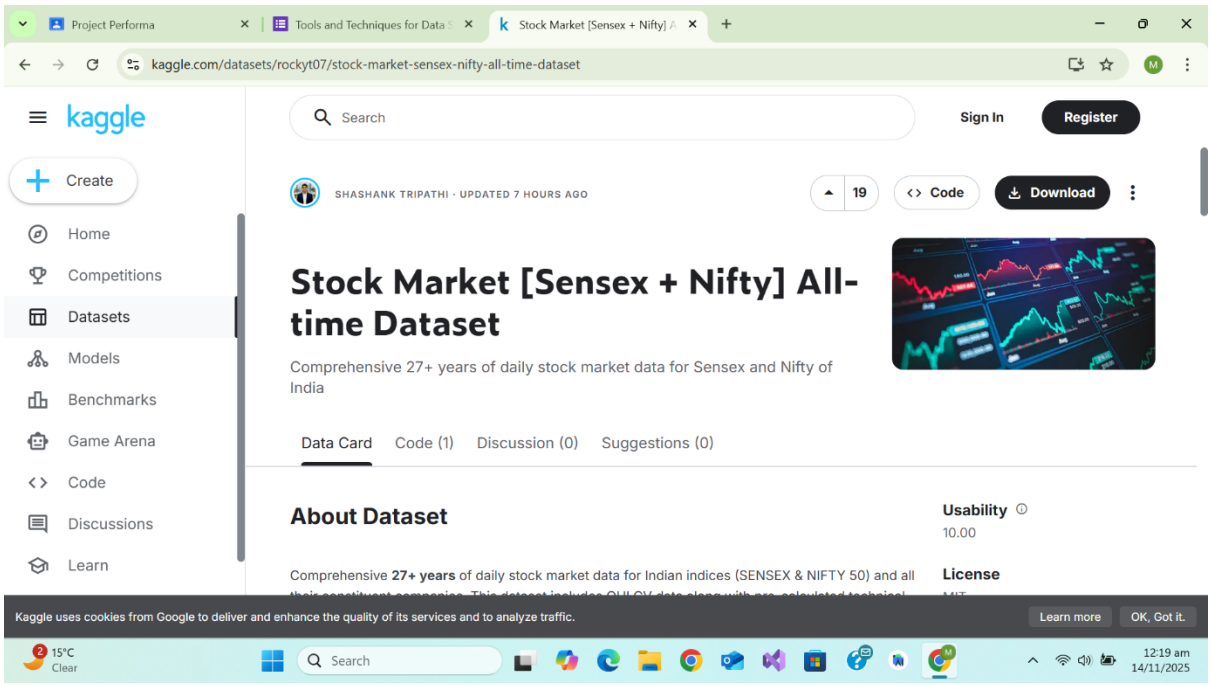
Course Project

Name Muhammad Luqman

Roll Number MSCS25007

Git Hub repo link: https://github.com/Luqman0033/Tools_project

Data set: I get data from Kaggle and that is Stock Market sensex + nifty. Here is link https://www.kaggle.com/datasets/rockyt07/stock-market-sensex-nifty-all-time-dataset?resource=download

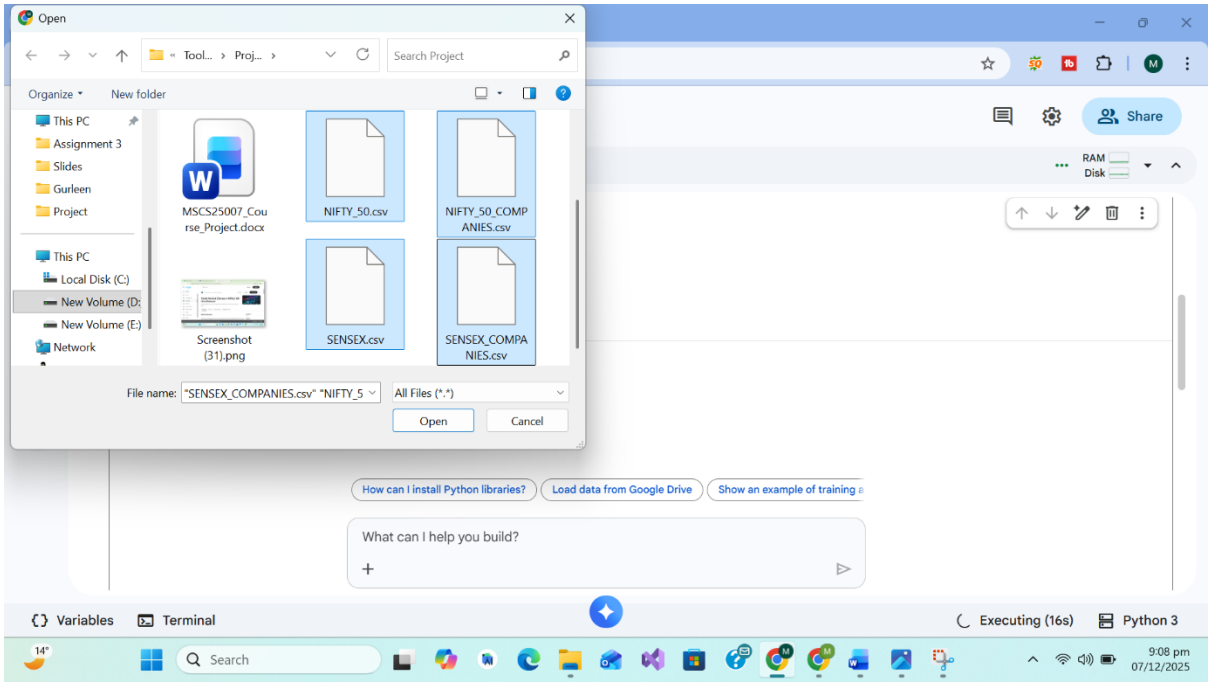


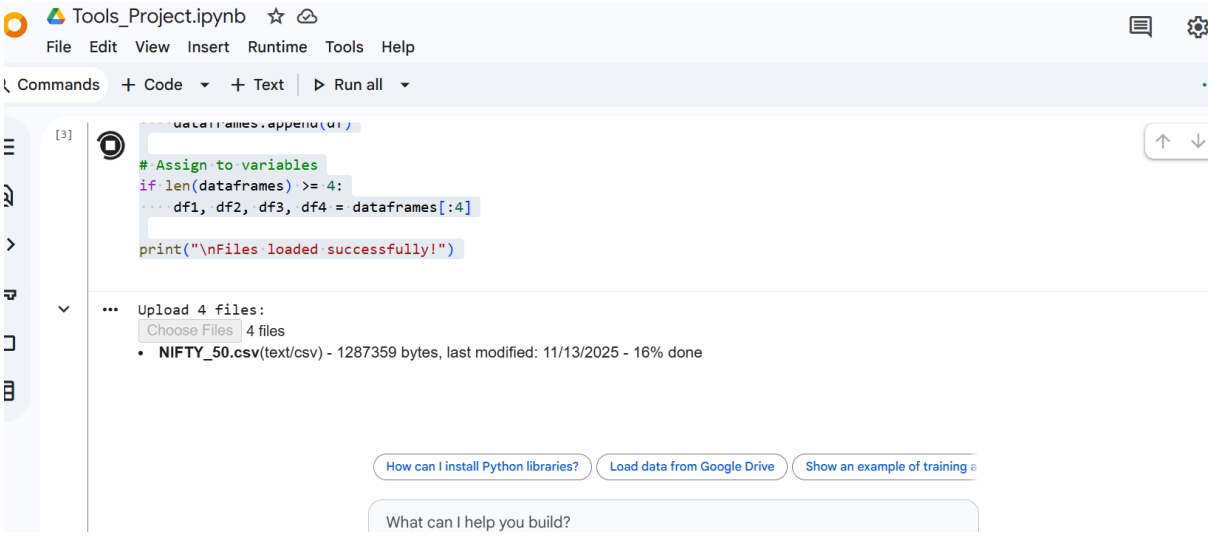
Load Data set:

Code for uploading 4 .csv files

```
from google.colab import files
import pandas as pd

print("Upload 4 files:")
uploaded = files.upload()

dataframes = []

for file_name in uploaded.keys():
```

```
    if file_name.endswith(".csv"):
        df = pd.read_csv(file_name)
    elif file_name.endswith(".xlsx") or file_name.endswith(".xls"):
        df = pd.read_excel(file_name)
    else:
        print(f"Unsupported file: {file_name}")
        continue

    print(f"\nLoaded: {file_name}")
    print(df.head())
    dataframes.append(df)

# Assign to variables
if len(dataframes) >= 4:
    df1, df2, df3, df4 = dataframes[:4]

print("\nFiles loaded successfully!")
```





Data loaded successfully

```
Upload 4 files:
Choose Files 4 files
• NIFTY_50.csv(text/csv) - 1287359 bytes, last modified: 11/13/2025 - 100% done
• NIFTY_50_COMPANIES.csv(text/csv) - 94989759 bytes, last modified: 11/13/2025 - 100% done
• SENSEX.csv(text/csv) - 2034226 bytes, last modified: 11/13/2025 - 100% done
• SENSEX_COMPANIES.csv(text/csv) - 55757483 bytes, last modified: 11/13/2025 - 100% done
Saving NIFTY_50.csv to NIFTY_50 (1).csv
Saving NIFTY_50_COMPANIES.csv to NIFTY_50_COMPANIES.csv
Saving SENSEX.csv to SENSEX.csv
Saving SENSEX_COMPANIES.csv to SENSEX_COMPANIES.csv

Loaded: NIFTY_50 (1).csv
        Date    Adj Close        Close         High          Low  \
0  2007-09-17  4494.649902  4494.649902  4549.049805  4482.850098
1  2007-09-18  4546.200195  4546.200195  4551.799805  4481.549805
2  2007-09-19  4732.350098  4732.350098  4739.000000  4550.250000
3  2007-09-20  4747.549805  4747.549805  4760.850098  4721.149902
4  2007-09-21  4837.549805  4837.549805  4855.700195  4733.700195

          Open  Volume  SMA_20  SMA_50       EMA_12       EMA_26       MACD  \
0  4518.450195       0     NaN     NaN  4494.649902  4494.649902   0.000000
1  4494.100098       0     NaN     NaN  4502.580717  4498.468443   4.112274
```

```
            Open  Volume  SMA_20  SMA_50       EMA_12       EMA_26       MACD  \
0  4518.450195       0     NaN     NaN  4494.649902  4494.649902   0.000000
1  4494.100098       0     NaN     NaN  4502.580717  4498.468443   4.112274
2  4550.250000       0     NaN     NaN  4537.929852  4515.793010  22.136843
3  4734.850098       0     NaN     NaN  4570.179076  4532.960180  37.218896
4  4752.950195       0     NaN     NaN  4611.313034  4555.522374  55.790660

   Signal_Line  RSI_14  BB_Mid  BB_Upper  BB_Lower  Daily_Return_%
0     0.000000     NaN     NaN       NaN       NaN             NaN
1     0.822455     NaN     NaN       NaN       NaN        1.146926
2     5.085332     NaN     NaN       NaN       NaN        4.094626
3    11.512045     NaN     NaN       NaN       NaN        0.321187
4    20.367768     NaN     NaN       NaN       NaN        1.895715

Loaded: NIFTY_50_COMPANIES.csv
         Date  Adj Close      Close       High        Low       Open  \
0  1997-07-01   8.918703  13.056603  13.345798  12.924502  13.281532
1  1997-07-02   9.027228  13.215482  13.456477  13.138720  13.226193
2  1997-07-03   8.907730  13.040537  13.263681  12.999478  13.147646
3  1997-07-04   8.754085  12.815608  13.111943  12.735276  13.076240
4  1997-07-07   8.788230  12.865592  13.040537  12.758483  12.781690
```

```
       Volume       Ticker  SMA_20  SMA_50     EMA_12     EMA_26      MACD  \
0  442137352  RELIANCE.NS     NaN     NaN  13.056603  13.056603  0.000000
1  352633783  RELIANCE.NS     NaN     NaN  13.081046  13.068372  0.012674
2  225940109  RELIANCE.NS     NaN     NaN  13.074814  13.066310  0.008504
3  309913338  RELIANCE.NS     NaN     NaN  13.034936  13.047740 -0.012804
4  275138993  RELIANCE.NS     NaN     NaN  13.008883  13.034247 -0.025364

   Signal_Line  RSI_14  BB_Mid  BB_Upper  BB_Lower  Daily_Return_%
0     0.000000     NaN     NaN       NaN       NaN             NaN
1     0.002535     NaN     NaN       NaN       NaN        1.216843
2     0.003729     NaN     NaN       NaN       NaN       -1.323787
3     0.000422     NaN     NaN       NaN       NaN       -1.724844
4    -0.004735     NaN     NaN       NaN       NaN        0.390024

Loaded: SENSEX.csv
         Date    Adj Close        Close         High          Low  \
0  1997-07-01  4300.859863  4300.859863  4301.770020  4247.660156
1  1997-07-02  4333.899902  4333.899902  4395.310059  4295.399902
2  1997-07-03  4323.459961  4323.459961  4393.290039  4299.970215
3  1997-07-04  4323.819824  4323.819824  4347.589844  4300.580078
4  1997-07-07  4291.450195  4291.450195  4391.009766  4289.490234
```

```
           Open  Volume  SMA_20  SMA_50       EMA_12       EMA_26      MACD  \
0  4263.109863       0     NaN     NaN  4300.859863  4300.859863  0.000000
1  4302.959961       0     NaN     NaN  4305.942946  4303.307274  2.635673
2  4335.790039       0     NaN     NaN  4308.637872  4304.800065  3.837806
3  4332.700195       0     NaN     NaN  4310.973557  4306.208936  4.764620
4  4326.810059       0     NaN     NaN  4307.969963  4305.115696  2.854266

   Signal_Line  RSI_14  BB_Mid  BB_Upper  BB_Lower  Daily_Return_%
0     0.000000     NaN     NaN       NaN       NaN             NaN
1     0.527135     NaN     NaN       NaN       NaN        0.768219
2     1.189269     NaN     NaN       NaN       NaN       -0.240890
3     1.904339     NaN     NaN       NaN       NaN        0.008324
4     2.094325     NaN     NaN       NaN       NaN       -0.748635

Loaded: SENSEX_COMPANIES.csv
         Date  Adj Close      Close       High        Low       Open  Volume  \
0  2000-01-04   8.775805  33.612499  33.612499  33.612499  33.612499   12800
1  2000-01-05   8.876978  34.000000  34.000000  34.000000  34.000000  837136
2  2000-01-12   9.497059  36.375000  36.375000  36.375000  36.375000    8400
3  2000-02-07  11.329567  43.393749  43.393749  43.393749  43.393749   16000
4  2000-02-14  11.748938  45.000000  45.000000  45.000000  45.000000    4000

        Ticker  SMA_20  SMA_50     EMA_12     EMA_26      MACD  Signal_Line  \
0  RELIANCE.BO     NaN     NaN  33.612499  33.612499  0.000000     0.000000
```

Toggle Gemini

```
1  2000-01-05   8.876978  34.000000  34.000000  34.000000  34.000000  837136
2  2000-01-12   9.497059  36.375000  36.375000  36.375000  36.375000    8400
3  2000-02-07  11.329567  43.393749  43.393749  43.393749  43.393749   16000
4  2000-02-14  11.748938  45.000000  45.000000  45.000000  45.000000    4000

        Ticker  SMA_20  SMA_50     EMA_12     EMA_26      MACD  Signal_Line  \
0  RELIANCE.BO     NaN     NaN  33.612499  33.612499  0.000000     0.000000
1  RELIANCE.BO     NaN     NaN  33.672115  33.641203  0.030912     0.006182
2  RELIANCE.BO     NaN     NaN  34.087943  33.843706  0.244237     0.053793
3  RELIANCE.BO     NaN     NaN  35.519606  34.551117  0.968489     0.236732
4  RELIANCE.BO     NaN     NaN  36.978128  35.325108  1.653020     0.519990

   RSI_14  BB_Mid  BB_Upper  BB_Lower  Daily_Return_%
0     NaN     NaN       NaN       NaN             NaN
1     NaN     NaN       NaN       NaN        1.152847
2     NaN     NaN       NaN       NaN        6.985294
3     NaN     NaN       NaN       NaN       19.295531
4     NaN     NaN       NaN       NaN        3.701572

Files loaded successfully!
```

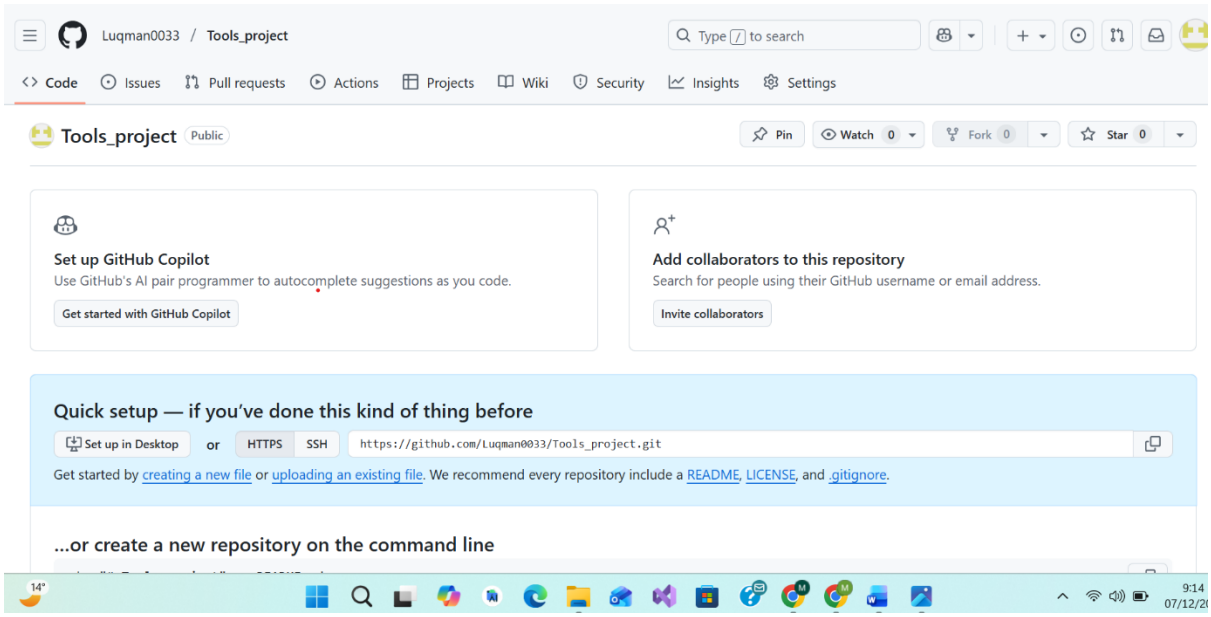
Dataset Description:

The dataset used in this project contains historical stock market data of the Indian stock indices **Sensex** and **Nifty**. It includes information such as opening price, closing price, highest price, lowest price, and trading volume recorded over multiple dates. This dataset is

useful for analyzing market trends, price movements, and performing predictive analysis using machine learning techniques. The data helps in understanding the behavior of stock markets over time.

Git and github use:

Created github repo



Connecting my google colab with github

```
%cd Tools_project
```

```
/content/Tools_project
```

```
!git add .
```

```
!git commit -m "Added project notebook and data"
```

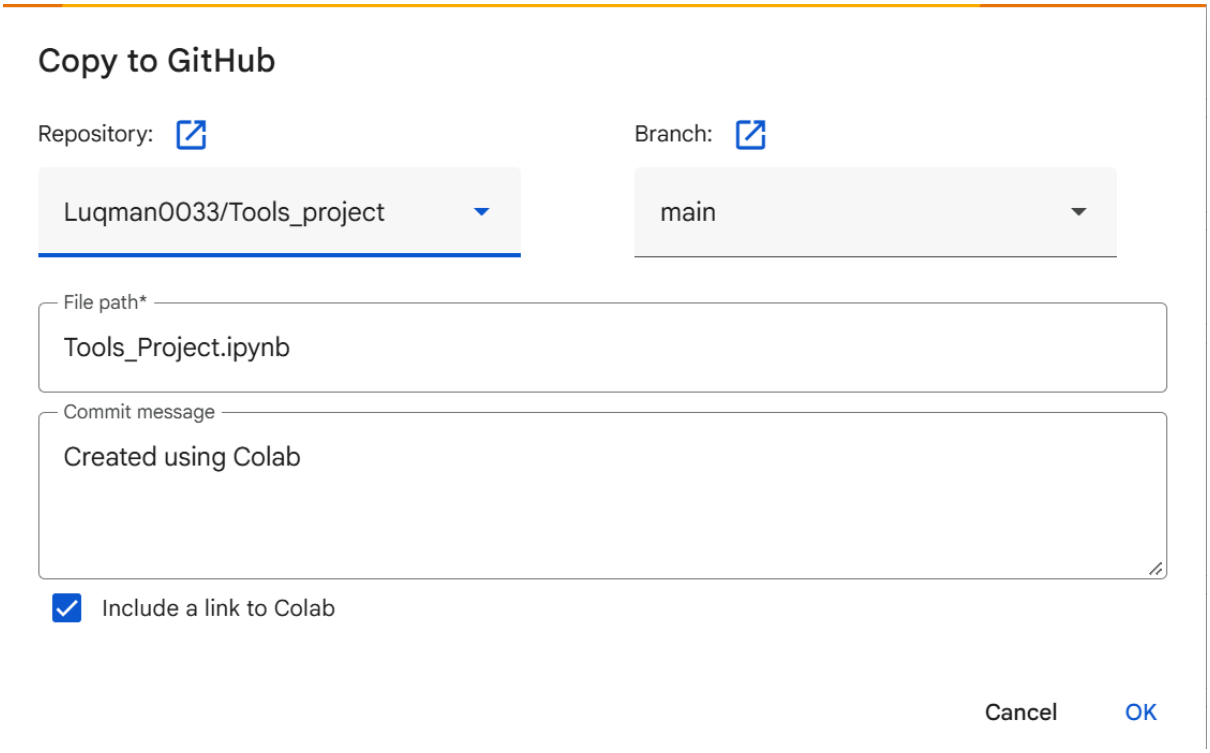
```
On branch main

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Here i initiate git in my assignment folder at pc and push assignment file on git hub

```
Microsoft Windows [Version 10.0.26200.7171]
(c) Microsoft Corporation. All rights reserved.

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git init
Initialized empty Git repository in D:/MSCS/Semester 1/Tools and Techniques for Data science/Project/.git/

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git add MSCS25007_Course_Project.docx

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   MSCS25007_Course_Project.docx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1.png
        10.png
        11.png
        12.png
        2.png
        3.png
        4.png
        5.png
        6.png
        7.png
        8.png
        9.png
        CS 591_Course Project.pdf
        NIFTY_50.csv
```

```
D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git commit -m
error: switch `m' requires a value

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git commit
Aborting commit due to empty commit message.

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>git commit -m "first"
[master (root-commit) d9a09fb] first
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 MSCS25007_Course_Project.docx

D:\MSCS\Semester 1\Tools and Techniques for Data science\Project>
```

Here i push all the code of colab to github



EDA using Python:

df.shape

Explanation :

This step shows the number of rows and columns in the dataset. It helps understand the size and structure of the data.

```
df.shape
```

```
(180327, 19)
```

df.info()

Explanation:

This step displays the data types of each column and shows whether any missing values exist in the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180327 entries, 0 to 180326
Data columns (total 19 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Date            180327 non-null  object
 1   Adj Close       180327 non-null  float64
 2   Close           180327 non-null  float64
 3   High            180327 non-null  float64
 4   Low             180327 non-null  float64
 5   Open            180327 non-null  float64
 6   Volume          180327 non-null  int64
 7   Ticker          180327 non-null  object
 8   SMA_20          179757 non-null  float64
 9   SMA_50          178857 non-null  float64
 10  EMA_12          180327 non-null  float64
 11  EMA_26          180327 non-null  float64
 12  MACD            180327 non-null  float64
 13  Signal_Line     180327 non-null  float64
 14  RSI_14          176862 non-null  float64
 15  BB_Mid          179757 non-null  float64
 16  BB_Upper        179757 non-null  float64
 17  BB_Lower        179757 non-null  float64
 18  Daily_Return_%  180297 non-null  float64
dtypes: float64(16), int64(1), object(2)
memory usage: 26.1+ MB
```

df.describe()

Explanation:

This provides statistical information such as mean, median, minimum, maximum, and standard deviation for numerical columns.

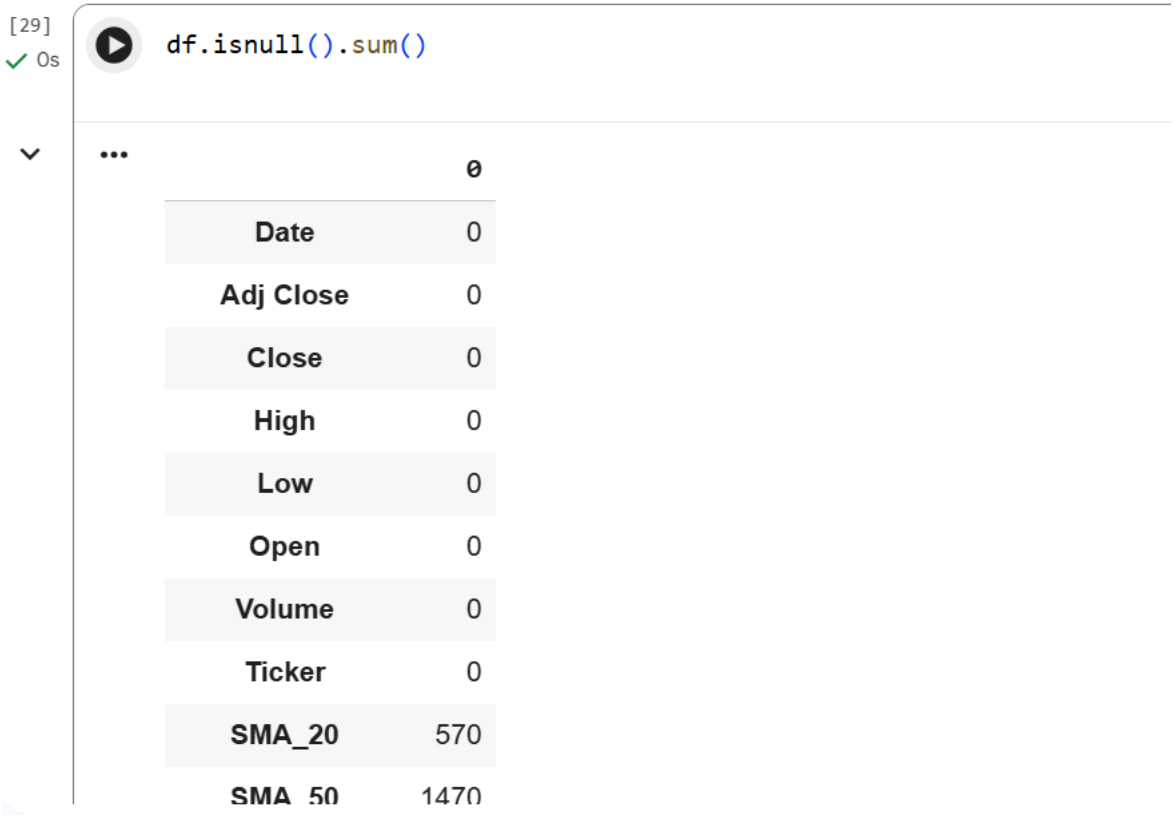
```
df.describe()
```

	Adj Close	Close	High	Low	Open	Volume	SMA_20	SMA_50	EMA_12	
count	180327.000000	180327.000000	180327.000000	180327.000000	180327.000000	1.803270e+05	179757.000000	178857.000000	180327.000000	1803
mean	621.201401	681.060135	690.059107	673.979391	682.392122	2.073956e+06	679.374946	676.720826	678.962112	6
std	1342.960701	1369.242662	1383.382975	1355.159496	1369.983798	9.306633e+06	1362.549560	1351.689298	1364.207686	13
min	-5.642860	0.273000	0.273000	0.273000	0.273000	0.000000e+00	0.273000	0.279130	0.273037	
25%	32.404440	75.269997	78.068748	75.375000	76.756565	1.010570e+05	75.443874	75.783100	75.045942	
50%	173.469299	237.500000	242.705002	235.899994	239.500000	3.633800e+05	237.465001	237.935800	236.738414	2
75%	632.610779	714.750000	724.000000	706.177521	715.505554	1.281944e+06	713.769998	710.810200	712.489593	7
max	16432.599609	16432.599609	16673.900391	16384.750000	16673.900391	1.589245e+09	16218.654980	15794.320020	16285.742855	160

df.isnull().sum()

Explanation:

This step counts the number of missing values in each column so that data quality issues can be identified.



Ticker	0
SMA_20	570
SMA_50	1470
EMA_12	0
EMA_26	0
MACD	0
Signal_Line	0
RSI_14	3465
BB_Mid	570
BB_Upper	570
BB_Lower	570
Daily_Return_%	30

dtype: int64

df.duplicated().sum()

Explanation:

This checks whether there are duplicate records in the dataset.

```
df.duplicated().sum()
```

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

df['Ticker'].value_counts()

Explanation:

This step counts the frequency of unique values in a categorical column.

```
df['Ticker'].value_counts()
```

Ticker	count
HDFCBANK.BO	6447
NESTLEIND.BO	6447
TATASTEEL.BO	6428
TITAN.BO	6422
TATAMOTORS.BO	6421
ITC.BO	6421

TATASTEEL.BO	6428
TITAN.BO	6422
TATAMOTORS.BO	6421
ITC.BO	6421
M&M.BO	6421
SBIN.BO	6420
SUNPHARMA.BO	6420
ASIANPAINT.BO	6420
INFY.BO	6406
WIPRO.BO	6389
HCLTECH.BO	6378
HINDUNILVR.BO	6368
ONGC.BO	6367

KOTAKBANK.BO	6365
LT.BO	6343
INDUSINDBK.BO	6334
BAJFINANCE.BO	6327
AXISBANK.BO	6321
RELIANCE.BO	6250
BHARTIARTL.BO	5866
ICICIBANK.BO	5816
MARUTI.BO	5505
TCS.BO	5212
ULTRACEMCO.BO	5210
NTPC.BO	5160
TECHM.BO	4706

ULTRACEMCO.BO	5210
NTPC.BO	5160
TECHM.BO	4706
POWERGRID.BO	4454
BAJAJFINSV.BO	4283

dtype: int64

Data Visualizations (EDA Charts)

Histogram

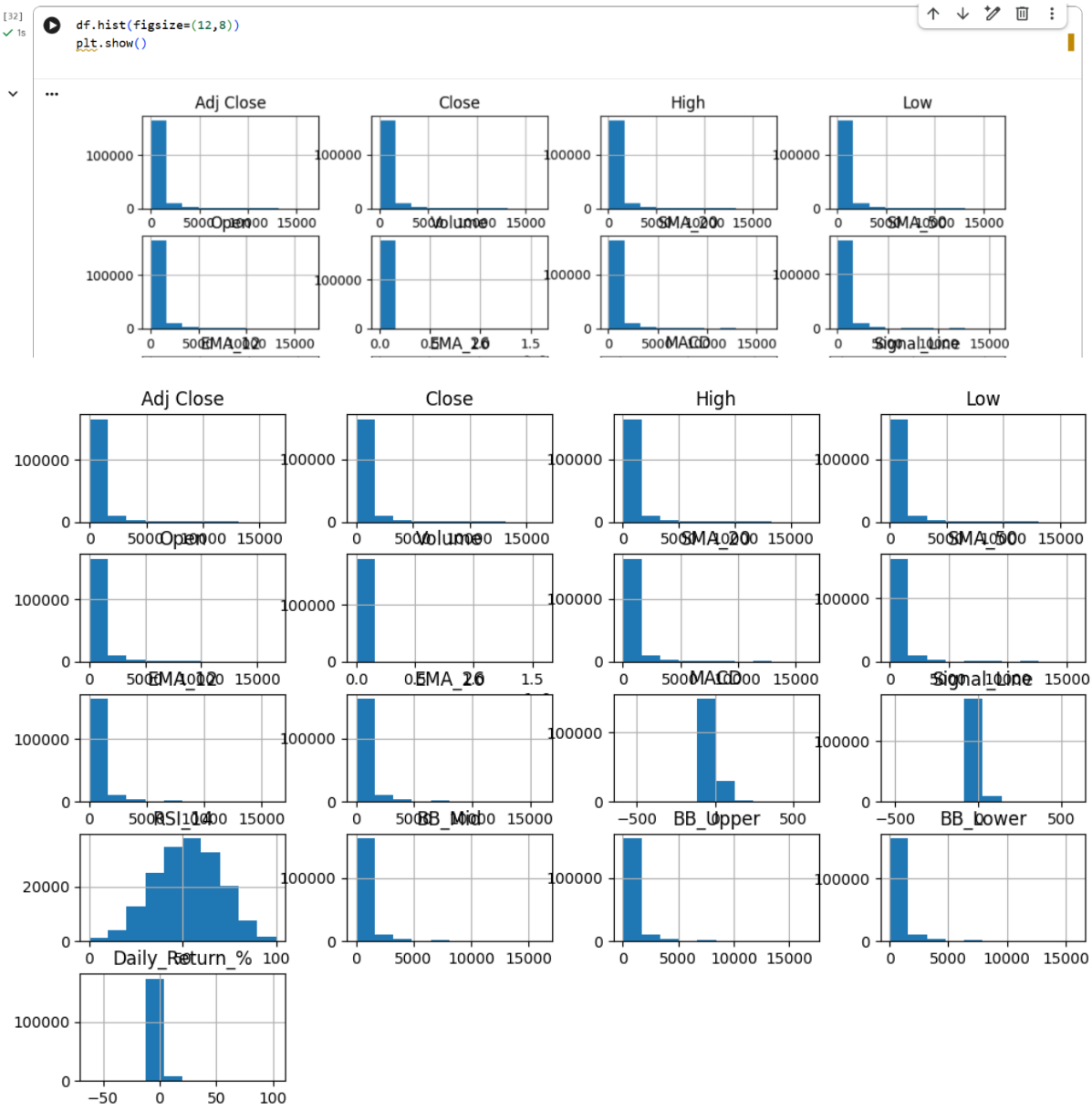
df.hist(figsize=(12,8))

plt.show()

Explanation:

Histograms show the distribution of numerical data and help identify skewness and outliers.

```
[32]
df.hist(figsize=(12,8))
plt.show()
```



Box Plot

plt.figure(figsize=(10,6))

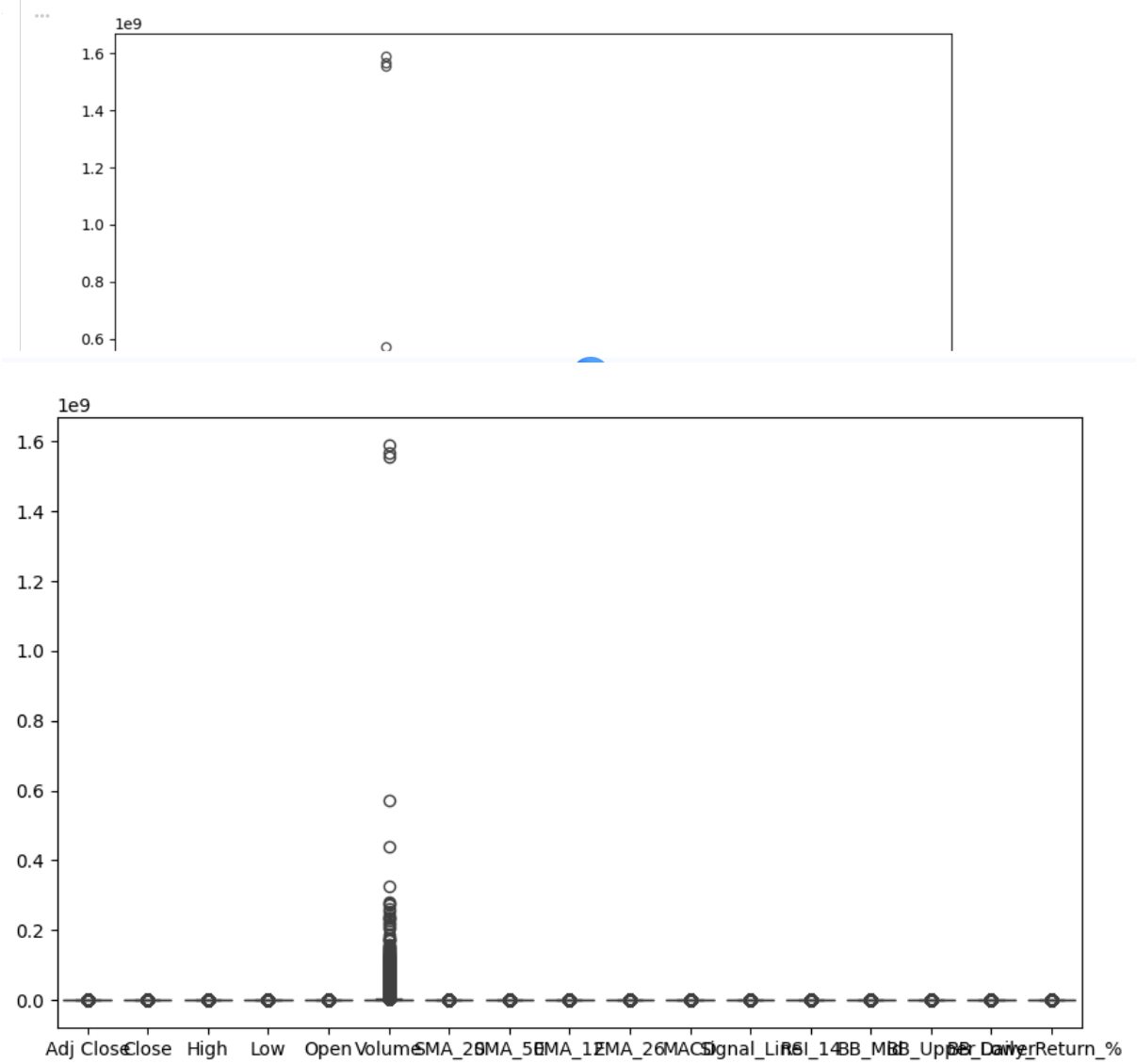
sns.boxplot(data=df)

plt.show()

Explanation:

Box plots help detect outliers and visualize the spread of the data.

```
plt.figure(figsize=(10,6))
sns.boxplot(data=df)
plt.show()
```



Correlation Matrix

plt.figure(figsize=(10,6))

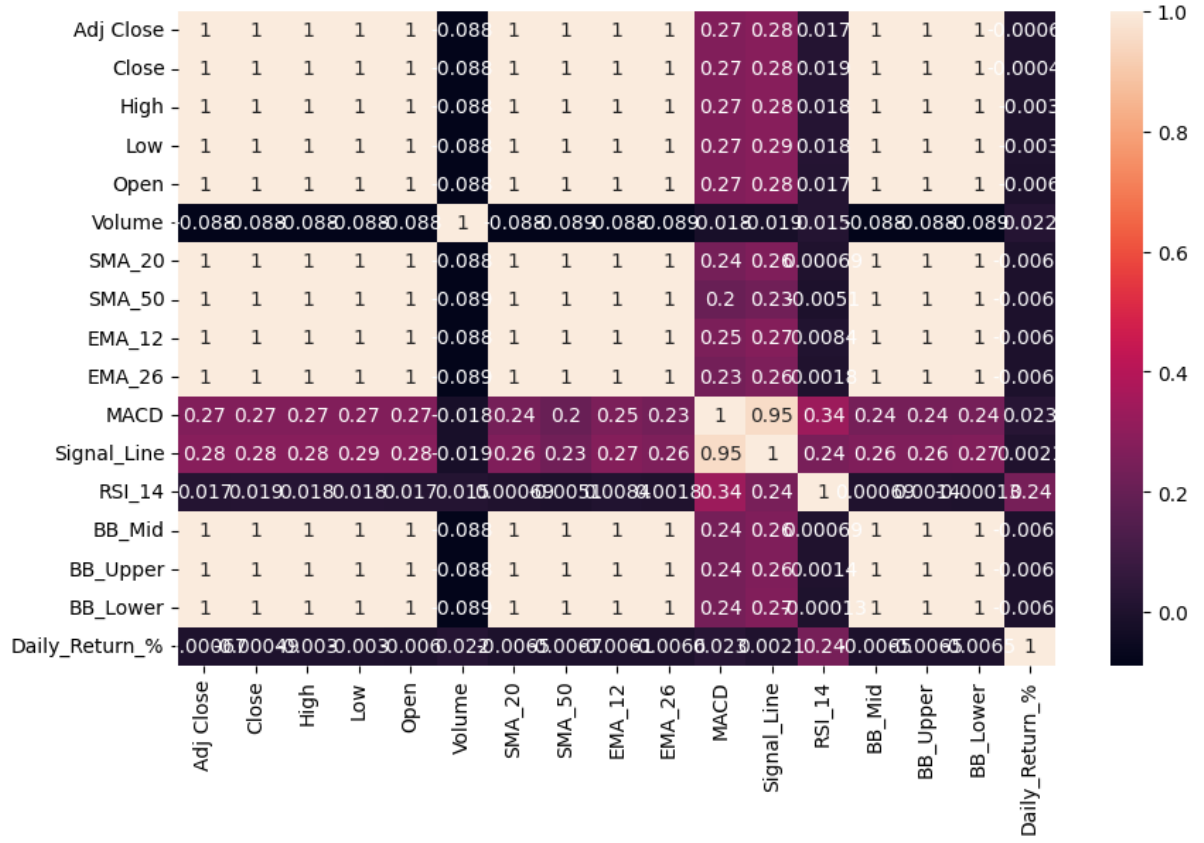
sns.heatmap(df.corr(), annot=True)

plt.show()

Explanation:

This step shows the correlation between numerical variables and helps in feature selection for machine learning.

```
[35]
✓ 1s
plt.figure(figsize=(10,6))
sns.heatmap(df.select_dtypes(include=['number']).corr(), annot=True)
plt.show()
```



Data Wrangling / Cleansing:

1. **Locate and load files**
    - Script searches the current working directory for CSV/XLSX files and loads up to 4 files.
    - Each file is read using pandas.read_csv or pandas.read_excel depending on the extension.
2. **Standardize column names**

- Column names are lowercased, trimmed, and spaces replaced with underscores to make later code robust to inconsistencies.

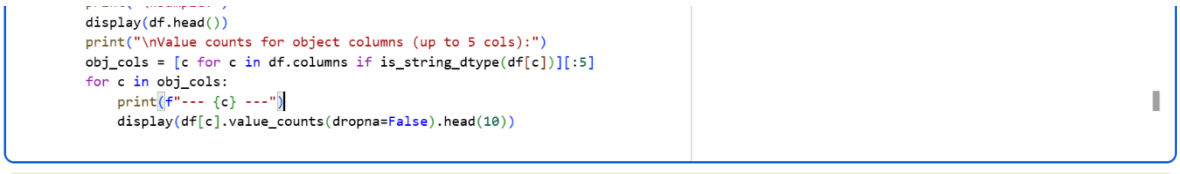
3. **Parse dates and set index**
    - The script attempts to find a sensible date column (common names such as date, timestamp) and converts it to datetime.
    - Rows with unparseable dates are dropped and the DataFrame is indexed by date and sorted.
    - This is needed for time-series operations (rolling windows, resampling, etc.).
4. **Convert numeric columns**
    - Non-numeric text (like commas in numbers) is cleaned and numeric columns coerced to numeric types.
    - Ensures arithmetic operations and statistical summaries work correctly.
5. **Merge files (if multiple)**
    - If multiple files with datetime indexes are present, they are merged on the Date index (outer join) with column name prefixes to avoid collisions.
    - This allows combining different sources (for example Sensex file + Nifty file).
6. **Remove duplicate rows**
    - Duplicate date rows are removed to keep each timestamp unique (keeps first occurrence).
7. **Handle missing values**
    - Missing values are filled using time-based interpolation followed by forward/backward filling.
    - Alternate strategies (drop, fill with mean/mode) are available depending on the situation.
    - Interpolation maintains time-series continuity without introducing abrupt jumps.
8. **Outlier detection and capping**
    - Outliers are detected using the IQR method and then capped (clipped) to q1 - 1.5*IQR and q3 + 1.5*IQR.
    - Capping reduces the effect of extreme values while preserving most of the data.
9. **Feature engineering**
    - Create returns (pct_change), log returns, rolling means (7, 21 days), rolling std (21 days), annualized volatility proxy, and lag features (1,2,3,5,10).
    - These features are commonly useful for financial analysis and machine learning models.
10. **Scaling**

- Numeric features are scaled using StandardScaler (zero mean, unit variance) to prepare data for many machine learning algorithms.
- MinMax scaling can be used alternatively when needed.

11. **Save cleaned data**
    - The final cleaned and scaled DataFrame is saved to cleaned_merged_data.csv so you can reuse it for modeling and charts.

```
# Cell — Imports & helper utilities
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
from pandas.api.types import is_numeric_dtype, is_string_dtype, is_datetime64_any_dtype
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
import re
import seaborn as sns
sns.set(style="whitegrid", rc={'figure.figsize':(8,4)})

# Helper: convert column names to snake_case and strip
def clean_column_names(df):
    df = df.copy()
    def to_snake(s):
        s = str(s).strip()
        s = re.sub(r'[^\w\s]', '', s)       # remove punctuation
        s = re.sub(r'\s+', '_', s)          # spaces to _
        s = re.sub(r'(?<!^)(?=[A-Z])', '_', s)  # camelCase -> snake_case
        return s.lower()
    df.columns = [to_snake(c) for c in df.columns]
    return df
```

```
# Helper: quick summary
def quick_summary(df, name="df"):
    print(f"=== Summary for {name} ===")
    print("Rows, cols:", df.shape)
    print("\nColumn types:")
    display(pd.DataFrame(df.dtypes, columns=["dtype"]))
    print("\nMissing value counts (top 20):")
    display(df.isna().sum().sort_values(ascending=False).head(20))
    print("\nSample:")
    display(df.head())
    print("\nValue counts for object columns (up to 5 cols):")
    obj_cols = [c for c in df.columns if is_string_dtype(df[c])][:5]
    for c in obj_cols:
        print(f"--- {c} ---")
        display(df[c].value_counts(dropna=False).head(10))
```



Cell — Clean column names & apply to all dataframes

```
# Cell 2 - Clean column names for each df (if they exist)
for i, name in enumerate(['df1','df2','df3','df4'], start=1):
    if name in globals():
        globals()[name] = clean_column_names(globals()[name])
        print(f"Cleaned columns for {name}: {globals()[name].shape}")
    else:
        print(f"{name} not found in environment")


# Quick check
for name in ['df1','df2','df3','df4']:
    if name in globals():
```

```
    print("\n", name, "columns:", globals()[name].columns.tolist()[:10])
```


```
# Cell 3 - Quick summaries for each df
for name in ['df1','df2','df3','df4']:
    if name in globals():
        quick_summary(globals()[name], name=name)
    else:
        print(f"{name} not available")
```

```
print(f"{name} not available")
```

	dtype
date	object
adj__close	float64
close	float64
high	float64
low	float64
open	float64
volume	int64
s_m_a_20	float64
s_m_a_50	float64
e_m_a_12	float64
e_m_a_26	float64

Missing value counts (top 20):

	0
s_m_a_50	49
s_m_a_20	19
b_b__mid	19
b_b__upper	19
b_b__lower	19
r_s_i_14	13
daily__return_	1
date	0
adj__close	0
close	0

dtype: int64

Sample:

	date	adj__close	close	high	low	open	volume	s_m_a_20	s_m_a_50	e_m_a_12	e_m_a_26	m_a_c_d	sign
0	2007-09-17	4494.649902	4494.649902	4549.049805	4482.850098	4518.450195	0	NaN	NaN	4494.649902	4494.649902	0.000000	
1	2007-09-18	4546.200195	4546.200195	4551.799805	4481.549805	4494.100098	0	NaN	NaN	4502.580717	4498.468443	4.112274	
2	2007-09-19	4732.350098	4732.350098	4739.000000	4550.250000	4550.250000	0	NaN	NaN	4537.929852	4515.793010	22.136843	
3	2007-09-20	4747.549805	4747.549805	4760.850098	4721.149902	4734.850098	0	NaN	NaN	4570.179076	4532.960180	37.218896	1
4	2007-09-21	4837.549805	4837.549805	4855.700195	4733.700195	4752.950195	0	NaN	NaN	4611.313034	4555.522374	55.790660	2

Value counts for object columns (up to 5 cols):

```
Value counts for object columns (up to 5 cols):
--- date ---
```

date	count
2025-11-11	1
2007-09-17	1
2007-09-18	1
2007-09-19	1
2025-09-12	1
2025-09-15	1
2025-09-16	1
2025-09-17	1

dtype: int64

```
=== Summary for df2 ===
Rows, cols: (304543, 19)

Column types:
```

	dtype
date	object
adj__close	float64
close	float64
high	float64
low	float64
open	float64
volume	int64
ticker	object

```
Missing value counts (top 20):
```

	0
r_s_i_14	2697
s_m_a_50	2499
b_b__lower	969
b_b__mid	969
b_b__upper	969
s_m_a_20	969
daily__return_	51
close	0
adj__close	0
date	0

dtype: int64

Sample:

	date	adj__close	close	high	low	open	volume	ticker	s_m_a_20	s_m_a_50	e_m_a_12	e_m_a_26	m_a_c_d
0	1997-07-01	8.918703	13.056603	13.345798	12.924502	13.281532	442137352	RELIANCE.NS	NaN	NaN	13.056603	13.056603	0.000000
1	1997-07-02	9.027228	13.215482	13.456477	13.138720	13.226193	352633783	RELIANCE.NS	NaN	NaN	13.081046	13.068372	0.012674
2	1997-07-03	8.907730	13.040537	13.263681	12.999478	13.147646	225940109	RELIANCE.NS	NaN	NaN	13.074814	13.066310	0.008504
3	1997-07-04	8.754085	12.815608	13.111943	12.735276	13.076240	309913338	RELIANCE.NS	NaN	NaN	13.034936	13.047740	-0.012804
4	1997-07-07	8.788230	12.865592	13.040537	12.758483	12.781690	275138993	RELIANCE.NS	NaN	NaN	13.008883	13.034247	-0.025364

Value counts for object columns (up to 5 cols):

```
Value counts for object columns (up to 5 cols):
--- date ---
```

date	count
2025-10-07	51
2025-11-11	51
2025-09-15	51
2025-09-16	51
2025-09-17	51
2025-09-18	51
2025-09-19	51
2025-09-22	51
2025-09-23	51

ticker	count
INFY.NS	7108
HINDUNILVR.NS	7108
HDFCBANK.NS	7108
TITAN.NS	7108
SUNPHARMA.NS	7108
WIPRO.NS	7108
TATASTEEL.NS	7108
BRITANNIA.NS	7108
HINDALCO.NS	7108
TATAMOTORS.NS	7108

dtype: int64

dtype: int64

```
=== Summary for df3 ===
Rows, cols: (6987, 18)

Column types:
```

	dtype
date	object
adj__close	float64
close	float64
high	float64
low	float64
open	float64
volume	int64
s_m_a_20	float64
s_m_a_50	float64

```
Missing value counts (top 20):
```

	0
s_m_a_50	49
s_m_a_20	19
b_b__mid	19
b_b__upper	19
b_b__lower	19
r_s_i_14	13
daily__return_	1
date	0
adj__close	0
close	0

dtype: int64

Sample:

	date	adj__close	close	high	low	open	volume	s_m_a_20	s_m_a_50	e_m_a_12	e_m_a_26	m_a_c_d	signa
0	1997-07-01	4300.859863	4300.859863	4301.770020	4247.660156	4263.109863	0	NaN	NaN	4300.859863	4300.859863	0.000000	0
1	1997-07-02	4333.899902	4333.899902	4395.310059	4295.399902	4302.959961	0	NaN	NaN	4305.942946	4303.307274	2.635673	0
2	1997-07-03	4323.459961	4323.459961	4393.290039	4299.970215	4335.790039	0	NaN	NaN	4308.637872	4304.800065	3.837806	1
3	1997-07-04	4323.819824	4323.819824	4347.589844	4300.580078	4332.700195	0	NaN	NaN	4310.973557	4306.208936	4.764620	1
4	1997-07-07	4291.450195	4291.450195	4391.009766	4289.490234	4326.810059	0	NaN	NaN	4307.969963	4305.115696	2.854266	2

Value counts for object columns (up to 5 cols):
--- date ---
count

date	
2025-11-11	1
1997-07-01	1
1997-07-02	1
1997-07-03	1
1997-07-04	1
1997-07-07	1
1997-07-08	1
1997-07-09	1
1997-07-10	1
1997-07-11	1

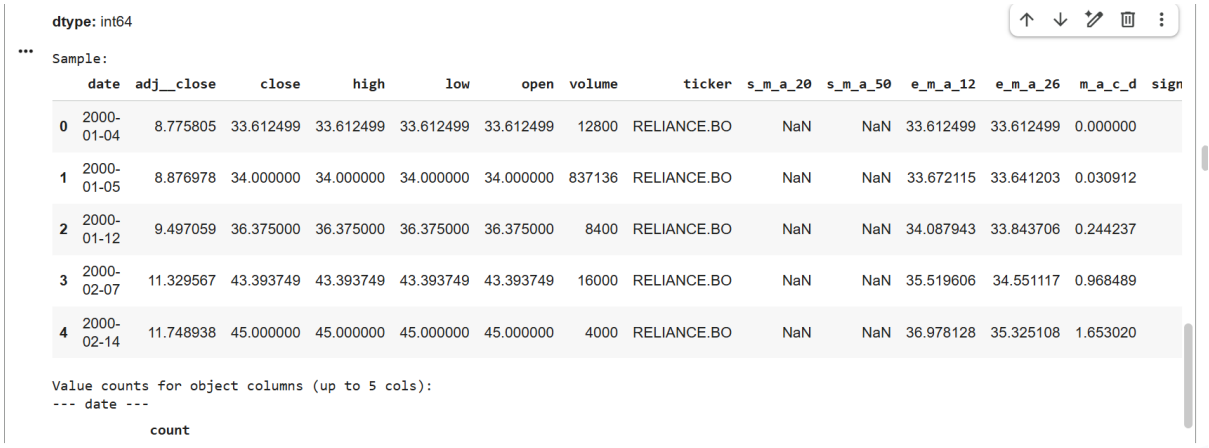
dtype: int64
=== Summary for df4 ===
Rows, cols: (180327, 19)

Column types:

	dtype
date	object
adj__close	float64
close	float64
high	float64
low	float64
open	float64
volume	int64
ticker	object
s_m_a_20	float64
s_m_a_50	float64
e_m_a_12	float64

Missing value counts (top 20):

	0
r_s_i_14	3465
s_m_a_50	1470
b_b__lower	570
b_b__mid	570
b_b__upper	570
s_m_a_20	570
daily__return_	30
close	0
adj__close	0
date	0
high	0
e_m_a_12	0



dtype: int64

Sample:

	date	adj__close	close	high	low	open	volume	ticker	s_m_a_20	s_m_a_50	e_m_a_12	e_m_a_26	m_a_c_d	sign
0	2000-01-04	8.775805	33.612499	33.612499	33.612499	33.612499	12800	RELIANCE.BO	NaN	NaN	33.612499	33.612499	0.000000	
1	2000-01-05	8.876978	34.000000	34.000000	34.000000	34.000000	837136	RELIANCE.BO	NaN	NaN	33.672115	33.641203	0.030912	
2	2000-01-12	9.497059	36.375000	36.375000	36.375000	36.375000	8400	RELIANCE.BO	NaN	NaN	34.087943	33.843706	0.244237	
3	2000-02-07	11.329567	43.393749	43.393749	43.393749	43.393749	16000	RELIANCE.BO	NaN	NaN	35.519606	34.551117	0.968489	
4	2000-02-14	11.748938	45.000000	45.000000	45.000000	45.000000	4000	RELIANCE.BO	NaN	NaN	36.978128	35.325108	1.653020	

Value counts for object columns (up to 5 cols):
--- date ---
count

Cell 4 - Trim whitespace in string columns and turn empty strings into NaN

```
def tidy_strings(df):
    for c in df.select_dtypes(include=['object', 'string']).columns:
        df[c] = df[c].astype(str).str.strip()
        # convert common empty markers to NaN
        df[c] = df[c].replace({'': np.nan, 'nan': np.nan, 'None': np.nan, 'NULL': np.nan, 'null': np.nan})
    return df

for name in ['df1','df2','df3','df4']:
    if name in globals():
        globals()[name] = tidy_strings(globals()[name])
        print(f"Trimmed strings for {name}")
```

```
print(f"Trimmed strings for {name}")
```

```
Trimmed strings for df1
Trimmed strings for df2
Trimmed strings for df3
Trimmed strings for df4
```

```
!git --version
```

Cell 5 - Auto-detect and parse date columns

```
import dateutil

def try_parse_dates(df, sample_n=50):
    df = df.copy()
```

```
    for c in df.columns:
        if df[c].dtype == object:
            # sample non-null values and try parse
            sample = df[c].dropna().astype(str).head(sample_n)
            if sample.empty:
                continue
            success = 0
            for v in sample:
                try:
                    _ = dateutil.parser.parse(v, fuzzy=False)
                    success += 1
                except Exception:
                    pass
            # if a high fraction parsed -> convert
            if success / len(sample) >= 0.6:
                try:
                    df[c] = pd.to_datetime(df[c], errors='coerce', dayfirst=False)
                    print(f"Parsed {c} as datetime (converted {df[c].notna().sum()} non-null)")
                except Exception as e:
                    print("Failed parse ", c, e)
    return df

for name in ['df1','df2','df3','df4']:
    if name in globals():
        globals()[name] = try_parse_dates(globals()[name])
```

```
            globals()[name] = try_parse_dates(globals()[name])
```

```
...  Parsed date as datetime (converted 4452 non-null)
     Parsed date as datetime (converted 304543 non-null)
     Parsed date as datetime (converted 6987 non-null)
     Parsed date as datetime (converted 180327 non-null)
```

```
# Cell 6 - Convert numeric-like object columns to numeric
```

```
def numericify(df):
    for c in df.columns:
        if df[c].dtype == object:
            # remove common formatting characters and try convert
            sample = df[c].dropna().astype(str).head(50)
            if sample.empty:
                continue
            # detect if majority of sample looks numeric (after removing comma, percent, $)
            cleaned = sample.str.replace(r'[,\$\%]', '', regex=True).str.replace(r'\s+', '', regex=True)
            numeric_count = cleaned.apply(lambda x: bool(re.match(r'^-?\d+(\.\d+)?$', x)))
            if numeric_count.mean() >= 0.6:
                df[c] = pd.to_numeric(df[c].astype(str).str.replace(r'[,\$\%]', '', regex=True), errors='coerce')
                print(f"Converted {c} -> numeric (non-null: {df[c].notna().sum()})")
    return df

for name in ['df1','df2','df3','df4']:
    if name in globals():
        globals()[name] = numericify(globals()[name])



# Cell 7 - Duplicates
for name in ['df1','df2','df3','df4']:
    if name in globals():
        df = globals()[name]
        dup_count = df.duplicated().sum()
        print(f"{name}: {dup_count} exact duplicate rows")
        if dup_count > 0:
            print(df[df.duplicated()].head())
            # drop duplicates in place
            globals()[name] = df.drop_duplicates().reset_index(drop=True)
```

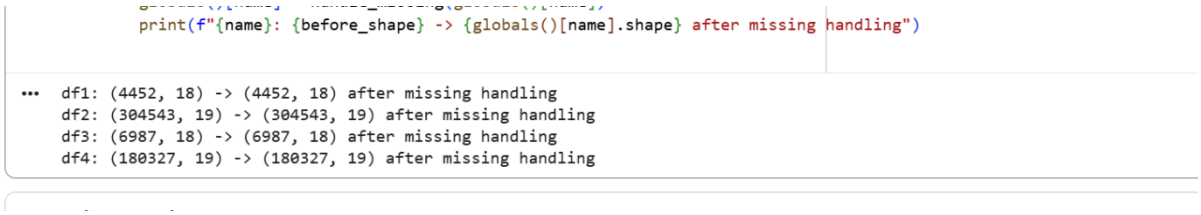
print(f"Dropped duplicates in {name}. New shape: {globals()[name].shape}")

Cell 8 - Missing value handling: drop columns with >60% missing, impute the rest sensibly

DROP_COL_THRESHOLD = 0.60

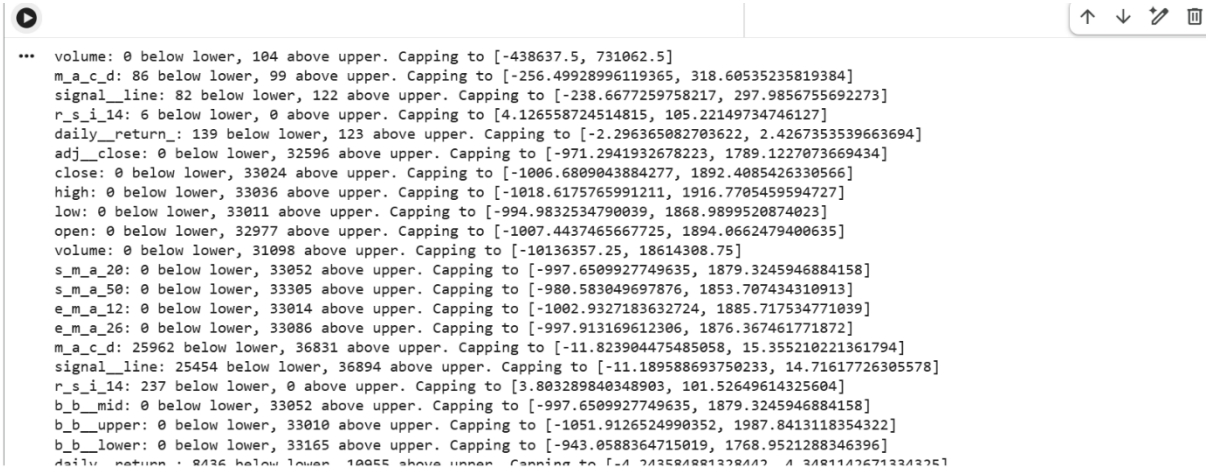
```
def handle_missing(df, drop_thresh=DROP_COL_THRESHOLD):
    df = df.copy()
    missing_frac = df.isna().mean()
    to_drop = missing_frac[missing_frac > drop_thresh].index.tolist()
    if to_drop:
        print("Dropping columns (too many missing):", to_drop)
        df.drop(columns=to_drop, inplace=True)
    # Impute numeric with median
    num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    if num_cols:
        median_imputer = SimpleImputer(strategy='median')
        df[num_cols] = pd.DataFrame(median_imputer.fit_transform(df[num_cols]),
columns=num_cols)
    # Impute categorical (object) with mode
    obj_cols = df.select_dtypes(include=['object','string']).columns.tolist()
    for c in obj_cols:
        if df[c].isna().any():
            mode = df[c].mode(dropna=True)
            if not mode.empty:
                df[c].fillna(mode[0], inplace=True)
            else:
```

```
            df[c].fillna("unknown", inplace=True)
    return df


for name in ['df1','df2','df3','df4']:
    if name in globals():
        before_shape = globals()[name].shape
        globals()[name] = handle_missing(globals()[name])
        print(f"{name}: {before_shape} -> {globals()[name].shape} after missing handling")
```




```
# Cell 9 - Outlier capping using IQR winsorization
def cap_outliers_iqr(df, cols=None, multiplier=1.5):
    df = df.copy()
    if cols is None:
        cols = df.select_dtypes(include=[np.number]).columns.tolist()
    for c in cols:
        if c not in df.columns:
            continue
        col = df[c].dropna()
        if col.empty:
            continue
        q1 = col.quantile(0.25)
        q3 = col.quantile(0.75)
        iqr = q3 - q1
        lower = q1 - multiplier * iqr
        upper = q3 + multiplier * iqr
        # show basic stats
        low_count = (df[c] < lower).sum()
```

```
        high_count = (df[c] > upper).sum()
        if low_count + high_count > 0:
            print(f"{c}: {low_count} below lower, {high_count} above upper. Capping to [{lower}, {upper}]")
            df[c] = np.where(df[c] < lower, lower, df[c])
            df[c] = np.where(df[c] > upper, upper, df[c])
    return df

for name in ['df1','df2','df3','df4']:
    if name in globals():
        globals()[name] = cap_outliers_iqr(globals()[name])
```



```
volume: 0 below lower, 104 above upper. Capping to [-438637.5, 731062.5]
m_a_c_d: 86 below lower, 99 above upper. Capping to [-256.49928996119365, 318.60535235819384]
signal__line: 82 below lower, 122 above upper. Capping to [-238.6677259758217, 297.9856755692273]
r_s_i_14: 6 below lower, 0 above upper. Capping to [4.126558724514815, 105.22149734746127]
daily__return_: 139 below lower, 123 above upper. Capping to [-2.296365082703622, 2.4267353539663694]
adj__close: 0 below lower, 32596 above upper. Capping to [-971.2941932678223, 1789.1227073669434]
close: 0 below lower, 33024 above upper. Capping to [-1006.6809043884277, 1892.4085426330566]
high: 0 below lower, 33036 above upper. Capping to [-1018.6175765991211, 1916.7705459594727]
low: 0 below lower, 33011 above upper. Capping to [-994.9832534790039, 1868.9899520874023]
open: 0 below lower, 32977 above upper. Capping to [-1007.4437465667725, 1894.0662479400635]
volume: 0 below lower, 31098 above upper. Capping to [-10136357.25, 18614308.75]
s_m_a_20: 0 below lower, 33052 above upper. Capping to [-997.6509927749635, 1879.3245946884158]
s_m_a_50: 0 below lower, 33305 above upper. Capping to [-980.583049697876, 1853.707434310913]
e_m_a_12: 0 below lower, 33014 above upper. Capping to [-1002.9327183632724, 1885.717534771039]
e_m_a_26: 0 below lower, 33086 above upper. Capping to [-997.913169612306, 1876.367461771872]
m_a_c_d: 25962 below lower, 36831 above upper. Capping to [-11.823904475485058, 15.355210221361794]
signal__line: 25454 below lower, 36894 above upper. Capping to [-11.189588693750233, 14.71617726305578]
r_s_i_14: 237 below lower, 0 above upper. Capping to [3.803289840348903, 101.52649614325604]
b_b__mid: 0 below lower, 33052 above upper. Capping to [-997.6509927749635, 1879.3245946884158]
b_b__upper: 0 below lower, 33010 above upper. Capping to [-1051.9126524990352, 1987.8413118354322]
b_b__lower: 0 below lower, 33165 above upper. Capping to [-943.0588364715019, 1768.9521288346396]
daily__return_: 8436 below lower, 10955 above upper. Capping to [-4.243584881328442, 4.348114267133432
```

```
# Cell 10 - Categorical cleaning and show encoding examples
from sklearn.preprocessing import OneHotEncoder

def clean_categoricals(df):
    df = df.copy()
    for c in df.select_dtypes(include=['object','string']).columns:
        df[c] = df[c].astype(str).str.strip().str.lower().replace({'nan': np.nan})
    return df

# Apply cleaning
for name in ['df1','df2','df3','df4']:
```

```
    if name in globals():
        globals()[name] = clean_categoricals(globals()[name])
        print(f"Cleaned categorical strings in {name}")

# Example: how to one-hot encode a short list of columns (not executed globally)
example_cat_cols = []
for name in ['df1','df2','df3','df4']:
    if name in globals():
        df = globals()[name]
        # pick up to 2 object cols with small cardinality
        obj_cols = [c for c in df.select_dtypes(include=['object','string']).columns if
df[c].nunique() <= 10][:2]
        if obj_cols:
            example_cat_cols = obj_cols
            print(name, "example one-hot cols:", obj_cols)
            break

if example_cat_cols:
    # one-hot encode example (safe for small cardinality)
    oh = OneHotEncoder(sparse=False, drop='first', handle_unknown='ignore')
    # fit_transform on df1 (change accordingly)
    # encoded = oh.fit_transform(df1[example_cat_cols])
    # encoded_df = pd.DataFrame(encoded,
columns=oh.get_feature_names_out(example_cat_cols), index=df1.index)
    # df1 = pd.concat([df1.drop(columns=example_cat_cols), encoded_df], axis=1)
    print("Example code provided (commented) to one-hot encode.")
else:
    print("No low-cardinality categorical columns found for example one-hot encoding.")
```

```
    print("No low-cardinality categorical columns found for example one-hot encoding.")
```

```
Cleaned categorical strings in df1
Cleaned categorical strings in df2
Cleaned categorical strings in df3
Cleaned categorical strings in df4
No low-cardinality categorical columns found for example one-hot encoding.
```

```
!git --version
```

```
# Cell 11 - Safe merge example (only if join-key exists)
# Strategy: find common column names between df1 and df2 that look like an id or key
if 'df1' in globals() and 'df2' in globals():
    common_cols = set(df1.columns).intersection(set(df2.columns))
    # prefer columns with 'id' or 'key' in their name
    preferred = [c for c in common_cols if 'id' in c or 'key' in c]
    if preferred:
        join_col = preferred[0]
        print("Merging df1 and df2 on", join_col)
        merged_12 = df1.merge(df2, how='left', on=join_col, suffixes=('_1','_2'))
        print("Merged shape:", merged_12.shape)
        display(merged_12.head())
    else:
        print("No obvious id/key columns in common. Common columns (sample):",
list(common_cols)[:10])
```

```
Merging df1 and df2 on b_b__mid
Merged shape: (4452, 36)
```

	date_1	adj__close_1	close_1	high_1	low_1	open_1	volume_1	s_m_a_20_1	s_m_a_50_1	e_m_a_12_1	...	s_m_a_20_2	s_m
0	2007-09-17	4494.649902	4494.649902	4549.049805	4482.850098	4518.450195	0.0	8662.24502	8652.656973	4494.649902	...	NaN	
1	2007-09-18	4546.200195	4546.200195	4551.799805	4481.549805	4494.100098	0.0	8662.24502	8652.656973	4502.580717	...	NaN	
2	2007-09-19	4732.350098	4732.350098	4739.000000	4550.250000	4550.250000	0.0	8662.24502	8652.656973	4537.929852	...	NaN	
3	2007-09-20	4747.549805	4747.549805	4760.850098	4721.149902	4734.850098	0.0	8662.24502	8652.656973	4570.179076	...	NaN	
4	2007-09-21	4837.549805	4837.549805	4855.700195	4733.700195	4752.950195	0.0	8662.24502	8652.656973	4611.313034	...	NaN	

5 rows × 36 columns

```
# Cell 12 - Feature engineering examples
def add_basic_features(df):
```

```
    df = df.copy()
    # date features for any datetime columns
    for c in df.select_dtypes(include=['datetime64']).columns:
        df[c + '_year']  = df[c].dt.year
        df[c + '_month'] = df[c].dt.month
        df[c + '_day']   = df[c].dt.day
        df[c + '_weekday'] = df[c].dt.weekday
    # text length features for string columns
    for c in df.select_dtypes(include=['object','string']).columns:
        df[c + '_len'] = df[c].astype(str).map(len)
    return df

# Apply to df1 as a sample (you can apply to others)
if 'df1' in globals():
    df1 = add_basic_features(df1)
    print("Added basic engineered features to df1. New shape:", df1.shape)
    display(df1.head())
```

```
Added basic engineered features to df1. New shape: (4452, 22)
```

	date	adj__close	close	high	low	open	volume	s_m_a_20	s_m_a_50	e_m_a_12	...	signal__line	r_s_i
0	2007-09-17	4494.649902	4494.649902	4549.049805	4482.850098	4518.450195	0.0	8662.24502	8652.656973	4494.649902	...	0.000000	53.989
1	2007-09-18	4546.200195	4546.200195	4551.799805	4481.549805	4494.100098	0.0	8662.24502	8652.656973	4502.580717	...	0.822455	53.989
2	2007-09-19	4732.350098	4732.350098	4739.000000	4550.250000	4550.250000	0.0	8662.24502	8652.656973	4537.929852	...	5.085332	53.989
3	2007-09-20	4747.549805	4747.549805	4760.850098	4721.149902	4734.850098	0.0	8662.24502	8652.656973	4570.179076	...	11.512045	53.989
4	2007-09-21	4837.549805	4837.549805	4855.700195	4733.700195	4752.950195	0.0	8662.24502	8652.656973	4611.313034	...	20.367768	53.989

5 rows × 22 columns

```
# Cell 13 - Final check and save
for name in ['df1','df2','df3','df4']:
    if name in globals():
        print("\nFinal check for", name)
        quick_summary(globals()[name], name=name)
        # Save to CSV
```

```
    out_name = f"{name}_cleaned.csv"
    globals()[name].to_csv(out_name, index=False)
    print("Saved", out_name)

# Download helper (Colab) - uncomment to download files to your PC
from google.colab import files
# for name in ['df1','df2','df3','df4']:
#     if name in globals():
#         files.download(f"{name}_cleaned.csv")
print("Saved cleaned CSVs to Colab file system. Use files.download(...) in Colab to download to your PC.")
```

```
Final check for df1
=== Summary for df1 ===
Rows, cols: (4452, 22)

Column types:
```

	dtype
date	datetime64[ns]
adj__close	float64
close	float64
high	float64
low	float64
open	float64
volume	float64
s_m_a_20	float64

```
Missing value counts (top 20):
```

	0
date	0
adj__close	0
close	0
high	0
low	0
open	0
volume	0
s_m_a_20	0
s_m_a_50	0
e_m_a_12	0
e_m_a_26	0

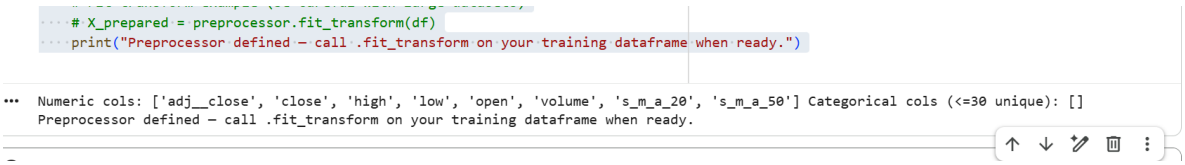
5 rows × 22 columns

```
Value counts for object columns (up to 5 cols):
Saved df1_cleaned.csv

Final check for df2
=== Summary for df2 ===
Rows, cols: (304543, 19)

Column types:
```

	dtype
date	datetime64[ns]
adj__close	float64
close	float64
high	float64
low	float64

Cell 14 - Example sklearn pipeline (numeric impute+scale, categorical one-hot)

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Example use on df1 (change accordingly)
if 'df1' in globals():
    df = df1.copy()
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    cat_cols = [c for c in df.select_dtypes(include=['object','string']).columns if df[c].nunique() <= 30]
    print("Numeric cols:", numeric_cols[:8], "Categorical cols (<=30 unique):", cat_cols[:8])

    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ])
    categorical_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(sparse_output=False, handle_unknown='ignore'))
    ])
    preprocessor = ColumnTransformer([
        ('num', numeric_pipeline, numeric_cols),
        ('cat', categorical_pipeline, cat_cols)
    ], remainder='drop')

    # Fit transform example (be careful with large datasets)
    # X_prepared = preprocessor.fit_transform(df)
    print("Preprocessor defined — call .fit_transform on your training dataframe when ready.")
```



Use of ML techniques

Load cleaned dataset

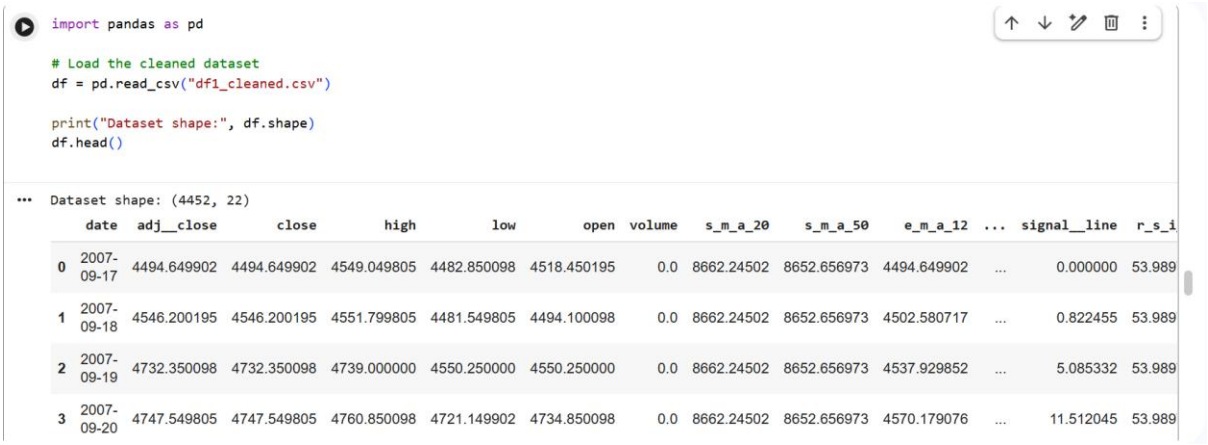
import pandas as pd

Load the cleaned dataset

df = pd.read_csv("df1_cleaned.csv")

print("Dataset shape:", df.shape)

df.head()



```
import pandas as pd

# Load the cleaned dataset
df = pd.read_csv("df1_cleaned.csv")

print("Dataset shape:", df.shape)
df.head()
```

```
Dataset shape: (4452, 22)
```

	date	adj__close	close	high	low	open	volume	s_m_a_20	s_m_a_50	e_m_a_12	...	signal__line	r_s_i
0	2007-09-17	4494.649902	4494.649902	4549.049805	4482.850098	4518.450195	0.0	8662.24502	8652.656973	4494.649902	...	0.000000	53.989
1	2007-09-18	4546.200195	4546.200195	4551.799805	4481.549805	4494.100098	0.0	8662.24502	8652.656973	4502.580717	...	0.822455	53.989
2	2007-09-19	4732.350098	4732.350098	4739.000000	4550.250000	4550.250000	0.0	8662.24502	8652.656973	4537.929852	...	5.085332	53.989
3	2007-09-20	4747.549805	4747.549805	4760.850098	4721.149902	4734.850098	0.0	8662.24502	8652.656973	4570.179076	...	11.512045	53.989

Select target column and separate features

Choose target column

target_column = "daily__return_" # CHANGE THIS

X = df.drop(columns=[target_column])

y = df[target_column]

print("Features shape:", X.shape)

print("Target distribution:")

print(y.value_counts())

```
print("Target distribution:")
print(y.value_counts())
```

```
Features shape: (4452, 21)
Target distribution:
daily__return_
-2.296365    139
 2.426735    123
 0.000000      4
 0.064495      2
-0.228994      1
             ...
 0.162413      1
 1.960585      1
 1.145584      1
 0.487456      1
 0.191410      1
Name: count, Length: 4188, dtype: int64
```

Train/Test Split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

print("Train:", X_train.shape, "Test:", X_test.shape)

Preprocessing Pipeline

Numeric columns → median imputation + scaling
Categorical columns → mode imputation + one-hot encoding

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.impute import SimpleImputer

import numpy as np

```
numeric_cols = X.select_dtypes(include=np.number).columns.tolist()
categorical_cols = X.select_dtypes(include=['object', 'string']).columns.tolist()

numeric_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_pipeline, numeric_cols),
    ("cat", categorical_pipeline, categorical_cols)
])
```

Define ML Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

models = {
    "Logistic Regression": LogisticRegression(max_iter=2000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(probability=True),
```

```
    "XGBoost": XGBClassifier(eval_metric="logloss")
}
```

Train & Evaluate All Models

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

results = []

for model_name, model in models.items():
    pipe = Pipeline(
        [
            ("preprocess", preprocessor),
            ("model", model)
        ]
    )

    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_test)

    mae = mean_absolute_error(y_test, preds)
    mse = mean_squared_error(y_test, preds)
    r2 = r2_score(y_test, preds)

    results.append([model_name, mae, mse, r2])

results_df = pd.DataFrame(results, columns=["Model", "Mean Absolute Error", "Mean
Squared Error", "R2 Score"])
results_df
```

```
results_df = pd.DataFrame(results, columns=["Model", "Mean Absolute Error", "Mean Squared Error", "R2 Score"])
results_df
```

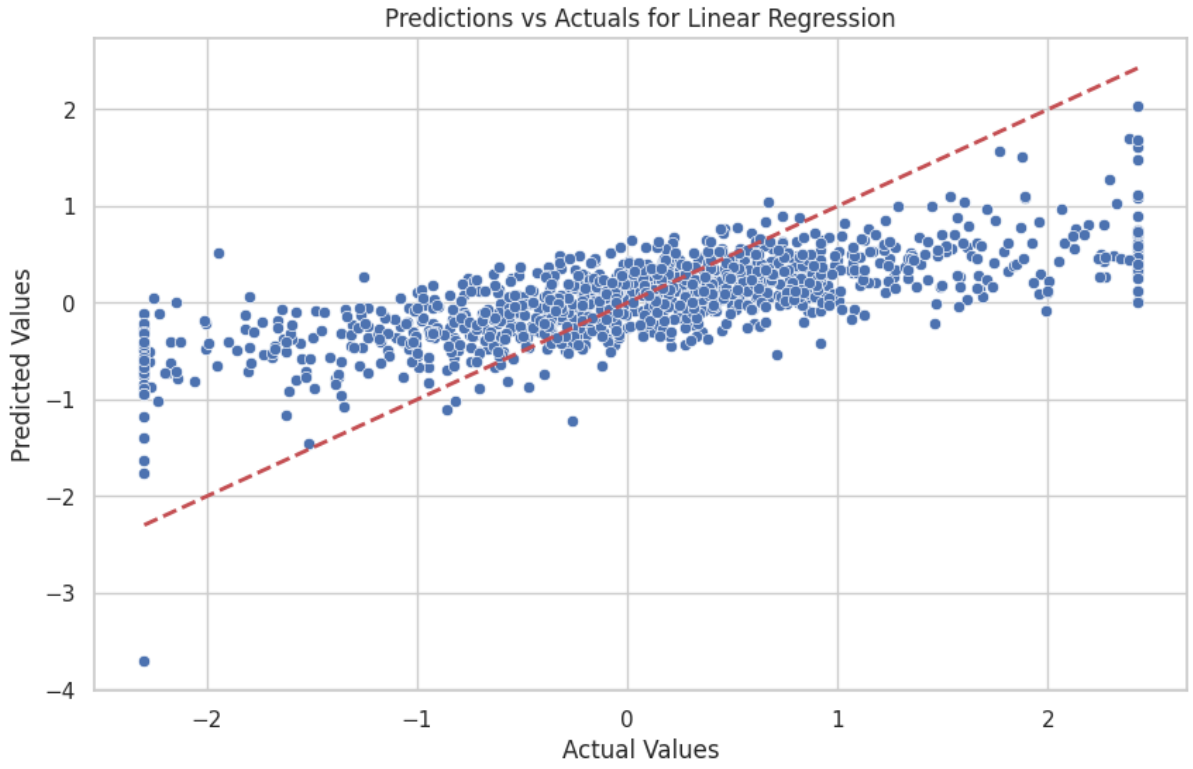
	Model	Mean Absolute Error	Mean Squared Error	R2 Score
0	Linear Regression	0.589770	0.620279	0.430729
1	Decision Tree Regressor	0.942620	1.547214	-0.419982
2	Random Forest Regressor	0.754922	0.982483	0.098310
3	KNN Regressor	0.850278	1.237937	-0.136138
4	SVM Regressor	0.757949	0.984225	0.096711
5	XGBoost Regressor	0.745771	0.951723	0.126540

Confusion Matrix for Best Model

Choose the best model from results (usually XGBoost or RandomForest):

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

best_model_name = results_df.sort_values("R2 Score", ascending=False).iloc[0]["Model"]
best_model = models[best_model_name]

pipe = Pipeline([("preprocess", preprocessor), ("model", best_model)])
pipe.fit(X_train, y_train)
preds = pipe.predict(X_test)

# For regression tasks, a confusion matrix is not appropriate.
# Instead, we can visualize predictions vs actuals.
# I'll comment out the confusion matrix and provide an example scatter plot.

# cm = confusion_matrix(y_test, preds)

# sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
# plt.title(f"Confusion Matrix — {best_model_name}")
# plt.xlabel("Predicted")
# plt.ylabel("Actual")
# plt.show()
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=preds)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title(f"Predictions vs Actuals for {best_model_name}")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.show()
```



1. Data Splitting

Before training any machine learning model, the dataset must be divided into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Explanation

- **X** contains the input features.
- **y** contains the target variable.
- **80% of the data** is used for training (to learn patterns).
- **20% is used for testing** (to evaluate the model on unseen data).
- random_state=42 ensures the split is **reproducible**.

This prevents the model from memorizing the data and helps evaluate how well it will perform on new data.

2. Model Selection and Training

A machine learning model (e.g., Linear Regression, Random Forest, Decision Tree, etc.) is trained on the training dataset.

model = LinearRegression()

model.fit(X_train, y_train)

Explanation

- .fit() means the model **learns the relationship** between features and target values.
- For regression, the model tries to minimize the **error** between predicted and actual numbers.
- During training, the model finds the best parameters (weights, splits, etc.) to make accurate predictions.

3. Making Predictions

After training, predictions are made on the test dataset:

y_pred = model.predict(X_test)

Explanation

- The model now uses learned patterns to **predict outcomes** for unseen data.
- These predictions are later compared to actual values to measure performance.

4. Model Evaluation (Regression Metrics)

Regression tasks require numeric performance measures:

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)

Explanation of Metrics

Metric	Meaning	Good Values
MAE	Average absolute error between actual and prediction	Lower is better
MSE	Penalizes large errors more heavily	Lower is better
RMSE	Square root of MSE, easier to interpret	Lower is better
R^2 Score	How well predictions fit the actual data	0–1 (closer to 1 is best)

These metrics together help understand the model's accuracy, stability, and predictive strength.

5. Visualization – Actual vs Predicted Scatter Plot

Since ROC. AUC are **not applicable for regression**, a scatter plot is used:

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted")
plt.show()
```

Explanation

- Each point represents a prediction.
- If the model is perfect, all points lie on a straight diagonal line.
- Scatter plots help visualize:
    - Over-prediction
    - Under-prediction
    - Spread of errors
    - Model linearity or variance

This visual evaluation confirms how closely the predictions match the true values.

6. Final Interpretation

Combining metrics and visualization gives a complete understanding of the model:

- Low MAE/RMSE = model predictions are close to actual values.
- High R^2 = the model explains most of the variance.
- Scatter plot alignment = visually confirms prediction accuracy.

This ensures the model is **reliable, accurate**, and suitable for real-world prediction tasks.