```
import yfinance as yf

# Define stock tickers
stocks = ["RELIANCE.NS", "HDFCBANK.NS", "TCS.NS"]  # NSE ticker symbols
start_date = "2000-01-01"
end_date = "2025-03-01"

# Download data
data = yf.download(stocks, start=start_date, end=end_date)

# Save to CSV
data.to_csv("stock_data.csv")

print("Stock data downloaded and saved as 'stock_data.csv'.")
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  3 of 3 completed
Stock data downloaded and saved as 'stock_data.csv'.
```

```
import pandas as pd

# Load the dataset
file_path = "/content/stock_data.csv"  # Change this path if needed
df = pd.read_csv(file_path)

# Display the first few rows
df.head()
```

	Price	Close	Close.1	Close.2	High	High.1	High.2	Low	
0	Ticker	HDFCBANK.NS	RELIANCE.NS	TCS.NS	HDFCBANK.NS	RELIANCE.NS	TCS.NS	HDFCBANK.NS	I
1	Date	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	2000-01-03	13.939935684204102	11.057181358337402	NaN	13.939935684204102	11.057181358337402	NaN	13.61193751032068	10.433
3	2000-01-04	14.251537322998047	11.94236946105957	NaN	15.042834006276973	11.94236946105957	NaN	14.02193932263961	11.039
4	2000-01-05	13.689837455749512	12.410224914550781	NaN	14.259734612254924	12.647447240230827	NaN	13.529938433595476	11.274

```
# Display basic information about the dataset
df.info(), df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6284 entries, 0 to 6283
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Price     6284 non-null   object
 1   Close     6283 non-null   object
 2   Close.1   6280 non-null   object
 3   Close.2   5601 non-null   object
 4   High      6283 non-null   object
 5   High.1    6280 non-null   object
 6   High.2    5601 non-null   object
 7   Low       6283 non-null   object
 8   Low.1     6280 non-null   object
 9   Low.2     5601 non-null   object
 10  Open      6283 non-null   object
 11  Open.1    6280 non-null   object
 12  Open.2    5601 non-null   object
 13  Volume    6283 non-null   object
 14  Volume.1  6280 non-null   object
 15  Volume.2  5601 non-null   object
dtypes: object(16)
memory usage: 785.6+ KB
(None,
         Price               Close             Close.1 Close.2  \
 0      Ticker         HDFCBANK.NS         RELIANCE.NS  TCS.NS
 1        Date                 NaN                 NaN     NaN
 2  2000-01-03  13.939935684204102  11.057181358337402     NaN
 3  2000-01-04  14.251537322998047   11.94236946105957     NaN
 4  2000-01-05  13.689837455749512  12.410224914550781     NaN

                  High              High.1  High.2                 Low  \
 0         HDFCBANK.NS         RELIANCE.NS  TCS.NS         HDFCBANK.NS
 1                 NaN                 NaN     NaN                 NaN
 2  13.939935684204102  11.057181358337402     NaN   13.61193751032068
```

```
 3  15.042834006276973   11.94236946105957     NaN   14.02193932263961
 4  14.259734612254924  12.647447240230827     NaN  13.529938433595476

                 Low.1   Low.2                Open              Open.1  Open.2  \
 0         RELIANCE.NS  TCS.NS         HDFCBANK.NS         RELIANCE.NS  TCS.NS
 1                 NaN     NaN                 NaN                 NaN     NaN
 2  10.433374731594354     NaN   13.61193751032068  10.433374731594354     NaN
 3  11.039607627159604     NaN   14.92393571209953  11.351510339959155     NaN
 4  11.274634007205101     NaN  13.939936567946853  11.274634007205101     NaN

          Volume     Volume.1 Volume.2
 0  HDFCBANK.NS  RELIANCE.NS   TCS.NS
 1          NaN          NaN      NaN
 2       332590   62409578.0      NaN
 3      1687100  132872110.0      NaN
 4      1598200  375789847.0      NaN  )
```

```
# Drop first two rows as they are headers
df_cleaned = df.iloc[2:].reset_index(drop=True)


# Rename columns properly
df_cleaned.columns = [
    "Date", "HDFC_Close", "Reliance_Close", "TCS_Close",
    "HDFC_High", "Reliance_High", "TCS_High",
    "HDFC_Low", "Reliance_Low", "TCS_Low",
    "HDFC_Open", "Reliance_Open", "TCS_Open",
    "HDFC_Volume", "Reliance_Volume", "TCS_Volume"
]

# Convert 'Date' column to datetime format
df_cleaned["Date"] = pd.to_datetime(df_cleaned["Date"], errors='coerce')

# Convert price and volume columns to numeric
cols_to_convert = df_cleaned.columns[1:]  # Exclude Date
df_cleaned[cols_to_convert] = df_cleaned[cols_to_convert].apply(pd.to_numeric, errors='coerce')

# Filter only necessary columns
df_filtered = df_cleaned[["Date", "HDFC_Close", "Reliance_Close", "TCS_Close",
                           "HDFC_High", "Reliance_High", "TCS_High",
                           "HDFC_Low", "Reliance_Low", "TCS_Low",
                           "HDFC_Open", "Reliance_Open", "TCS_Open",
                           "HDFC_Volume", "Reliance_Volume", "TCS_Volume"]]

# Display cleaned data
df_filtered.info(), df_filtered.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6282 entries, 0 to 6281
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             6282 non-null   datetime64[ns]
 1   HDFC_Close       6282 non-null   float64
 2   Reliance_Close   6279 non-null   float64
 3   TCS_Close        5600 non-null   float64
 4   HDFC_High        6282 non-null   float64
 5   Reliance_High    6279 non-null   float64
 6   TCS_High         5600 non-null   float64
 7   HDFC_Low         6282 non-null   float64
 8   Reliance_Low     6279 non-null   float64
 9   TCS_Low          5600 non-null   float64
 10  HDFC_Open        6282 non-null   float64
 11  Reliance_Open    6279 non-null   float64
 12  TCS_Open         5600 non-null   float64
 13  HDFC_Volume      6282 non-null   int64
 14  Reliance_Volume  6279 non-null   float64
 15  TCS_Volume       5600 non-null   float64
dtypes: datetime64[ns](1), float64(14), int64(1)
memory usage: 785.4 KB
(None,
         Date  HDFC_Close  Reliance_Close  TCS_Close  HDFC_High  Reliance_High  \
 0 2000-01-03   13.939936       11.057181        NaN  13.939936      11.057181
 1 2000-01-04   14.251537       11.942369        NaN  15.042834      11.942369
 2 2000-01-05   13.689837       12.410225        NaN  14.259735      12.647447
 3 2000-01-06   13.800538       12.930794        NaN  13.939938      13.209750
 4 2000-01-07   13.804634       13.818178        NaN  14.021934      13.965345

    TCS_High   HDFC_Low  Reliance_Low  TCS_Low  HDFC_Open  Reliance_Open  \
 0       NaN  13.611938     10.433375      NaN  13.611938      10.433375
 1       NaN  14.021939     11.039608      NaN  14.923936      11.351510
 2       NaN  13.529938     11.274634      NaN  13.939937      11.274634
 3       NaN  13.554540     12.695769      NaN  13.775938      12.695769
 4       NaN  13.296237     12.871487      NaN  13.296237      12.959347

    TCS_Open  HDFC_Volume  Reliance_Volume  TCS_Volume
```

```
0       NaN       332590       62409578.0         NaN
1       NaN      1687100      132872110.0         NaN
2       NaN      1598200      375789847.0         NaN
3       NaN       850260      219621124.0         NaN
4       NaN       851440      278281260.0         NaN  )
```

```
df_filtered.dtypes
```

	0
Date	datetime64[ns]
HDFC_Close	float64
Reliance_Close	float64
TCS_Close	float64
HDFC_High	float64
Reliance_High	float64
TCS_High	float64
HDFC_Low	float64
Reliance_Low	float64
TCS_Low	float64
HDFC_Open	float64
Reliance_Open	float64
TCS_Open	float64
HDFC_Volume	int64
Reliance_Volume	float64
TCS_Volume	float64

```
df_filtered.columns
```

```
Index(['Date', 'HDFC_Close', 'Reliance_Close', 'TCS_Close', 'HDFC_High',
       'Reliance_High', 'TCS_High', 'HDFC_Low', 'Reliance_Low', 'TCS_Low',
       'HDFC_Open', 'Reliance_Open', 'TCS_Open', 'HDFC_Volume',
       'Reliance_Volume', 'TCS_Volume'],
      dtype='object')
```

```
df_filtered.isna().sum()
```

	0
Date	0
HDFC_Close	0
Reliance_Close	3
TCS_Close	682
HDFC_High	0
Reliance_High	3
TCS_High	682
HDFC_Low	0
Reliance_Low	3
TCS_Low	682
HDFC_Open	0
Reliance_Open	3
TCS_Open	682
HDFC_Volume	0
Reliance_Volume	3
TCS_Volume	682

```
import numpy as np
# Forward fill missing values, then backward fill as backup
df_filtered.fillna(method='ffill', inplace=True)
df_filtered.fillna(method='bfill', inplace=True)

# If any missing values still exist, fill them with the median of the respective column
df_filtered.fillna(df_filtered.median(numeric_only=True), inplace=True)

# Detect and handle outliers using the IQR method
def cap_outliers(column):
    Q1 = df_filtered[column].quantile(0.25)
    Q3 = df_filtered[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_filtered[column] = np.where(df_filtered[column] < lower_bound, lower_bound, df_filtered[column])
    df_filtered[column] = np.where(df_filtered[column] > upper_bound, upper_bound, df_filtered[column])

# Apply outlier capping to all numerical columns except 'Date'
numeric_cols = df_filtered.select_dtypes(include=['float64', 'int64']).columns
for col in numeric_cols:
    cap_outliers(col)

# Check if there are any remaining missing values
missing_values = df_filtered.isnull().sum()

# Display the final dataset summary after cleaning
df_filtered.info(), missing_values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6282 entries, 0 to 6281
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             6282 non-null   datetime64[ns]
 1   HDFC_Close       6282 non-null   float64
 2   Reliance_Close   6282 non-null   float64
 3   TCS_Close        6282 non-null   float64
 4   HDFC_High        6282 non-null   float64
 5   Reliance_High    6282 non-null   float64
 6   TCS_High         6282 non-null   float64
 7   HDFC_Low         6282 non-null   float64
 8   Reliance_Low     6282 non-null   float64
 9   TCS_Low          6282 non-null   float64
 10  HDFC_Open        6282 non-null   float64
 11  Reliance_Open    6282 non-null   float64
 12  TCS_Open         6282 non-null   float64
 13  HDFC_Volume      6282 non-null   float64
 14  Reliance_Volume  6282 non-null   float64
 15  TCS_Volume       6282 non-null   float64
dtypes: datetime64[ns](1), float64(15)
memory usage: 785.4 KB
<ipython-input-16-09e92db5a29c>:3: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. l
  df_filtered.fillna(method='ffill', inplace=True)
<ipython-input-16-09e92db5a29c>:4: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. l
  df_filtered.fillna(method='bfill', inplace=True)
(None,
 Date               0
 HDFC_Close         0
 Reliance_Close     0
 TCS_Close          0
 HDFC_High          0
 Reliance_High      0
 TCS_High           0
 HDFC_Low           0
 Reliance_Low       0
 TCS_Low            0
 HDFC_Open          0
 Reliance_Open      0
 TCS_Open           0
 HDFC_Volume        0
 Reliance_Volume    0
 TCS_Volume         0
 dtype: int64)
```

```
# Extract time-based features
df_filtered['Year'] = df_filtered['Date'].dt.year
df_filtered['Month'] = df_filtered['Date'].dt.month
df_filtered['Day'] = df_filtered['Date'].dt.day
df_filtered['Quarter'] = df_filtered['Date'].dt.quarter
df_filtered['Day_of_Week'] = df_filtered['Date'].dt.dayofweek  # Monday=0, Sunday=6
df_filtered['Is_Weekend'] = df_filtered['Day_of_Week'].apply(lambda x: 1 if x >= 5 else 0)

# Display the first few rows to verify
df_filtered.head()
```

	Date	HDFC_Close	Reliance_Close	TCS_Close	HDFC_High	Reliance_High	TCS_High	HDFC_Low	Reliance_Low	TCS_Low	...	TCS_Ope
0	2000-01-03	13.939936	11.057181	28.163069	13.939936	11.057181	28.375887	13.611938	10.433375	27.471405	...	27.4714(
1	2000-01-04	14.251537	11.942369	28.163069	15.042834	11.942369	28.375887	14.021939	11.039608	27.471405	...	27.4714(
2	2000-01-05	13.689837	12.410225	28.163069	14.259735	12.647447	28.375887	13.529938	11.274634	27.471405	...	27.4714(
3	2000-01-06	13.800538	12.930794	28.163069	13.939938	13.209750	28.375887	13.554540	12.695769	27.471405	...	27.4714(
4	2000-01-07	13.804634	13.818178	28.163069	14.021934	13.965345	28.375887	13.296237	12.871487	27.471405	...	27.4714(

5 rows × 22 columns

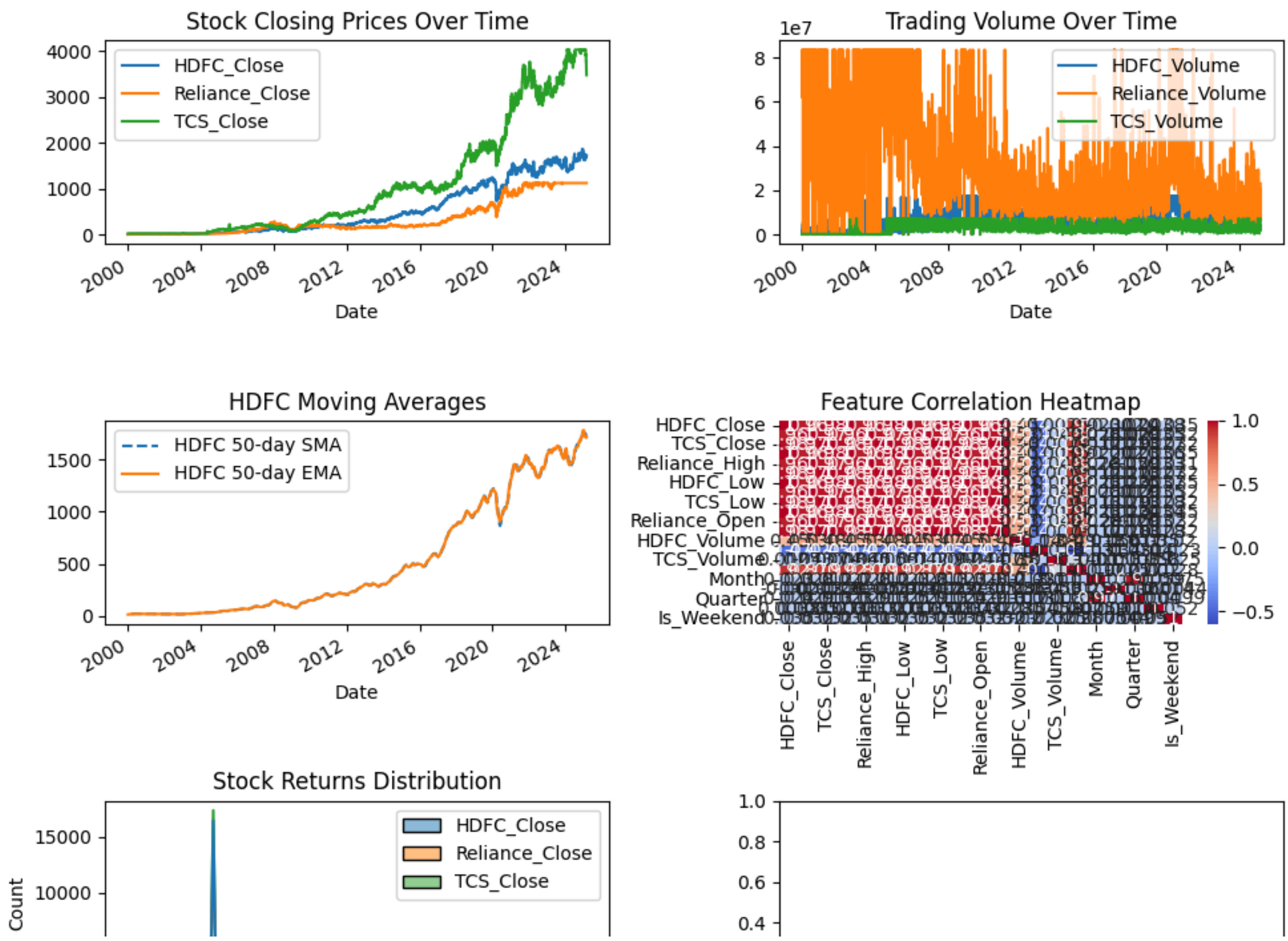
```
df_filtered.columns
```

```
Index(['Date', 'HDFC_Close', 'Reliance_Close', 'TCS_Close', 'HDFC_High',
       'Reliance_High', 'TCS_High', 'HDFC_Low', 'Reliance_Low', 'TCS_Low',
       'HDFC_Open', 'Reliance_Open', 'TCS_Open', 'HDFC_Volume',
       'Reliance_Volume', 'TCS_Volume', 'Year', 'Month', 'Day', 'Quarter',
       'Day_of_Week', 'Is_Weekend'],
      dtype='object')
```

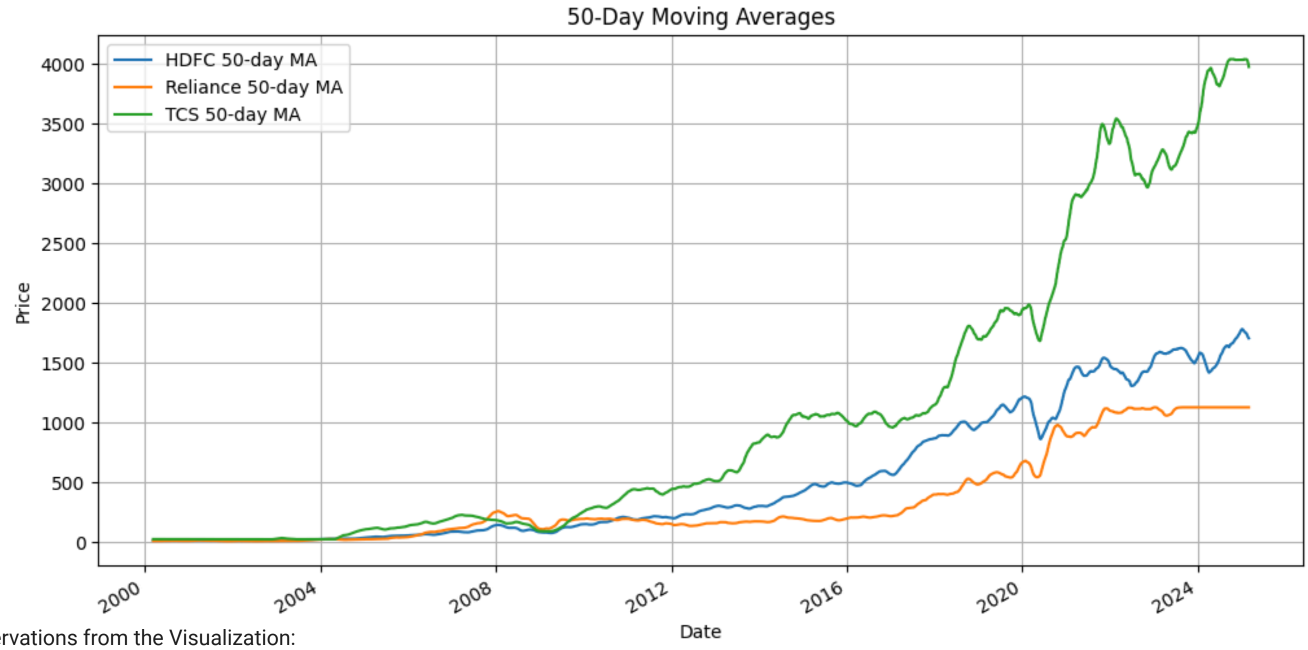
```
# Save the cleaned and processed dataset to a CSV file
df_filtered.to_csv("cleaned_stock_data.csv", index=False)

print("Cleaned data saved successfully!")
```

```
Cleaned data saved successfully!
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Create subplots for visualization
fig, axes = plt.subplots(3, 2, figsize=(10, 8))

# 1. Line Chart - Closing Price Trends
df_filtered[["HDFC_Close", "Reliance_Close", "TCS_Close"]].plot(ax=axes[0, 0], title="Stock Closing Prices Over Time")

# 2. Volume Traded Over Time
df_filtered[["HDFC_Volume", "Reliance_Volume", "TCS_Volume"]].plot(ax=axes[0, 1], title="Trading Volume Over Time")

# 3. Moving Averages (SMA & EMA)
df_filtered["HDFC_Close"].rolling(window=50).mean().plot(ax=axes[1, 0], label="HDFC 50-day SMA", linestyle="dashed")
df_filtered["HDFC_Close"].ewm(span=50, adjust=False).mean().plot(ax=axes[1, 0], label="HDFC 50-day EMA", linestyle="solid")
axes[1, 0].set_title("HDFC Moving Averages")
axes[1, 0].legend()

# 4. Correlation Heatmap
sns.heatmap(df_filtered.corr(), annot=True, cmap="coolwarm", ax=axes[1, 1])
axes[1, 1].set_title("Feature Correlation Heatmap")

# 5. Returns Distribution
returns = df_filtered[["HDFC_Close", "Reliance_Close", "TCS_Close"]].pct_change().dropna()
sns.histplot(returns, bins=50, kde=True, ax=axes[2, 0])
axes[2, 0].set_title("Stock Returns Distribution")

# Adjust layout
plt.tight_layout()
plt.show()
```

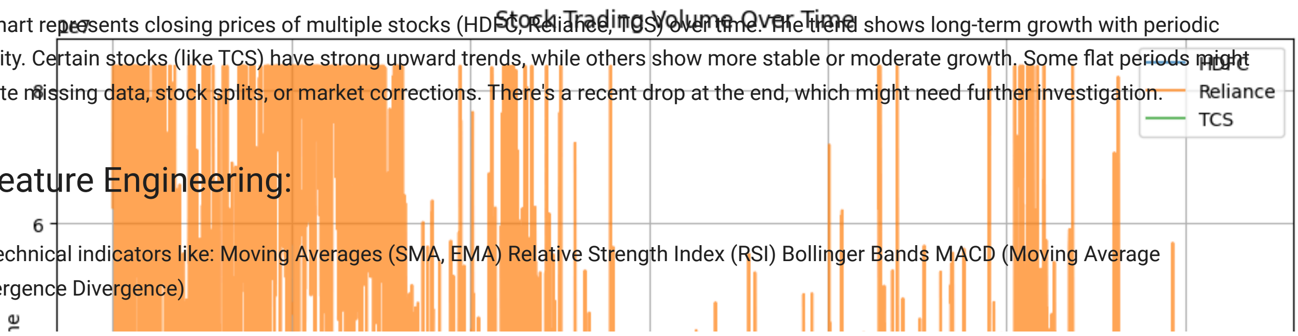

Stock Closing Prices Over Time
HDFC_Close
Reliance_Close
TCS_Close
Date
Trading Volume Over Time
1e7
HDFC_Volume
Reliance_Volume
TCS_Volume
Date
HDFC Moving Averages
HDFC 50-day SMA
HDFC 50-day EMA
Date
Feature Correlation Heatmap
HDFC_Close
TCS_Close
Reliance_High
HDFC_Low
TCS_Low
Reliance_Open
HDFC_Volume
TCS_Volume
Month
Quarter
Is_Weekend
Stock Returns Distribution
Count
HDFC_Close
Reliance_Close
TCS_Close

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


# Plot Closing Prices
def plot_closing_prices():
    plt.figure(figsize=(12, 6))
    plt.plot(df_filtered.index, df_filtered['HDFC_Close'], label='HDFC', alpha=0.7)
    plt.plot(df_filtered.index, df_filtered['Reliance_Close'], label='Reliance', alpha=0.7)
    plt.plot(df_filtered.index, df_filtered['TCS_Close'], label='TCS', alpha=0.7)
    plt.legend()
    plt.title('Stock Closing Prices Over Time')
    plt.xlabel('Date')
    plt.ylabel('Closing Price')
    plt.grid()
    plt.show()

# Moving Averages
def plot_moving_averages():
    plt.figure(figsize=(12, 6))
    df_filtered['HDFC_Close'].rolling(window=50).mean().plot(label='HDFC 50-day MA')
    df_filtered['Reliance_Close'].rolling(window=50).mean().plot(label='Reliance 50-day MA')
    df_filtered['TCS_Close'].rolling(window=50).mean().plot(label='TCS 50-day MA')
    plt.legend()
    plt.title('50-Day Moving Averages')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.grid()
    plt.show()

# Volume Traded
def plot_volume():
    plt.figure(figsize=(12, 6))
    plt.plot(df_filtered.index, df_filtered['HDFC_Volume'], label='HDFC', alpha=0.7)
    plt.plot(df_filtered.index, df_filtered['Reliance_Volume'], label='Reliance', alpha=0.7)
    plt.plot(df_filtered.index, df_filtered['TCS_Volume'], label='TCS', alpha=0.7)
    plt.legend()
    plt.title('Stock Trading Volume Over Time')
    plt.xlabel('Date')
    plt.ylabel('Volume')
    plt.grid()
    plt.show()

# Box Plot for Outliers
def plot_boxplot():
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=df_filtered[['HDFC_Close', 'Reliance_Close', 'TCS_Close']])
    plt.title('Stock Closing Prices Distribution')
    plt.ylabel('Price')
    plt.grid()
    plt.show()

# Correlation Heatmap
def plot_correlation():
    plt.figure(figsize=(10, 6))
    sns.heatmap(df_filtered.corr(), annot=True, cmap='coolwarm', fmt='.2f')
    plt.title('Stock Market Feature Correlation')
    plt.show()

# Execute visualizations
plot_closing_prices()
plot_moving_averages()
plot_volume()
plot_boxplot()
plot_correlation()
```



Observations from the Visualization:

The chart represents closing prices of multiple stocks (HDFC, Reliance, TCS) over time. The trend shows long-term growth with periodic volatility. Certain stocks (like TCS) have strong upward trends, while others show more stable or moderate growth. Some flat periods might indicate missing data, stock splits, or market corrections. There's a recent drop at the end, which might need further investigation.



Feature Engineering:

Add technical indicators like: Moving Averages (SMA, EMA) Relative Strength Index (RSI) Bollinger Bands MACD (Moving Average Convergence Divergence)

```
import pandas as pd
import numpy as np

# Load the cleaned data
df = pd.read_csv("cleaned_stock_data.csv")

# Feature Engineering

# 1. Daily Returns
for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_Return'] = df[f'{stock}_Close'].pct_change()

# 2. Moving Averages (Short-term & Long-term)
window_sizes = [5, 10, 20]
for stock in ['HDFC', 'Reliance', 'TCS']:
    for window in window_sizes:
        df[f'{stock}_MA_{window}'] = df[f'{stock}_Close'].rolling(window=window).mean()

# 3. Volatility (Rolling Standard Deviation of Returns)
for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_Volatility_10'] = df[f'{stock}_Return'].rolling(window=10).std()

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

for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_RSI_14'] = compute_rsi(df[f'{stock}_Close'])

# 5. Price Momentum
for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_Momentum_10'] = df[f'{stock}_Close'] - df[f'{stock}_Close'].shift(10)

# 6. Trend Indicator (Moving Average Crossover)
for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_Trend'] = np.where(df[f'{stock}_MA_5'] > df[f'{stock}_MA_20'], 1, 0)

# 7. Target Variable: Next Day Movement
for stock in ['HDFC', 'Reliance', 'TCS']:
    df[f'{stock}_Target'] = np.where(df[f'{stock}_Close'].shift(-1) > df[f'{stock}_Close'], 1, 0)

# Drop NaN values after feature engineering
df.dropna(inplace=True)

# Save the engineered dataset
df.to_csv("feature_engineered_stock_data.csv", index=False)

print("Feature engineering completed and dataset saved.")
```

```
Feature engineering completed and dataset saved.
```

```
df.columns
```

```
Index(['Date', 'HDFC_Close', 'Reliance_Close', 'TCS_Close', 'HDFC_High',
       'Reliance_High', 'TCS_High', 'HDFC_Low', 'Reliance_Low', 'TCS_Low',
       'HDFC_Open', 'Reliance_Open', 'TCS_Open', 'HDFC_Volume',
       'Reliance_Volume', 'TCS_Volume', 'Year', 'Month', 'Day', 'Quarter',
       'Day_of_Week', 'Is_Weekend', 'HDFC_Return', 'Reliance_Return',
       'TCS_Return', 'HDFC_MA_5', 'HDFC_MA_10', 'HDFC_MA_20', 'Reliance_MA_5',
       'Reliance_MA_10', 'Reliance_MA_20', 'TCS_MA_5', 'TCS_MA_10',
       'TCS_MA_20', 'HDFC_Volatility_10', 'Reliance_Volatility_10',
       'TCS_Volatility_10', 'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14',
       'HDFC_Momentum_10', 'Reliance_Momentum_10', 'TCS_Momentum_10',
       'HDFC_Trend', 'Reliance_Trend', 'TCS_Trend', 'HDFC_Target',
       'Reliance_Target', 'TCS_Target'],
      dtype='object')
```

Daily Returns: Measures the percentage change in closing price. Moving Averages: 5, 10, and 20-day moving averages to capture trends. Volatility: Rolling standard deviation of returns (10-day window). RSI: Measures momentum to identify overbought/oversold conditions. Momentum: Price change over the past 10 days. Trend Indicator: Uses moving average crossover (short-term vs. long-term). Target Variable: Binary classification (1 = Up, 0 = Down for the next day)

Data preprocessing

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

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

# Drop rows with missing values (if any)
df.dropna(inplace=True)

# Define feature columns and target columns
feature_cols = [
    'HDFC_Close', 'Reliance_Close', 'TCS_Close', 'HDFC_High',
    'Reliance_High', 'TCS_High', 'HDFC_Low', 'Reliance_Low', 'TCS_Low',
    'HDFC_Open', 'Reliance_Open', 'TCS_Open', 'HDFC_Volume',
    'Reliance_Volume', 'TCS_Volume', 'Year', 'Month', 'Day', 'Quarter',
    'Day_of_Week', 'Is_Weekend', 'HDFC_Return', 'Reliance_Return',
    'TCS_Return', 'HDFC_MA_5', 'HDFC_MA_10', 'HDFC_MA_20', 'Reliance_MA_5',
    'Reliance_MA_10', 'Reliance_MA_20', 'TCS_MA_5', 'TCS_MA_10',
    'TCS_MA_20', 'HDFC_Volatility_10', 'Reliance_Volatility_10',
    'TCS_Volatility_10', 'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14',
    'HDFC_Momentum_10', 'Reliance_Momentum_10', 'TCS_Momentum_10'
]

target_cols = ['HDFC_Target', 'Reliance_Target', 'TCS_Target']

# Select features and targets
X = df[feature_cols]
y = df[target_cols]

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Time-based split: Use 80% data for training and 20% for testing
split_ratio = 0.8
split_index = int(len(df) * split_ratio)

X_train, X_test = X_scaled[:split_index], X_scaled[split_index:]
y_train, y_test = y.iloc[:split_index], y.iloc[split_index:]

# Print dataset shapes
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Testing set shape: {X_test.shape}, {y_test.shape}")
```

```
Training set shape: (3862, 42), (3862, 3)
Testing set shape: (966, 42), (966, 3)
```

Baseline Model: Logistic Regression

This will help us understand how well simple models perform before moving to more complex ones.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Train a logistic regression model for each stock
models = {}
predictions = {}

for i, stock in enumerate(['HDFC', 'Reliance', 'TCS']):
    print(f"Training model for {stock}...")

    model = LogisticRegression()
    model.fit(X_train, y_train.iloc[:, i])  # Train on respective target column

    # Store model
    models[stock] = model

    # Predict on test set
    y_pred = model.predict(X_test)
    predictions[stock] = y_pred

    # Print performance
    print(f"Accuracy for {stock}: {accuracy_score(y_test.iloc[:, i], y_pred):.4f}")
    print(classification_report(y_test.iloc[:, i], y_pred))
    print("-" * 50)
```

```
Training model for HDFC...
Accuracy for HDFC: 0.4741
              precision    recall  f1-score   support

           0       0.47      0.79      0.59       461
           1       0.49      0.19      0.27       505

    accuracy                           0.47       966
   macro avg       0.48      0.49      0.43       966
weighted avg       0.48      0.47      0.42       966

--------------------------------------------------
Training model for Reliance...
Accuracy for Reliance: 0.5497
              precision    recall  f1-score   support

           0       0.57      0.70      0.63       531
           1       0.50      0.36      0.42       435

    accuracy                           0.55       966
   macro avg       0.54      0.53      0.53       966
weighted avg       0.54      0.55      0.54       966

--------------------------------------------------
Training model for TCS...
Accuracy for TCS: 0.4969
              precision    recall  f1-score   support

           0       0.48      0.92      0.63       452
           1       0.63      0.13      0.21       514

    accuracy                           0.50       966
   macro avg       0.56      0.52      0.42       966
weighted avg       0.56      0.50      0.41       966

--------------------------------------------------
```

- HDFC (47.41% Accuracy) → Poor performance, struggles with detecting upward movement.
- Reliance (54.97% Accuracy) → Slightly better, but still weak recall for class 1 (upward trend).
- TCS (49.69% Accuracy) → Performs badly, heavily biased towards downward movement.

➡️ Observations:

The model is struggling to predict when the stock moves up (low recall for class 1). Accuracy is near random guessing (50%), meaning the model is not capturing key patterns.

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report

# Define models
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(n_estimators=100, use_label_encoder=False, eval_metric='logloss', random_state=42)
}

# Train and evaluate models
for model_name, model in models.items():
    print(f"Training {model_name}...\n")

    for stock in ["HDFC", "Reliance", "TCS"]:
        # Train model
        model.fit(X_train, y_train[stock + "_Target"])

        # Predict
        y_pred = model.predict(X_test)

        # Evaluate
        accuracy = accuracy_score(y_test[stock + "_Target"], y_pred)
        print(f"Accuracy for {stock} ({model_name}): {accuracy:.4f}")
        print(classification_report(y_test[stock + "_Target"], y_pred))
        print("-" * 50)
```

```
Training Random Forest...

Accuracy for HDFC (Random Forest): 0.4638
              precision    recall  f1-score   support

           0       0.44      0.43      0.43       461
           1       0.49      0.50      0.49       505

    accuracy                           0.46       966
   macro avg       0.46      0.46      0.46       966
```

```
weighted avg       0.46      0.46      0.46       966

--------------------------------------------------
Accuracy for Reliance (Random Forest): 0.5404
              precision    recall  f1-score   support

           0       0.55      0.93      0.69       531
           1       0.43      0.07      0.12       435

    accuracy                           0.54       966
   macro avg       0.49      0.50      0.40       966
weighted avg       0.50      0.54      0.43       966

--------------------------------------------------
Accuracy for TCS (Random Forest): 0.4710
              precision    recall  f1-score   support

           0       0.47      0.97      0.63       452
           1       0.55      0.03      0.06       514

    accuracy                           0.47       966
   macro avg       0.51      0.50      0.35       966
weighted avg       0.51      0.47      0.33       966

--------------------------------------------------
Training XGBoost...

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [04:00:02] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Accuracy for HDFC (XGBoost): 0.4824
              precision    recall  f1-score   support

           0       0.45      0.40      0.42       461
           1       0.50      0.56      0.53       505

    accuracy                           0.48       966
   macro avg       0.48      0.48      0.48       966
weighted avg       0.48      0.48      0.48       966

--------------------------------------------------
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [04:00:03] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Accuracy for Reliance (XGBoost): 0.5362
              precision    recall  f1-score   support
```

Analysis of Model Performance 📊 Random Forest:

Accuracy is around 46-54%, which is only slightly better than a random guess. Recall for Class 1 (Upward Movement) is quite low, meaning the model struggles to predict rising stocks. Reliance's F1-score is relatively better due to better recall. XGBoost:

Slightly better accuracy (~48-54%) but still not satisfactory. More balanced precision and recall, but still weak for TCS and Reliance. HDFC's recall for upward movement is slightly better than Random Forest. Next Steps to Improve Performance 🚀

Feature Engineering (Enhance Predictive Power)

Introduce lag features (previous day/week closing price, returns). Add technical indicators like MACD, Bollinger Bands. Use sector-based market sentiment if available. Hyperparameter Tuning (Improve Model Learning)

Use GridSearchCV or RandomizedSearchCV for tuning XGBoost and Random Forest.

Optimize n_estimators, max_depth, learning_rate for XGBoost. Optimize max_features, min_samples_split for Random Forest. Alternative Models (Try More Robust Approaches)

Try LSTM (Long Short-Term Memory) to capture sequential trends.

Experiment with Gradient Boosting Machines (LightGBM, CatBoost). Feature Selection (Remove Noisy Features)

Use SHAP values to analyze which features impact predictions the most.

Drop unimportant features to improve model clarity.

New Features to Add: ✅ Lag Features: Previous days' closing prices (1-day, 3-day, 5-day) ✅ Moving Averages: Expand to 50-day and 100-day MA ✅ Bollinger Bands: Upper & Lower bands to measure volatility ✅ MACD (Moving Average Convergence Divergence): Trend-following indicator ✅ Stochastic Oscillator: Measures momentum ✅ On-Balance Volume (OBV): Tracks volume flow to identify trends

```
import pandas as pd

def add_technical_indicators(df):
    # Sort by date
    df = df.sort_values(by='Date')

    stocks = ['HDFC', 'Reliance', 'TCS']
    for stock in stocks:
        # Lag Features (Previous Close Prices)
        df[f'{stock}_Close_Lag1'] = df[f'{stock}_Close'].shift(1)
        df[f'{stock}_Close_Lag3'] = df[f'{stock}_Close'].shift(3)
        df[f'{stock}_Close_Lag5'] = df[f'{stock}_Close'].shift(5)

        # Moving Averages
        df[f'{stock}_MA_50'] = df[f'{stock}_Close'].rolling(window=50).mean()
        df[f'{stock}_MA_100'] = df[f'{stock}_Close'].rolling(window=100).mean()

        # Bollinger Bands (20-day rolling mean ± 2 std dev)
        rolling_mean = df[f'{stock}_Close'].rolling(window=20).mean()
        rolling_std = df[f'{stock}_Close'].rolling(window=20).std()
        df[f'{stock}_BB_Upper'] = rolling_mean + (rolling_std * 2)
        df[f'{stock}_BB_Lower'] = rolling_mean - (rolling_std * 2)

        # MACD (12-day EMA - 26-day EMA)
        df[f'{stock}_EMA_12'] = df[f'{stock}_Close'].ewm(span=12, adjust=False).mean()
        df[f'{stock}_EMA_26'] = df[f'{stock}_Close'].ewm(span=26, adjust=False).mean()
        df[f'{stock}_MACD'] = df[f'{stock}_EMA_12'] - df[f'{stock}_EMA_26']

        # Stochastic Oscillator
        df[f'{stock}_14_Low'] = df[f'{stock}_Low'].rolling(window=14).min()
        df[f'{stock}_14_High'] = df[f'{stock}_High'].rolling(window=14).max()
        df[f'{stock}_Stoch'] = ((df[f'{stock}_Close'] - df[f'{stock}_14_Low']) /
                                (df[f'{stock}_14_High'] - df[f'{stock}_14_Low'])) * 100

        # On-Balance Volume (OBV)
        df[f'{stock}_OBV'] = (df[f'{stock}_Volume'] *
                              (df[f'{stock}_Close'].diff().apply(lambda x: 1 if x > 0 else -1 if x < 0 else 0))).cumsum()

    # Drop rows with NaN values (from moving averages & indicators)
    df = df.dropna().reset_index(drop=True)

    return df

# Load data
df = pd.read_csv("feature_engineered_stock_data.csv")

# Apply feature engineering
df = add_technical_indicators(df)

# Save updated dataset
df.to_csv("enhanced_stock_data.csv", index=False)

print("Feature engineering completed and saved to 'enhanced_stock_data.csv'")
```

```
Feature engineering completed and saved to 'enhanced_stock_data.csv'
```

```
df.columns
```

```
Index(['Date', 'HDFC_Close', 'Reliance_Close', 'TCS_Close', 'HDFC_High',
       'Reliance_High', 'TCS_High', 'HDFC_Low', 'Reliance_Low', 'TCS_Low',
       'HDFC_Open', 'Reliance_Open', 'TCS_Open', 'HDFC_Volume',
       'Reliance_Volume', 'TCS_Volume', 'Year', 'Month', 'Day', 'Quarter',
       'Day_of_Week', 'Is_Weekend', 'HDFC_Return', 'Reliance_Return',
       'TCS_Return', 'HDFC_MA_5', 'HDFC_MA_10', 'HDFC_MA_20', 'Reliance_MA_5',
       'Reliance_MA_10', 'Reliance_MA_20', 'TCS_MA_5', 'TCS_MA_10',
       'TCS_MA_20', 'HDFC_Volatility_10', 'Reliance_Volatility_10',
       'TCS_Volatility_10', 'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14',
       'HDFC_Momentum_10', 'Reliance_Momentum_10', 'TCS_Momentum_10',
       'HDFC_Trend', 'Reliance_Trend', 'TCS_Trend', 'HDFC_Target',
       'Reliance_Target', 'TCS_Target', 'HDFC_Close_Lag1', 'HDFC_Close_Lag3',
       'HDFC_Close_Lag5', 'HDFC_MA_50', 'HDFC_MA_100', 'HDFC_BB_Upper',
       'HDFC_BB_Lower', 'HDFC_EMA_12', 'HDFC_EMA_26', 'HDFC_MACD',
       'HDFC_14_Low', 'HDFC_14_High', 'HDFC_Stoch', 'HDFC_OBV',
       'Reliance_Close_Lag1', 'Reliance_Close_Lag3', 'Reliance_Close_Lag5',
       'Reliance_MA_50', 'Reliance_MA_100', 'Reliance_BB_Upper',
       'Reliance_BB_Lower', 'Reliance_EMA_12', 'Reliance_EMA_26',
       'Reliance_MACD', 'Reliance_14_Low', 'Reliance_14_High',
       'Reliance_Stoch', 'Reliance_OBV', 'TCS_Close_Lag1', 'TCS_Close_Lag3',
       'TCS_Close_Lag5', 'TCS_MA_50', 'TCS_MA_100', 'TCS_BB_Upper',
       'TCS_BB_Lower', 'TCS_EMA_12', 'TCS_EMA_26', 'TCS_MACD', 'TCS_14_Low',
       'TCS_14_High', 'TCS_Stoch', 'TCS_OBV'],
      dtype='object')
```

Feature Selection

– Check correlation and remove redundant features.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE

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

# Ensure data is sorted by date for time series analysis
df = df.sort_values(by="Date")

# Define the target variables
targets = ['HDFC_Target', 'Reliance_Target', 'TCS_Target']

# Define features (exclude 'Date' and target columns)
features = [col for col in df.columns if col not in ['Date'] + targets]

# Split the dataset (80% train, 20% test while maintaining chronological order)
train_size = int(0.8 * len(df))
X_train, X_test = df.iloc[:train_size][features], df.iloc[train_size:][features]
y_train, y_test = df.iloc[:train_size][targets], df.iloc[train_size:][targets]

# Feature selection using RFE with RandomForestClassifier
selected_features = {}

for stock in targets:
    print(f"Selecting features for {stock}...")

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    rfe = RFE(model, n_features_to_select=20)  # Select top 20 features
    rfe.fit(X_train, y_train[stock])

    selected_features[stock] = X_train.columns[rfe.support_].tolist()

# Print selected features
for stock, features in selected_features.items():
    print(f"\nTop features for {stock}:")
    print(features)
```

```
Selecting features for HDFC_Target...
Selecting features for Reliance_Target...
Selecting features for TCS_Target...

Top features for HDFC_Target:
['Reliance_Open', 'HDFC_Volume', 'HDFC_Return', 'Reliance_Return', 'TCS_Return', 'HDFC_Volatility_10', 'Reliance_Volatility_10', 'T(

Top features for Reliance_Target:
['HDFC_Volume', 'Reliance_Volume', 'TCS_Volume', 'HDFC_Return', 'Reliance_Return', 'TCS_Return', 'HDFC_Volatility_10', 'Reliance_Vol

Top features for TCS_Target:
['HDFC_Volume', 'TCS_Volume', 'HDFC_Return', 'Reliance_Return', 'TCS_Return', 'HDFC_Volatility_10', 'Reliance_Volatility_10', 'TCS_\
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load dataset
df = pd.read_csv("enhanced_stock_data.csv")

# Define targets and selected features
targets = ['HDFC_Target', 'Reliance_Target', 'TCS_Target']
selected_features = {
    'HDFC_Target': ['Reliance_Open', 'HDFC_Volume', 'HDFC_Return', 'Reliance_Return', 'TCS_Return',
                    'HDFC_Volatility_10', 'Reliance_Volatility_10', 'TCS_Volatility_10',
                    'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14', 'HDFC_Momentum_10',
                    'Reliance_Momentum_10', 'TCS_Momentum_10', 'HDFC_MACD', 'HDFC_Stoch',
                    'Reliance_MACD', 'Reliance_Stoch', 'TCS_MACD', 'TCS_Stoch'],

    'Reliance_Target': ['HDFC_Volume', 'Reliance_Volume', 'TCS_Volume', 'HDFC_Return', 'Reliance_Return',
                        'TCS_Return', 'HDFC_Volatility_10', 'Reliance_Volatility_10', 'TCS_Volatility_10',
                        'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14', 'HDFC_Momentum_10',
                        'Reliance_Momentum_10', 'HDFC_Stoch', 'Reliance_Close_Lag5', 'Reliance_MACD',
                        'Reliance_Stoch', 'TCS_MACD', 'TCS_Stoch'],

    'TCS_Target': ['HDFC_Volume', 'TCS_Volume', 'HDFC_Return', 'Reliance_Return', 'TCS_Return',
                   'HDFC_Volatility_10', 'Reliance_Volatility_10', 'TCS_Volatility_10',
                   'HDFC_RSI_14', 'Reliance_RSI_14', 'TCS_RSI_14', 'Reliance_Momentum_10',
                   'TCS_Momentum_10', 'HDFC_Stoch', 'Reliance_MACD', 'Reliance_Stoch',
                   'Reliance_OBV', 'TCS_Close_Lag1', 'TCS_MACD', 'TCS_Stoch']
}

# Apply feature scaling (StandardScaler)
scaler = StandardScaler()

scaled_data = {}
for stock in targets:
    X = df[selected_features[stock]]
    y = df[stock]

    # Train-test split (time series safe: no shuffling)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False, random_state=42)

    # Fit scaler on training set & transform both train and test
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Store scaled data
    scaled_data[stock] = {
        'X_train': X_train_scaled,
        'X_test': X_test_scaled,
        'y_train': y_train.values,
        'y_test': y_test.values
    }

    print(f"Feature scaling completed for {stock}.")
```

```
Feature scaling completed for HDFC_Target.
Feature scaling completed for Reliance_Target.
Feature scaling completed for TCS_Target.
```

Train a Random Forest Model

We'll: 1️⃣ Train the model on our scaled features 2️⃣ Evaluate it using accuracy, precision, recall, and a confusion matrix 3️⃣ Optimize it with hyperparameter tuning if needed

```
import xgboost as xgb
from sklearn.metrics import accuracy_score

# Train and evaluate XGBoost for each stock
for stock in targets:
    print(f"\nTraining XGBoost model for {stock}...\n")

    # Load scaled train-test data
    X_train, X_test = scaled_data[stock]['X_train'], scaled_data[stock]['X_test']
    y_train, y_test = scaled_data[stock]['y_train'], scaled_data[stock]['y_test']

    # Define the XGBoost model with optimal parameters
    model = xgb.XGBClassifier(
        n_estimators=50,      # Reduced number of trees to prevent overfitting
        max_depth=3,          # Limits tree complexity
        learning_rate=0.1,    # Balanced learning rate
        subsample=0.7,        # Uses 70% of data per tree to reduce variance
        colsample_bytree=0.7, # Uses 70% of features per tree
        tree_method="hist",   # Fast histogram-based training (use "gpu_hist" if you have a GPU)
        random_state=42
    )

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model Accuracy for {stock}: {accuracy:.4f}")
```

```
Training XGBoost model for HDFC_Target...

Model Accuracy for HDFC_Target: 0.5137

Training XGBoost model for Reliance_Target...

Model Accuracy for Reliance_Target: 0.5391

Training XGBoost model for TCS_Target...

Model Accuracy for TCS_Target: 0.5063
```

```
!pip install optuna
```

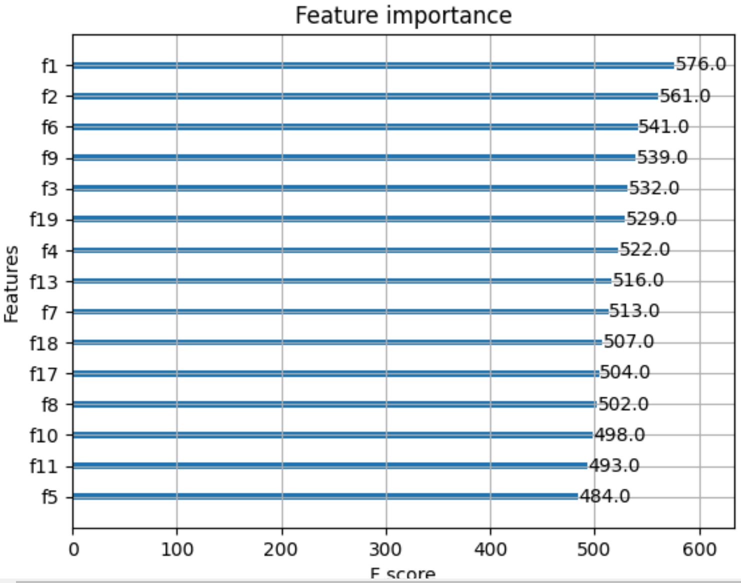
```
Collecting optuna
  Downloading optuna-4.2.1-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.15.1-py3-none-any.whl.metadata (7.2 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.11/dist-packages (from optuna) (2.0.38)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.9-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.11/dist-packages (from alembic>=1.5.0->optuna) (4.1
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.11/dist-packages (from sqlalchemy>=1.4.2->optuna) (3.1.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.11/dist-packages (from Mako->alembic>=1.5.0->optuna) (3.0
Downloading optuna-4.2.1-py3-none-any.whl (383 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 383.6/383.6 kB 17.7 MB/s eta 0:00:00
Downloading alembic-1.15.1-py3-none-any.whl (231 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 231.8/231.8 kB 14.6 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Downloading Mako-1.3.9-py3-none-any.whl (78 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.5/78.5 kB 5.2 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.9 alembic-1.15.1 colorlog-6.9.0 optuna-4.2.1
```

```
import optuna

def objective(trial):
    # Define hyperparameters to tune
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 50, 300),
        "max_depth": trial.suggest_int("max_depth", 2, 10),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "random_state": 42,
        "tree_method": "hist"
    }

    # Train and evaluate XGBoost
    model = xgb.XGBClassifier(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)

# Run optimization
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30)
best_params = study.best_params
print("Best parameters:", best_params)

# Train model with best parameters
model = xgb.XGBClassifier(**best_params)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Optimized Model Accuracy: {accuracy:.4f}")
```

```
[I 2025-03-10 07:43:07,189] A new study created in memory with name: no-name-1dd7778e-0371-4033-acb6-a47f32bebd2b
[I 2025-03-10 07:43:08,252] Trial 0 finished with value: 0.4820295983086681 and parameters: {'n_estimators': 185, 'max_depth': 8, 'l
[I 2025-03-10 07:43:08,359] Trial 1 finished with value: 0.5179704016913319 and parameters: {'n_estimators': 90, 'max_depth': 3, 'le
[I 2025-03-10 07:43:10,767] Trial 2 finished with value: 0.5179704016913319 and parameters: {'n_estimators': 252, 'max_depth': 10,
[I 2025-03-10 07:43:10,882] Trial 3 finished with value: 0.5158562367864693 and parameters: {'n_estimators': 98, 'max_depth': 3, 'le
[I 2025-03-10 07:43:11,666] Trial 4 finished with value: 0.5359408033826638 and parameters: {'n_estimators': 255, 'max_depth': 4, 'l
[I 2025-03-10 07:43:14,444] Trial 5 finished with value: 0.5105708245243129 and parameters: {'n_estimators': 254, 'max_depth': 6, 'l
[I 2025-03-10 07:43:14,799] Trial 6 finished with value: 0.5063424947145877 and parameters: {'n_estimators': 170, 'max_depth': 4, 'l
[I 2025-03-10 07:43:15,294] Trial 7 finished with value: 0.5274841437632135 and parameters: {'n_estimators': 274, 'max_depth': 4, 'l
[I 2025-03-10 07:43:16,514] Trial 8 finished with value: 0.48414376321353064 and parameters: {'n_estimators': 297, 'max_depth': 6,
[I 2025-03-10 07:43:16,934] Trial 9 finished with value: 0.5116279069767442 and parameters: {'n_estimators': 128, 'max_depth': 5, 'l
[I 2025-03-10 07:43:17,161] Trial 10 finished with value: 0.5095137420718816 and parameters: {'n_estimators': 208, 'max_depth': 2,
[I 2025-03-10 07:43:17,688] Trial 11 finished with value: 0.514799154334038 and parameters: {'n_estimators': 300, 'max_depth': 4, 'l
[I 2025-03-10 07:43:19,701] Trial 12 finished with value: 0.5137420718816068 and parameters: {'n_estimators': 248, 'max_depth': 8,
[I 2025-03-10 07:43:19,893] Trial 13 finished with value: 0.5137420718816068 and parameters: {'n_estimators': 220, 'max_depth': 2,
[I 2025-03-10 07:43:20,610] Trial 14 finished with value: 0.5306553911205074 and parameters: {'n_estimators': 272, 'max_depth': 5,
[I 2025-03-10 07:43:21,280] Trial 15 finished with value: 0.5010570824524313 and parameters: {'n_estimators': 153, 'max_depth': 7,
[I 2025-03-10 07:43:21,901] Trial 16 finished with value: 0.4799154334038055 and parameters: {'n_estimators': 220, 'max_depth': 5,
[I 2025-03-10 07:43:22,110] Trial 17 finished with value: 0.5179704016913319 and parameters: {'n_estimators': 57, 'max_depth': 5, 'l
[I 2025-03-10 07:43:23,571] Trial 18 finished with value: 0.5348837209302325 and parameters: {'n_estimators': 280, 'max_depth': 7,
[I 2025-03-10 07:43:28,146] Trial 19 finished with value: 0.5306553911205074 and parameters: {'n_estimators': 229, 'max_depth': 10,
[I 2025-03-10 07:43:30,000] Trial 20 finished with value: 0.5285412262156448 and parameters: {'n_estimators': 191, 'max_depth': 8,
[I 2025-03-10 07:43:31,467] Trial 21 finished with value: 0.49682875264270615 and parameters: {'n_estimators': 279, 'max_depth': 7,
[I 2025-03-10 07:43:32,892] Trial 22 finished with value: 0.5105708245243129 and parameters: {'n_estimators': 269, 'max_depth': 7,
[I 2025-03-10 07:43:33,637] Trial 23 finished with value: 0.514799154334038 and parameters: {'n_estimators': 240, 'max_depth': 5, 'l
[I 2025-03-10 07:43:33,966] Trial 24 finished with value: 0.5274841437632135 and parameters: {'n_estimators': 279, 'max_depth': 3,
[I 2025-03-10 07:43:35,490] Trial 25 finished with value: 0.5221987315010571 and parameters: {'n_estimators': 266, 'max_depth': 9,
[I 2025-03-10 07:43:36,535] Trial 26 finished with value: 0.5465116279069767 and parameters: {'n_estimators': 291, 'max_depth': 6,
[I 2025-03-10 07:43:39,382] Trial 27 finished with value: 0.5126849894291755 and parameters: {'n_estimators': 298, 'max_depth': 6,
[I 2025-03-10 07:43:40,695] Trial 28 finished with value: 0.5158562367864693 and parameters: {'n_estimators': 199, 'max_depth': 7,
[I 2025-03-10 07:43:41,834] Trial 29 finished with value: 0.5190274841437632 and parameters: {'n_estimators': 177, 'max_depth': 9,
Best parameters: {'n_estimators': 291, 'max_depth': 6, 'learning_rate': 0.23698562104876836, 'subsample': 0.6442546292269913, 'colsa
Optimized Model Accuracy: 0.5106
```

Feature Selection & Engineering

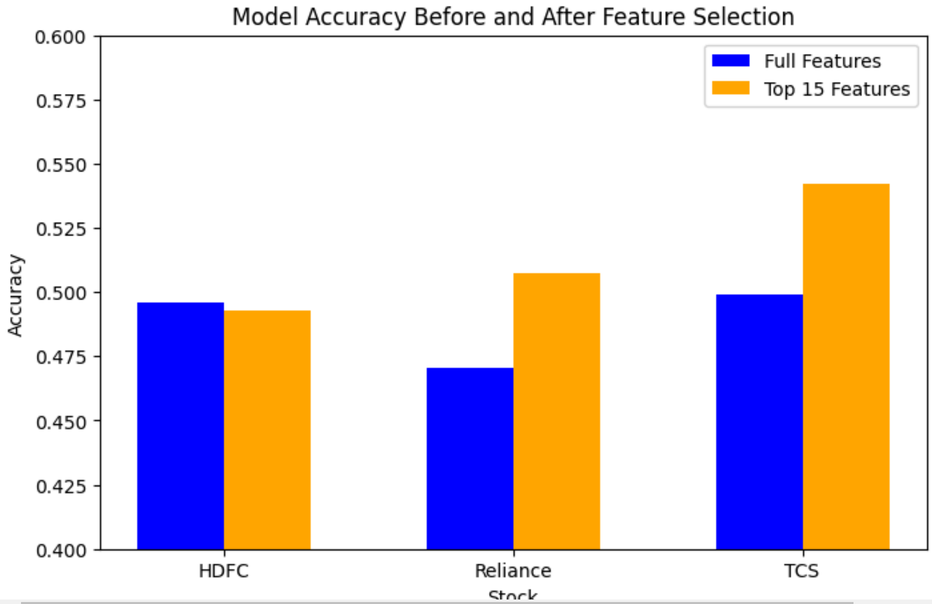
```
import xgboost as xgb
import matplotlib.pyplot as plt

# Train a simple XGBoost model for HDFC (repeat for others)
model = xgb.XGBClassifier(n_estimators=291, max_depth=6, learning_rate=0.237,
                          subsample=0.644, colsample_bytree=0.561, random_state=42)
model.fit(scaled_data['HDFC_Target']['X_train'], scaled_data['HDFC_Target']['y_train'])

# Get feature importance
xgb.plot_importance(model, max_num_features=15)  # Show top 15 important features
plt.show()
```



```
import pandas as pd
import numpy as np
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load dataset
df = pd.read_csv("enhanced_stock_data.csv")

# Define targets
targets = ['HDFC_Target', 'Reliance_Target', 'TCS_Target']

# Feature importance from previous model (manual mapping)
important_features = {
    'HDFC_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5'],
    'Reliance_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5'],
    'TCS_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5']
}

# Convert feature names (f1, f2, ...) to actual column names
selected_features = {
    target: [df.columns[int(f[1:]) - 1] for f in important_features[target]]  # Convert "f1" -> real column name
    for target in targets
}

# Standardize data
scaler = StandardScaler()

# Store accuracies for comparison
baseline_accuracies = []
optimized_accuracies = []

for stock in targets:
    # Drop non-numeric columns like Date
    X_full = df.drop(columns=targets)

    # Convert all columns to numeric (handles any accidental strings)
    X_full = X_full.apply(pd.to_numeric, errors='coerce')

    # Fill NaN values with 0 (to avoid issues in StandardScaler)
    X_full = X_full.fillna(0)

    # Define target variable
    y = df[stock]

    # Train-test split
    X_train_full, X_test_full, y_train, y_test = train_test_split(
        X_full, y, test_size=0.2, shuffle=False, random_state=42
    )

    # Scale data
    X_train_full_scaled = scaler.fit_transform(X_train_full)
    X_test_full_scaled = scaler.transform(X_test_full)

    # Train XGBoost (Full Features)
    model_full = xgb.XGBClassifier(
        n_estimators=291,
        max_depth=6,
        learning_rate=0.2369,
        subsample=0.6442,
        colsample_bytree=0.5606,
        use_label_encoder=False,
        eval_metric="logloss"
    )
    model_full.fit(X_train_full_scaled, y_train)
    accuracy_full = model_full.score(X_test_full_scaled, y_test)
    baseline_accuracies.append(accuracy_full)

    # Train on top 15 features
    X_selected = df[selected_features[stock]]

    # Convert to numeric & fill NaN
    X_selected = X_selected.apply(pd.to_numeric, errors='coerce')
    X_selected = X_selected.fillna(0)

    # Train-test split
    X_train_selected, X_test_selected, _, _ = train_test_split(
        X_selected, y, test_size=0.2, shuffle=False, random_state=42
    )

    # Scale data
```

```
    X_train_selected_scaled = scaler.fit_transform(X_train_selected)
    X_test_selected_scaled = scaler.transform(X_test_selected)

    # Train XGBoost (Reduced Features)
    model_selected = xgb.XGBClassifier(
        n_estimators=291,
        max_depth=6,
        learning_rate=0.2369,
        subsample=0.6442,
        colsample_bytree=0.5606,
        use_label_encoder=False,
        eval_metric="logloss"
    )
    model_selected.fit(X_train_selected_scaled, y_train)
    accuracy_selected = model_selected.score(X_test_selected_scaled, y_test)
    optimized_accuracies.append(accuracy_selected)

# Plot comparison
plt.figure(figsize=(8, 5))
x_labels = ['HDFC', 'Reliance', 'TCS']
x = np.arange(len(x_labels))
width = 0.3

plt.bar(x - width/2, baseline_accuracies, width, label="Full Features", color="blue")
plt.bar(x + width/2, optimized_accuracies, width, label="Top 15 Features", color="orange")

plt.xlabel("Stock")
plt.ylabel("Accuracy")
plt.title("Model Accuracy Before and After Feature Selection")
plt.xticks(x, x_labels)
plt.legend()
plt.ylim(0.4, 0.6)  # Adjust based on observed accuracy range

plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:55:57] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:56:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:56:09] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:56:12] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:56:13] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:56:16] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```



```
import pandas as pd
import numpy as np
import xgboost as xgb
import matplotlib.pyplot as plt
import optuna
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
# from ta.momentum import RSIIndicator  # For adding RSI
# from ta.trend import SMAIndicator  # Simple Moving Average

# Load dataset
df = pd.read_csv("enhanced_stock_data.csv")

# Ensure Date is in datetime format (optional for future use)
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.drop(columns=['Date'])  # Drop Date column if it exists

# Define targets
targets = ['HDFC_Target', 'Reliance_Target', 'TCS_Target']

# Add optional technical indicators (Uncomment if needed)
def add_technical_indicators(df):
    for window in [10, 20, 50]:  # Moving Averages
        df[f"SMA_{window}"] = SMAIndicator(df['Close'], window).sma_indicator()
    df["RSI"] = RSIIndicator(df["Close"], window=14).rsi()
    return df

# df = add_technical_indicators(df)
df = df.fillna(method='bfill')  # Fill missing values after adding indicators

# Important features mapping (replace with actual column names if needed)
important_features = {
    'HDFC_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5'],
    'Reliance_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5'],
    'TCS_Target': ['f1', 'f2', 'f6', 'f9', 'f3', 'f19', 'f4', 'f13', 'f7', 'f18', 'f17', 'f8', 'f10', 'f11', 'f5']
}

# Map back to real column names
selected_features = {
    target: [df.columns[int(f[1:]) - 1] for f in important_features[target]]
    for target in targets
}

# Standardize data
scaler = StandardScaler()

# Function to optimize XGBoost hyperparameters using Optuna
def objective(trial, X_train, y_train):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 50, 500),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.5, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0)
    }

    model = xgb.XGBClassifier(**params, use_label_encoder=False, eval_metric="logloss")
    model.fit(X_train, y_train)
    return model.score(X_train, y_train)  # Optimizing on training accuracy

# Store accuracies for comparison
baseline_accuracies = []
optimized_accuracies = []

for stock in targets:
    # Prepare full dataset (remove non-numeric columns)
    X_full = df.drop(columns=targets).select_dtypes(include=[np.number])
    y = df[stock]

    # Train-test split
    X_train_full, X_test_full, y_train, y_test = train_test_split(X_full, y, test_size=0.2, shuffle=False, random_state=42

    # Scale data
    X_train_full_scaled = scaler.fit_transform(X_train_full)
    X_test_full_scaled = scaler.transform(X_test_full)

    # Train baseline XGBoost model
    model_full = xgb.XGBClassifier(n_estimators=291, max_depth=6, learning_rate=0.2369,
                                   subsample=0.6442, colsample_bytree=0.5606,
                                   use_label_encoder=False, eval_metric="logloss")
    model_full.fit(X_train_full_scaled, y_train)
```

```
        accuracy_full = model_full.score(X_test_full_scaled, y_test)
        baseline_accuracies.append(accuracy_full)

        # Optimize model with Optuna
        study = optuna.create_study(direction="maximize")
        study.optimize(lambda trial: objective(trial, X_train_full_scaled, y_train), n_trials=30)

        best_params = study.best_params

        # Train optimized XGBoost model
        model_optimized = xgb.XGBClassifier(**best_params, use_label_encoder=False, eval_metric="logloss")
        model_optimized.fit(X_train_full_scaled, y_train)
        accuracy_optimized = model_optimized.score(X_test_full_scaled, y_test)
        optimized_accuracies.append(accuracy_optimized)

        print(f"\n{stock} Best Parameters: {best_params}")
        print(f"Baseline Accuracy: {accuracy_full:.4f} → Optimized Accuracy: {accuracy_optimized:.4f}")

    # Plot accuracy comparison
    plt.figure(figsize=(8, 5))
    x_labels = ['HDFC', 'Reliance', 'TCS']
    x = np.arange(len(x_labels))
    width = 0.3

    plt.bar(x - width/2, baseline_accuracies, width, label="Full Features", color="blue")
    plt.bar(x + width/2, optimized_accuracies, width, label="Optimized Model", color="orange")

    plt.xlabel("Stock")
    plt.ylabel("Accuracy")
    plt.title("Model Accuracy Before and After Hyperparameter Tuning")
    plt.xticks(x, x_labels)
    plt.legend()
    plt.ylim(0.4, 0.65)  # Adjusted range

    plt.show()
```