Stock Market Data Analysis

Objective:

The goal of this project is to analyse historical stock price data of Apple (AAPL), explore key metrics such as daily returns, moving averages, and volatility, and visualize patterns and trends over time to derive financial insights.

Complete code:

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

# Load Stock Data
df = yf.download( tickers: 'AAPL', start='2019-01-01', end='2024-01-01')
df.reset_index(inplace=True)

print(df.info())
print(df.describe())

# Data Cleaning
duplicate_rows = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_rows}")
df.drop_duplicates(inplace=True)

# Dropping missing values
missing_values = df.isnull().sum()
print("Missing values per column:\n", missing_values)
df.dropna(inplace=True)

# Top Best and Worst months
monthly_returns = df.groupby('Month')['Daily Return'].mean().sort_values()
print("Top 5 Worst Months:\n", monthly_returns.head())
print("Top 5 Best Months:\n", monthly_returns.tail())

# Feature Engineering
df['Daily Return'] = df['Close'].pct_change()
df['MA20'] = df['Close'].rolling(window=20).mean()
df['MA50'] = df['Close'].rolling(window=50).mean()
df['Month'] = df['Date'].dt.to_period('M')
```

```python
# Visualizations

# A. Closing Price Over Time
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Close'], label='Close Price')
plt.title("AAPL Closing Price Over Time")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.tight_layout()
plt.show()

# B. Rolling Volatility (20-day)
df['Volatility_20d'] = df['Daily Return'].rolling(window=20).std()
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Volatility_20d'], color='orange')
plt.title("20-Day Rolling Volatility (AAPL)")
plt.xlabel("Date")
plt.ylabel("Volatility")
plt.tight_layout()
plt.show()

# C. Moving Averages
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Close'], label='Close Price')
plt.plot( *args: df['Date'], df['MA20'], label='20-Day MA')
plt.plot( *args: df['Date'], df['MA50'], label='50-Day MA')
plt.title("AAPL Price with Moving Averages")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.tight_layout()
plt.show()
```

```python
# D. Daily Return Distribution
plt.figure(figsize=(8, 5))
sns.histplot(df['Daily Return'].dropna(), bins=50, kde=True, color='skyblue')
plt.title("Distribution of Daily Returns")
plt.xlabel("Daily Return")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# E. Monthly Average Close Price
monthly_avg = df.groupby('Month')['Close'].mean()
monthly_avg.plot(figsize=(12, 5), title="Average Monthly Closing Price")
plt.ylabel("Price ($)")
plt.xlabel("Month")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Snippetwise details:

1.Importing libraries , Using 'ggplot' style for visualization ,loading the data

- Yfinance - For fetching stock market data
- Pandas - For data manipulation and analysis
- Matplotlib - For plotting static charts
- Seaborn - For enhanced visual styling

- We fetch 5 years of Apple stock data from Yahoo Finance
- yfinance downloads open, high, low, close, volume, and adjusted close data.
- reset_index() ensures 'Date' is a column, not an index, for easier plotting.

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

# Load Stock Data
df = yf.download( tickers: 'AAPL', start='2019-01-01', end='2024-01-01')
df.reset_index(inplace=True)
```

2.Basic Information about the dataset

```
print(df.info())
print(df.describe())
```

A.info()

- **Purpose:** Provides a concise summary of a DataFrame.
- **Key points:**
  - Lists the number of non-null entries in each column.
  - Shows the data type (dtype) of each column.
  - Displays the total number of rows in the DataFrame.
  - Indicates memory usage of the DataFrame.
  - Useful for quickly understanding the structure and completeness of the data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   (Date, )        1258 non-null   datetime64[ns]
 1   (Close, AAPL)   1258 non-null   float64
 2   (High, AAPL)    1258 non-null   float64
 3   (Low, AAPL)     1258 non-null   float64
 4   (Open, AAPL)    1258 non-null   float64
 5   (Volume, AAPL)  1258 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
```

B.describe()

- **Purpose:** Generates descriptive statistics of the DataFrame's numeric columns by default.
- **Key points:**
  - Provides statistical summaries such as:
    - Count (count)
    - Mean (mean)
    - Standard deviation (std)
    - Minimum (min)
    - 25th percentile (25%)
    - Median / 50th percentile (50%)
    - 75th percentile (75%)
    - Maximum (max)
  - Can be customized to include non-numeric data (using include='all').
  - Helps to understand data distribution, central tendency, and spread.

```
Price                           Date        Close  ...         Open        Volume
Ticker                                      AAPL  ...         AAPL          AAPL
count                           1258  1258.000000  ...  1258.000000  1.258000e+03
mean    2021-06-30 18:48:38.918918912   120.684961  ...   120.562025  1.015904e+08
min               2019-01-02 00:00:00    33.870827  ...    34.297218  2.404830e+07
25%               2020-04-01 06:00:00    74.842861  ...    74.433461  6.803012e+07
50%               2021-06-30 12:00:00   131.693474  ...   132.042892  8.861740e+07
75%               2022-09-28 18:00:00   157.062782  ...   157.401414  1.189786e+08
max               2023-12-29 00:00:00   196.669754  ...   196.580412  4.265100e+08
std                               NaN    46.477971  ...    46.461484  5.261087e+07

[8 rows x 6 columns]

Process finished with exit code 0
```

3.Feature Engineering

```
# Feature Engineering
df['Daily Return'] = df['Close'].pct_change()
df['MA20'] = df['Close'].rolling(window=20).mean()
df['MA50'] = df['Close'].rolling(window=50).mean()
df['Month'] = df['Date'].dt.to_period('M')
```

- **Definition:** The process of transforming raw data into meaningful features that improve model performance.
- **Key Points:**
  - Includes selecting, modifying, or creating new features from the original data.
  - Techniques: encoding categorical variables, scaling, normalization, binning, polynomial features, etc.
  - Helps the model learn better patterns and reduces overfitting.
  - Essential for enhancing the predictive power of machine learning algorithms.

4.Data cleaning and dropping missing values:

```
# Data Cleaning
duplicate_rows = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_rows}")
df.drop_duplicates(inplace=True)

# Dropping missing values
missing_values = df.isnull().sum()
print("Missing values per column:\n", missing_values)
df.dropna(inplace=True)
```

- **Definition:** The process of fixing or removing incorrect, inconsistent, or incomplete data is called data cleaning
- **Key Points:**
  - Ensures data quality, which directly impacts model accuracy.
  - Reduces noise and errors that can mislead the model.
  - Improves the efficiency of model training.
  - Includes handling outliers, correcting data entry errors, standardizing formats.

Dropping Missing Values

- **Why Drop Missing Values?**
  - Simplifies data preprocessing when missing data is minimal or random.
  - Prevents algorithms from failing or producing biased results.
  - Useful when imputation could introduce bias or complexity.
- **Considerations:**
  - Dropping rows or columns should be done carefully; excessive removal may lead to data loss.
  - Better suited for large datasets where missing data is sparse.
  - Always analyze the pattern of missing data before dropping.

```
Number of duplicate rows: 0
Missing values per column:
 Price    Ticker
Date                0
Close    AAPL       0
High     AAPL       0
Low      AAPL       0
Open     AAPL       0
Volume   AAPL       0
dtype: int64
```

5.Visualization:

A.

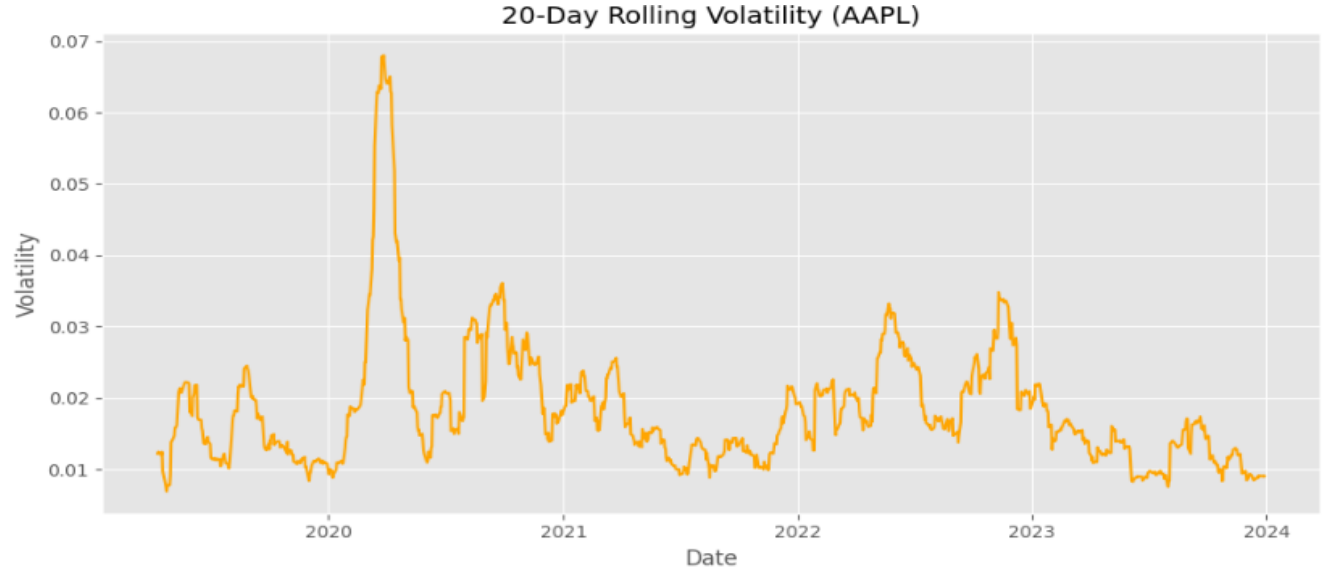
```python
# A. Closing Price Over Time
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Close'], label='Close Price')
plt.title("AAPL Closing Price Over Time")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.tight_layout()
plt.show()
```

- This chart shows how Apple's stock has trended over the past 5 years. Useful for identifying overall upward or downward movement and major dips/spikes .

B.

```python
# B. Rolling Volatility (20-day)
df['Volatility_20d'] = df['Daily Return'].rolling(window=20).std()
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Volatility_20d'], color='orange')
plt.title("20-Day Rolling Volatility (AAPL)")
plt.xlabel("Date")
plt.ylabel("Volatility")
plt.tight_layout()
plt.show()
```

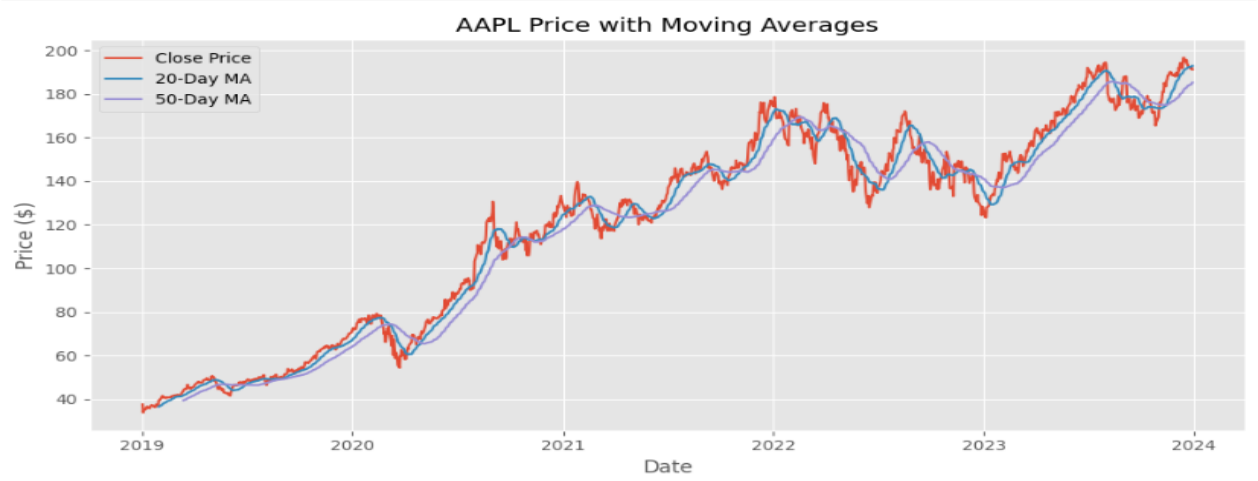
- Measures recent risk or market uncertainty.
- Spikes in volatility often precede big price changes.
- Important for investors managing risk.



C.

```
# C. Moving Averages
plt.figure(figsize=(10, 5))
plt.plot( *args: df['Date'], df['Close'], label='Close Price')
plt.plot( *args: df['Date'], df['MA20'], label='20-Day MA')
plt.plot( *args: df['Date'], df['MA50'], label='50-Day MA')
plt.title("AAPL Price with Moving Averages")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.tight_layout()
plt.show()
```

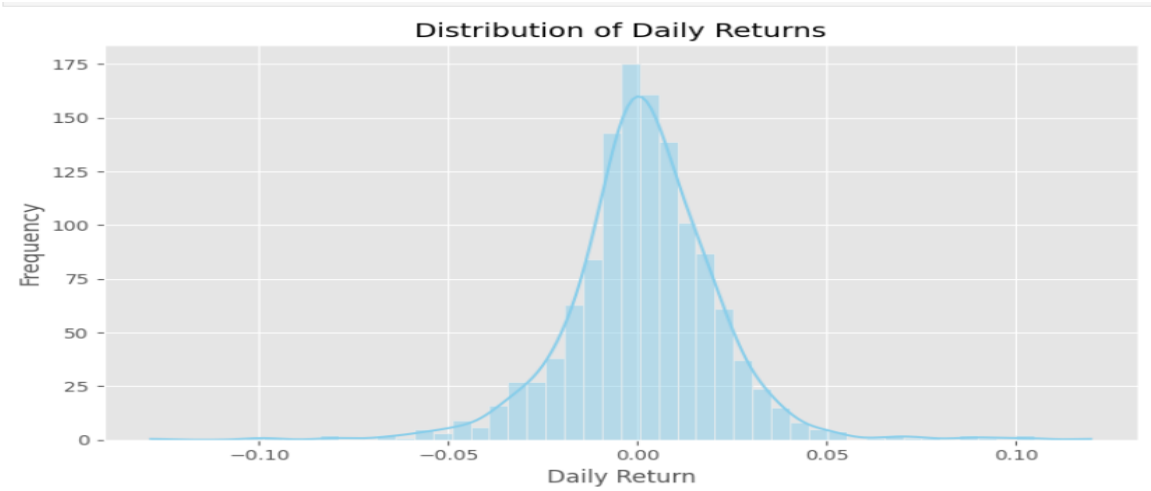
- Helps identify short- vs. long-term trend changes.
- A bullish crossover (MA20 > MA50) may indicate an upward trend.
- Used by traders to spot entry/exit points.



D.

```
# D. Daily Return Distribution
plt.figure(figsize=(8, 5))
sns.histplot(df['Daily Return'].dropna(), bins=50, kde=True, color='skyblue')
plt.title("Distribution of Daily Returns")
plt.xlabel("Daily Return")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

- Shows the distribution of returns (volatility).
- Helps assess risk. A narrower peak → lower volatility.
- Useful for analysts and risk managers.



E.

```python
# E. Monthly Average Close Price
monthly_avg = df.groupby('Month')['Close'].mean()
monthly_avg.plot(figsize=(12, 5), title="Average Monthly Closing Price")
plt.ylabel("Price ($)")
plt.xlabel("Month")
plt.grid(True)
plt.tight_layout()
plt.show()
```

- Reveals long-term patterns and seasonal trends.
- Helpful for quarterly financial analysis.

Conclusion:

This project serves as a comprehensive hands-on demonstration of real-world stock market data analysis using Python. By analyzing 5 years of Apple Inc. (AAPL) stock data, I explored multiple key financial metrics — such as daily returns, moving averages, volatility, and monthly trends — to derive meaningful insights useful for both long-term investors and financial analysts.

Why This Project Is Valuable:

- Aligned with real financial data and industry practices
- Uses standard data pipelines — cleaning → transformation → visualization
- Builds strong foundations in data wrangling, time series analysis, and interpretation of financial indicators
- Offers reusable components for dashboards, investor reports, or further machine learning predictions

What I Learned :

- Downloading & handling real-world financial datasets (using yfinance)
- Performing essential data cleaning (handling duplicates, missing values)
- Engineering features (rolling metrics, returns) critical for financial time series
- Visualizing trends and volatility — core skills for decision-makers and analysts
- Drawing conclusions from charts to support investment or risk strategies