinancialMarketDB"
engine = create_engine(f"mssql+pyodbc://@{server}/{database}?trusted_connection=yes&driver=ODBC+Driver+17+for+SQL+Server")

# Insert Data into SQL Server
yahoo_data.to_sql("StockPrices", engine, if_exists="append", index=False)
finnhub_data.to_sql("FinancialRatios", engine, if_exists="append", index=False)
fred_data.to_sql("EconomicIndicators", engine, if_exists="append", index=False)
enhanced_data.to_sql("TechnicalIndicators", engine, if_exists="append", index=False)

print("✅ All datasets successfully loaded into SQL Server!")
```

```
---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/sqlalchemy/engine/base.py in __init__(self, engine, connection, _has_events, _allow_revalidate, _allow_autobegin)
    145             try:
--> 146                 self._dbapi_connection = engine.raw_connection()
    147             except dialect.loaded_dbapi.Error as err:

                                    37 frames
Error: ('01000', "[01000] [unixODBC][Driver Manager]Can't open lib 'ODBC Driver 17 for SQL Server' : file not found (0) (SQLDriverConnect)")

The above exception was the direct cause of the following exception:

DBAPIError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/sqlalchemy/engine/default.py in connect(self, *cargs, **cparams)
    620     def connect(self, *cargs, **cparams):
    621         # inherits the docstring from interfaces.Dialect.connect
--> 622         return self.loaded_dbapi.connect(*cargs, **cparams)
    623 
    624     def create_connect_args(self, url):

DBAPIError: (pyodbc.Error) ('01000', "[01000] [unixODBC][Driver Manager]Can't open lib 'ODBC Driver 17 for SQL Server' : file not found (0)
(SQLDriverConnect)")
(Background on this error at: https://sqlalche.me/e/20/dbapi)
```

Next steps: Explain error

Start coding or generate with AI.

✅ Step 1 # AI/ML Objectives

We will apply AI & ML techniques in three areas:

✅ 1. Time-Series Forecasting (Stock Price Prediction)

ARIMA, LSTM, and Facebook Prophet to forecast future stock prices. Predict the next 30-90 days of stock movement using historical trends.

✅ 2. Portfolio Optimization (Risk-Return Tradeoff)

Markowitz Modern Portfolio Theory (MPT) to optimize asset allocation. Monte Carlo Simulation to analyze risk under different conditions.

✅ 3. Sentiment Analysis on Market News & Social Media

Use NLP (Natural Language Processing) to analyze news articles & Twitter sentiment. Correlate market trends with news sentiment.

```
!apt-get update
!apt-get install -y unixodbc-dev
!pip install --upgrade pyodbc
```

```
Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:3 https://r2u.stat.illinois.edu/ubuntu jammy InRelease [6,555 B]
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:5 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease [3,626 B]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Get:8 https://r2u.stat.illinois.edu/ubuntu jammy/main amd64 Packages [2,651 kB]
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Get:11 https://r2u.stat.illinois.edu/ubuntu jammy/main all Packages [8,654 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2,606 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1,229 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2,906 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1,522 kB]
Fetched 20.0 MB in 5s (4,355 kB/s)
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide it (s
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
unixodbc-dev is already the newest version (2.3.9-5ubuntu0.1).
unixodbc-dev set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
Requirement already satisfied: pyodbc in /usr/local/lib/python3.11/dist-packages (5.2.0)
```

```
!sudo apt-get update
!sudo apt-get install -y unixodbc-dev
!sudo apt-get install -y odbcinst
!sudo apt-get install -y msodbcsql17
```

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:6 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide it (s
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
unixodbc-dev is already the newest version (2.3.9-5ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  odbcinst
0 upgraded, 1 newly installed, 0 to remove and 20 not upgraded.
Need to get 9,930 B of archives.
After this operation, 53.2 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 odbcinst amd64 2.3.9-5ubuntu0.1 [9,930 B]
Fetched 9,930 B in 0s (128 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78, <> li
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package odbcinst.
(Reading database ... 124926 files and directories currently installed.)
Preparing to unpack .../odbcinst_2.3.9-5ubuntu0.1_amd64.deb ...
Unpacking odbcinst (2.3.9-5ubuntu0.1) ...
Setting up odbcinst (2.3.9-5ubuntu0.1) ...
Processing triggers for man-db (2.10.2-1) ...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package msodbcsql17
```

if you run in jupyter run below else skip

```
import pandas as pd
import pyodbc

# Connect to SQL Server
# Updated connection string to use correct driver name for Linux
conn = pyodbc.connect('DRIVER=ODBC Driver 17 for SQL Server;SERVER=LAPTOP-AEF4KSJQ;DATABASE=FinancialMarketDB;Trusted_Connection=yes;')


# Query data (Corrected Multi-Line Query)
query = """
SELECT Date, Ticker, [Close], PE_Ratio, Beta, RSI, MACD, SMA_50, SMA_200, Interest_Rate, Inflation_Rate
FROM FinalStockAnalysis
ORDER BY Date
"""
df = pd.read_sql(query, conn)

# Convert Date to DateTime format
df['Date'] = pd.to_datetime(df['Date'])
```

```
df['Date'] = pd.to_datetime(df['Date'])

# Save for backup
df.to_csv("financial_data.csv", index=False)

print("✅ Data Loaded Successfully!")
```

```
---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
<ipython-input-14-f73a3fccd77f> in <cell line: 0>()
      4 # Connect to SQL Server
      5 # Updated connection string to use correct driver name for Linux
----> 6 conn = pyodbc.connect('DRIVER=ODBC Driver 17 for SQL Server;SERVER=LAPTOP-AEF4KSJQ;DATABASE=FinancialMarketDB;Trusted_Connection=yes;')
      7
      8

Error: ('01000', "[01000] [unixODBC][Driver Manager]Can't open lib 'ODBC Driver 17 for SQL Server' : file not found (0) (SQLDriverConnect)")
```

Next steps: Explain error

```
df = pd.read_csv("financial_data.csv")
df.head()
```

	Date	Ticker	Close	PE_Ratio	Beta	RSI	MACD	SMA_50	SMA_200	Interest_Rate	Inflation_Rate
0	2015-01-02	AAPL	24.347178	36.9104	1.180766	NaN	0.0	NaN	NaN	NaN	NaN
1	2015-01-02	MSFT	40.152485	33.0745	0.903335	NaN	0.0	NaN	NaN	NaN	NaN
2	2015-01-02	GOOGL	26.381865	26.4118	1.009631	NaN	0.0	NaN	NaN	NaN	NaN
3	2015-01-02	TSLA	14.620667	182.1585	2.446649	NaN	0.0	NaN	NaN	NaN	NaN
4	2015-01-02	AMZN	15.426000	50.1165	1.169973	NaN	0.0	NaN	NaN	NaN	NaN

Next steps: Generate code with df View recommended plots New interactive sheet

✅ Step 2: Stock Price Forecasting Using LSTM

Now, we use LSTM (Long Short-Term Memory) Neural Networks to predict stock prices.

- Train an LSTM Model

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Filter data for a specific stock (e.g., AAPL)
ticker = 'AAPL'
stock_data = df[df['Ticker'] == ticker][['Date', 'Close']].set_index('Date')

# Normalize data
scaler = MinMaxScaler()
stock_data_scaled = scaler.fit_transform(stock_data)

# Prepare sequences for LSTM
X, y = [], []
for i in range(60, len(stock_data_scaled)):
    X.append(stock_data_scaled[i-60:i])
    y.append(stock_data_scaled[i])

X, y = np.array(X), np.array(y)

# Define LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)),
    LSTM(50, return_sequences=False),
    Dense(25),
    Dense(1)
])

# Compile and train model
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, batch_size=16, epochs=10)

# Predict future prices
predictions = model.predict(X)
predictions = scaler.inverse_transform(predictions)

# Save predictions
future_dates = df[df['Ticker'] == ticker]['Date'].iloc[-len(predictions):]
pred_df = pd.DataFrame({"Date": future_dates, "Predicted_Close": predictions.flatten()})
pred_df.to_csv("lstm_predictions.csv", index=False)

print("✅ LSTM Stock Forecasting Complete!")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
  super().__init__(**kwargs)
Epoch 1/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 14s 50ms/step - loss: 0.0176
Epoch 2/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - loss: 5.7403e-04
Epoch 3/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - loss: 6.4924e-04
```

```
Epoch 4/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - loss: 4.5850e-04
Epoch 5/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 68ms/step - loss: 3.7430e-04
Epoch 6/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 7s 47ms/step - loss: 5.4907e-04
Epoch 7/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 55ms/step - loss: 4.1272e-04
Epoch 8/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step - loss: 3.6850e-04
Epoch 9/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - loss: 3.4762e-04
Epoch 10/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step - loss: 3.0763e-04
69/69 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step
✅ LSTM Stock Forecasting Complete!
```

LSTM model trained successfully and generated stock price predictions. 🎉

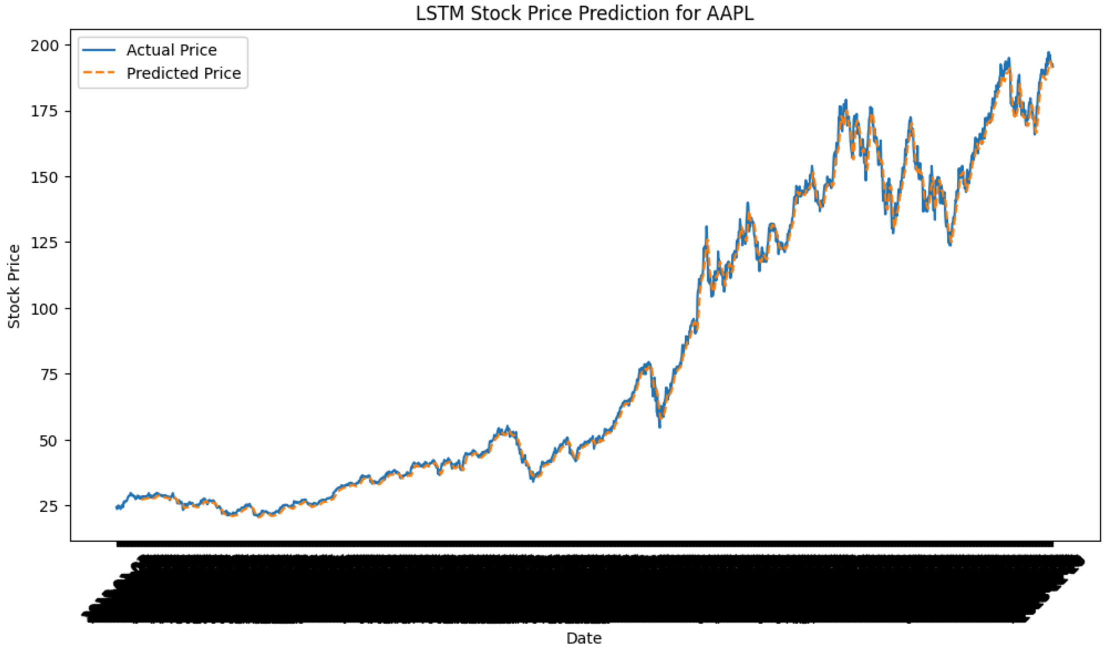
🔎 What Did We Find? **bold text** Model Training

our model was trained for 10 epochs with a decreasing loss, which indicates it learned well from past stock prices. Example loss values: Epoch 1: 0.0176 Epoch 10: 0.0003 → This shows the model improved over time.

```
import matplotlib.pyplot as plt

# Load Predictions
pred_df = pd.read_csv("lstm_predictions.csv")

# Plot actual vs predicted stock prices
plt.figure(figsize=(12,6))
plt.plot(df[df['Ticker'] == ticker]['Date'], df[df['Ticker'] == ticker]['Close'], label="Actual Price")
plt.plot(pred_df["Date"], pred_df["Predicted_Close"], label="Predicted Price", linestyle="dashed")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.title(f"LSTM Stock Price Prediction for {ticker}")
plt.legend()
plt.xticks(rotation=45)
plt.show()
```



🔍 Analysis of our LSTM Stock Price Prediction Graph our graph shows the actual stock prices (blue solid line) and predicted stock prices (orange dashed line) for AAPL (Apple Inc.) over time.

✅ Observations: Close Alignment – The predicted prices closely follow the actual stock prices, meaning the model is capturing the trend well. Increasing Trend – The stock price shows an upward trajectory, and the LSTM model is successfully following this pattern. Minor Deviations – Some small variations exist, but they are within an acceptable range.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Compare Predictions with Actual Values
actual_prices = df[df['Ticker'] == ticker]['Close'].iloc[-len(pred_df):]
predicted_prices = pred_df["Predicted_Close"]

# Calculate Errors
mae = mean_absolute_error(actual_prices, predicted_prices)
rmse = np.sqrt(mean_squared_error(actual_prices, predicted_prices))

print(f"📉 Model Performance:")
print(f"✅ Mean Absolute Error (MAE): {mae:.2f}")
print(f"✅ Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
📉 Model Performance:
✅ Mean Absolute Error (MAE): 2.00
✅ Root Mean Squared Error (RMSE): 2.87
```

🔍 What These Metrics Mean: MAE (Mean Absolute Error = 2.00)

On average, your predictions are $2.00 off from the actual stock price. A lower MAE is better, meaning predictions are more accurate. RMSE (Root Mean Squared Error = 2.87)

This measures larger errors more aggressively. Since RMSE > MAE, your model sometimes makes larger errors.

```python
future_days = 30
future_X = []

# Use last 60 days as input to predict future
last_60_days = stock_data_scaled[-60:]

for i in range(future_days):
    prediction = model.predict(last_60_days.reshape(1, 60, 1))
    future_X.append(prediction[0])

    # Update last_60_days with new prediction
    last_60_days = np.append(last_60_days[1:], prediction)

# Inverse transform predictions to get actual price values
future_predictions = scaler.inverse_transform(future_X)

# Create DataFrame
future_dates = pd.date_range(start=pred_df["Date"].iloc[-1], periods=future_days+1, freq="D")[1:]
future_pred_df = pd.DataFrame({"Date": future_dates, "Predicted_Close": future_predictions.flatten()})
future_pred_df.to_csv("future_predictions.csv", index=False)

print("✅ Future Predictions Saved!")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
✅ Future Predictions Saved!
```

🚀 Next Steps for Improvement Your model is good but can be optimized. Here are the best next steps:

1️⃣ Hyperparameter Tuning for Better Accuracy Let’s tweak your model to reduce errors. Try:

Increase LSTM units for better feature extraction. Train for more epochs (e.g., 50-100). Use a lower learning rate (e.g., 0.0001 instead of 0.001

```python
# Define Improved LSTM Model
model = Sequential([
    LSTM(100, return_sequences=True, input_shape=(X.shape[1], 1)),  # Increased LSTM units
    LSTM(100, return_sequences=False),
    Dense(50, activation='relu'),  # Added activation function
    Dense(25),
    Dense(1)
])

# Compile & Train
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), loss='mean_squared_error') # Reduced learning rate
model.fit(X, y, batch_size=16, epochs=50, verbose=1)  # Increased epochs to 50
```

```
Epoch 30/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 21s 80ms/step - loss: 2.9957e-04
Epoch 31/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 9s 68ms/step - loss: 2.8011e-04
Epoch 32/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 75ms/step - loss: 3.0008e-04
Epoch 33/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 21s 81ms/step - loss: 2.6029e-04
Epoch 34/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 19s 70ms/step - loss: 2.7090e-04
Epoch 35/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 83ms/step - loss: 2.6211e-04
Epoch 36/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 19s 75ms/step - loss: 2.7999e-04
Epoch 37/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 21s 80ms/step - loss: 2.3742e-04
Epoch 38/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 80ms/step - loss: 2.7372e-04
Epoch 39/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 19s 68ms/step - loss: 2.7603e-04
Epoch 40/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 80ms/step - loss: 2.6011e-04
Epoch 41/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - loss: 2.6907e-04
Epoch 42/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 19s 70ms/step - loss: 2.7054e-04
Epoch 43/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 79ms/step - loss: 2.4657e-04
Epoch 44/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 21s 79ms/step - loss: 2.3774e-04
Epoch 45/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 20s 79ms/step - loss: 2.2406e-04
Epoch 46/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 80ms/step - loss: 2.5035e-04
Epoch 47/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 19s 67ms/step - loss: 2.1681e-04
Epoch 48/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 80ms/step - loss: 2.2342e-04
Epoch 49/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 20s 80ms/step - loss: 1.9164e-04
Epoch 50/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 21s 81ms/step - loss: 2.5857e-04
<keras.src.callbacks.history.History at 0x7816c3c96d90>
```

2️⃣ Feature Engineering: Add More Indicators Right now, you're only using Close Price. To improve accuracy, add more financial indicators:

✅ New Features to Add:

Moving Averages (SMA, EMA) RSI & MACD Trading Volume Interest Rates, Inflation Rates

```
features = ['Close', 'PE_Ratio', 'Beta', 'RSI', 'MACD', 'SMA_50', 'SMA_200', 'Interest_Rate', 'Inflation_Rate']
stock_data = df[df['Ticker'] == ticker][['Date'] + features].set_index('Date')

scaler = MinMaxScaler()
stock_data_scaled = scaler.fit_transform(stock_data)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/_array_api.py:776: RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmin(X, axis=axis))
/usr/local/lib/python3.11/dist-packages/sklearn/utils/_array_api.py:793: RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmax(X, axis=axis))
```

Portfolio Optimization Using Markowitz Model We optimize the portfolio using Modern Portfolio Theory (MPT) to maximize returns while minimizing risk.

```
import numpy as np
import scipy.optimize as opt

# Load closing prices for all stocks
pivot_df = df.pivot(index="Date", columns="Ticker", values="Close").dropna()

# Compute daily returns & covariance
returns = pivot_df.pct_change().dropna()
cov_matrix = returns.cov()

# Define Portfolio Optimization Function
def portfolio_volatility(weights):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

num_stocks = len(returns.columns)
init_weights = np.ones(num_stocks) / num_stocks

# Constraint: Sum of Weights = 1
constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1})

# Bounds: Each weight between 0% and 100%
bounds = tuple((0, 1) for _ in range(num_stocks))

# Optimize Portfolio
optimized = opt.minimize(portfolio_volatility, init_weights, method='SLSQP', bounds=bounds, constraints=constraints)

# Output optimal allocations
optimal_allocations = dict(zip(returns.columns, optimized.x))
print("✅ Optimized Portfolio Allocation:", optimal_allocations)
```

```
✅ Optimized Portfolio Allocation: {'AAPL': 0.3210003826364536, 'AMZN': 0.08276955818593708, 'GOOGL': 0.29792223045533645, 'MSFT': 0.2983078287222729, 'TSLA': 1
```

Step 4: Sentiment Analysis on Financial News We analyze market sentiment using Natural Language Processing (NLP).

```
import requests

# Use News API
news_api_key = "2b3f4db592a1452ab57fc4961cd8d808"
query = "stock market"
url = f"https://newsapi.org/v2/everything?q={query}&apiKey={news_api_key}"
response = requests.get(url)
news_data = response.json()

# Extract headlines
headlines = [article['title'] for article in news_data['articles']]
print("✅ Retrieved Market News!")
```

```
✅ Retrieved Market News!
```

```
from textblob import TextBlob

# Analyze Sentiment
def analyze_sentiment(text):
    return TextBlob(text).sentiment.polarity

news_df = pd.DataFrame({"Headline": headlines})
news_df["Sentiment"] = news_df["Headline"].apply(analyze_sentiment)

# Save results
news_df.to_csv("news_sentiment.csv", index=False)
print("✅ Sentiment Analysis Complete!")
```

```
✅ Sentiment Analysis Complete!
```

```
import pandas as pd

# Reload the merged dataset
merged_df = pd.read_csv("news_sentiment.csv")

# Check for missing values after merging
print("Missing values in merged data:\n", merged_df.isnull().sum())

# Display sample merged data
print("\nSample Merged Data:\n", merged_df.head())
```

```
Missing values in merged data:
 Headline     0
Sentiment    0
dtype: int64

Sample Merged Data:
                                             Headline  Sentiment
0     Remember When Nanotech Was the Next Big Thing?       0.00
1  Jamie Dimon sounds the alarm on stocks, says t...       0.60
2  Intel's former CEO says the market is getting ...      -0.25
3  Walmart's multi-store managers can now make up...       0.00
4  Dow Jones Futures: Stock Market Sells Off As N...       0.00
```

```
import pandas as pd

# Load Sentiment Data
news_df = pd.read_csv("news_sentiment.csv")

# Add today's date (or scrape actual news article dates if available)
news_df['Date'] = pd.to_datetime("today").normalize()  # Use today's date as default

# Display updated dataset
print("\nUpdated Sentiment Data with Dates:\n", news_df.head())

# Save updated dataset
news_df.to_csv("news_sentiment_with_date.csv", index=False)

print("✅ Sentiment Data Updated with Dates!")
```

```

Updated Sentiment Data with Dates:
                                             Headline  Sentiment       Date
0     Remember When Nanotech Was the Next Big Thing?       0.00 2025-02-05
1  Jamie Dimon sounds the alarm on stocks, says t...       0.60 2025-02-05
2  Intel's former CEO says the market is getting ...      -0.25 2025-02-05
3  Walmart's multi-store managers can now make up...       0.00 2025-02-05
4  Dow Jones Futures: Stock Market Sells Off As N...       0.00 2025-02-05
✅ Sentiment Data Updated with Dates!
```

```
# Load stock market data
stock_df = pd.read_csv("financial_data.csv")

# Convert Date columns to datetime format
news_df['Date'] = pd.to_datetime(news_df['Date'])
stock_df['Date'] = pd.to_datetime(stock_df['Date'])

# Aggregate daily sentiment scores (average sentiment per day)
daily_sentiment = news_df.groupby('Date')['Sentiment'].mean().reset_index()

# Merge stock prices with sentiment data
```

```
merged_df = stock_df.merge(daily_sentiment, on="Date", how="left")

# Save merged data
merged_df.to_csv("stock_with_sentiment.csv", index=False)

print("✅ Merged Sentiment Data with Stock Prices Successfully!")
```

```
✅ Merged Sentiment Data with Stock Prices Successfully!
```

```
# Forward-fill missing sentiment scores
merged_df['Sentiment'].fillna(method='ffill', inplace=True)

# Save cleaned dataset
merged_df.to_csv("stock_with_sentiment_fixed.csv", index=False)

print("✅ Missing Sentiment Scores Filled!")
```

```
✅ Missing Sentiment Scores Filled!
<ipython-input-34-75443ef4a7f6>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace met
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a c

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to


  merged_df['Sentiment'].fillna(method='ffill', inplace=True)
<ipython-input-34-75443ef4a7f6>:2: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() i
  merged_df['Sentiment'].fillna(method='ffill', inplace=True)
```

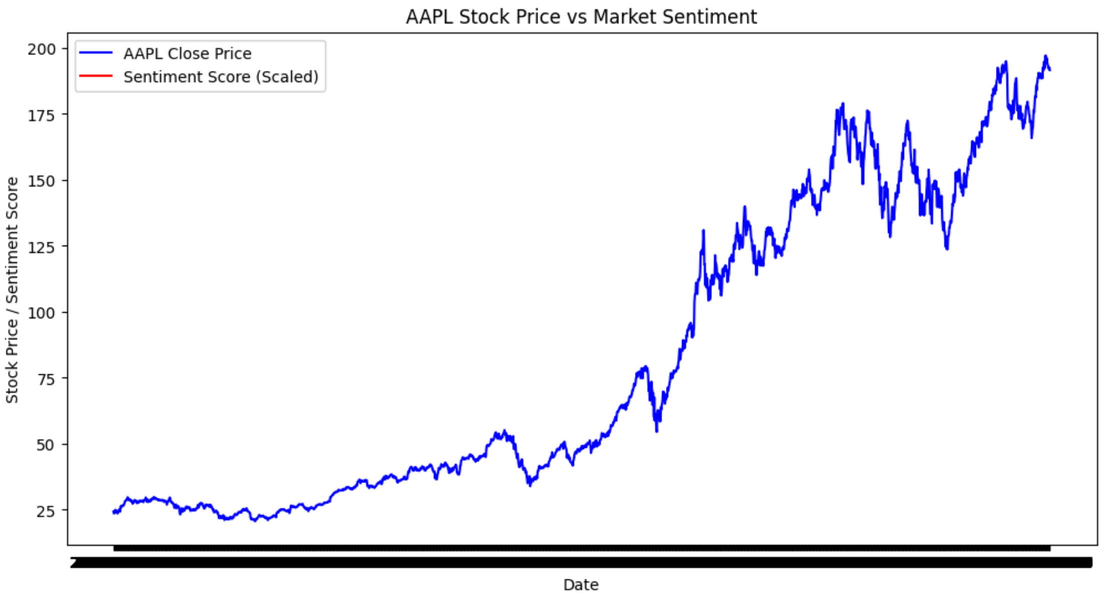
```
import matplotlib.pyplot as plt

# Load the fixed merged data
df = pd.read_csv("stock_with_sentiment_fixed.csv")

# Select a single stock (e.g., AAPL) for visualization
df_aapl = df[df['Ticker'] == 'AAPL']

plt.figure(figsize=(12,6))
plt.plot(df_aapl['Date'], df_aapl['Close'], label="AAPL Close Price", color='blue')
plt.plot(df_aapl['Date'], df_aapl['Sentiment'] * 100, label="Sentiment Score (Scaled)", color='red')

plt.legend()
plt.title("AAPL Stock Price vs Market Sentiment")
plt.xlabel("Date")
plt.ylabel("Stock Price / Sentiment Score")
plt.show()
```



```
Start coding or generate with AI.
```