databricks New_Model_Development_evening

(https://databricks.com)

```
%pip install shap
```

```
# Databricks notebook source
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error
import shap
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Step 1: Load your data
# Assuming data is stored in DBFS or uploaded to Databricks
file_path = "/dbfs/mnt/riskpredict-data/BANKBARODA.NS.csv"
data = pd.read_csv(file_path)
```

```
# Inspect the DataFrame
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1236 entries, 0 to 1235
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1236 non-null   object
 1   Open       1236 non-null   float64
 2   High       1236 non-null   float64
 3   Low        1236 non-null   float64
 4   Close      1236 non-null   float64
 5   Adj Close  1236 non-null   float64
 6   Volume     1236 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 67.7+ KB
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2019-06-17	119.349998	119.449997	116.050003	116.599998	109.915092	16409459
1	2019-06-18	117.300003	118.400002	115.099998	116.500000	109.820824	19869838
2	2019-06-19	118.000000	119.199997	114.300003	115.800003	109.160957	17739731
3	2019-06-20	116.199997	119.000000	115.250000	118.500000	111.706161	17575444
4	2019-06-21	118.599998	119.449997	117.199997	118.050003	111.281967	19336018

```
# Identify the target column
target_column = 'Close'
```

```
# Step 2: Data Cleaning
def clean_data(df):
    df = df.dropna()  # Handle missing values
    df = df.drop_duplicates()  # Remove duplicates
    return df

data = clean_data(data)
```

```
# Step 3: Feature Engineering
def feature_engineering(df, target_column):
    # Example feature engineering steps
    df['lag_1'] = df[target_column].shift(1)
    df['rolling_mean_3'] = df[target_column].rolling(window=3).mean()
    df = df.dropna()  # Drop rows with NaN values after feature engineering
    return df

data = feature_engineering(data, target_column)
```

```
X = data.drop(columns=[target_column, 'Date'])
y = data[target_column]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step 5: Model Training with Random Forest
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

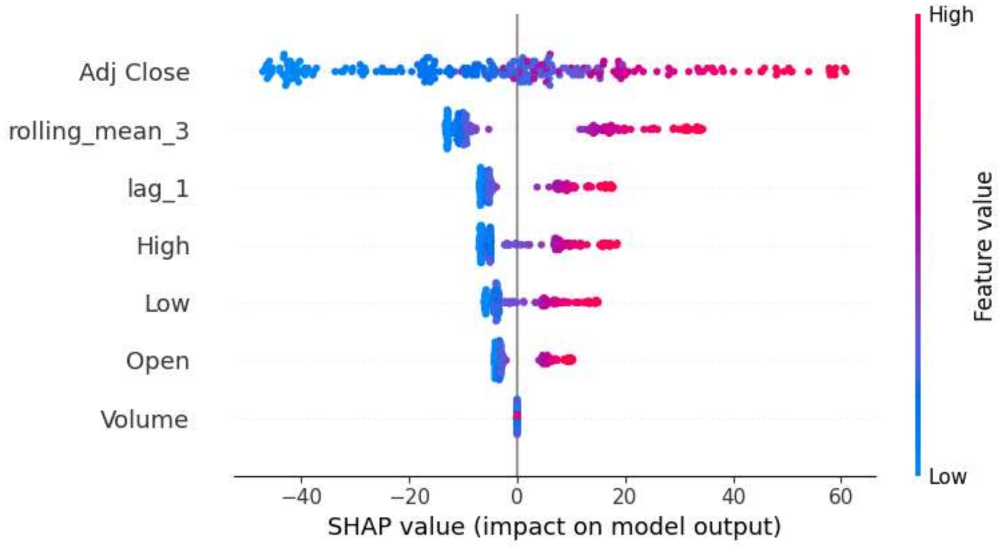
```
# Model Prediction and Evaluation
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Random Forest MSE: {mse}')
```

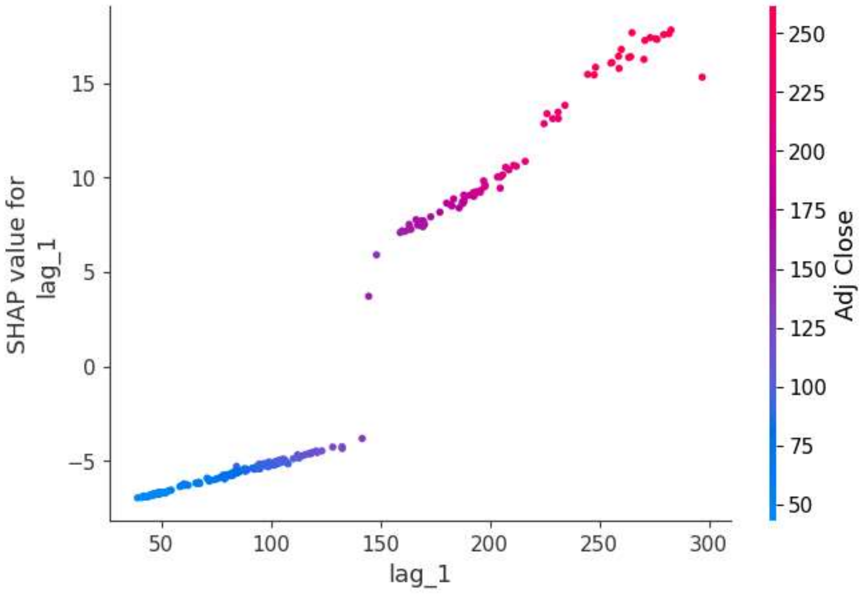
```
Random Forest MSE: 1.7127508912863194
```

```
# Step 6: Hyperparameter Tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

best_rf_model = grid_search.best_estimator_
best_rf_pred = best_rf_model.predict(X_test)
best_mse = mean_squared_error(y_test, best_rf_pred)
print(f'Best Random Forest MSE after Hyperparameter Tuning: {best_mse}')
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Random Forest MSE after Hyperparameter Tuning: 1.7193213043098032
```

```
# Step 7: Model Interpretability with SHAP
explainer = shap.TreeExplainer(best_rf_model)
shap_values = explainer.shap_values(X_test)

shap.summary_plot(shap_values, X_test)
shap.dependence_plot("lag_1", shap_values, X_test)  # Example dependence plot
```





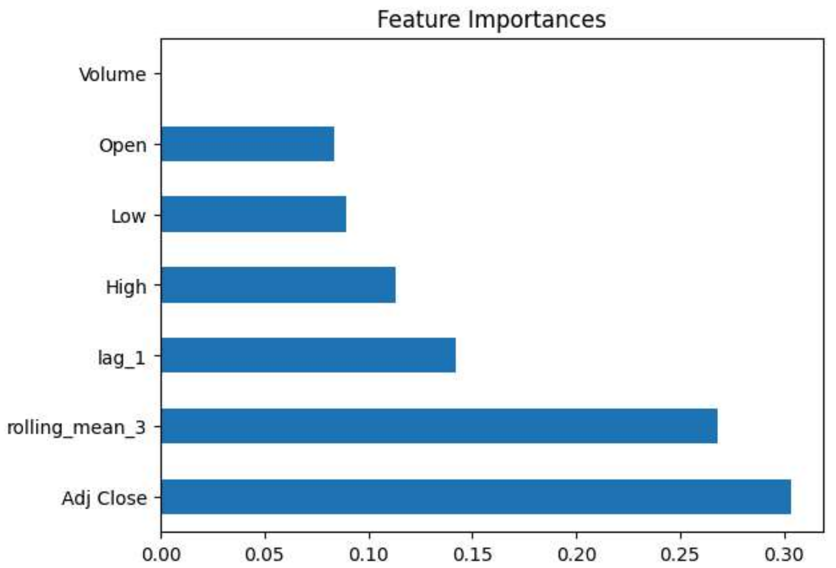
```
# Step 8: Trying Gradient Boosting Machine
gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
gb_pred = gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_pred)
print(f'Gradient Boosting MSE: {gb_mse}')
```

```
Gradient Boosting MSE: 1.7802678492558304
```

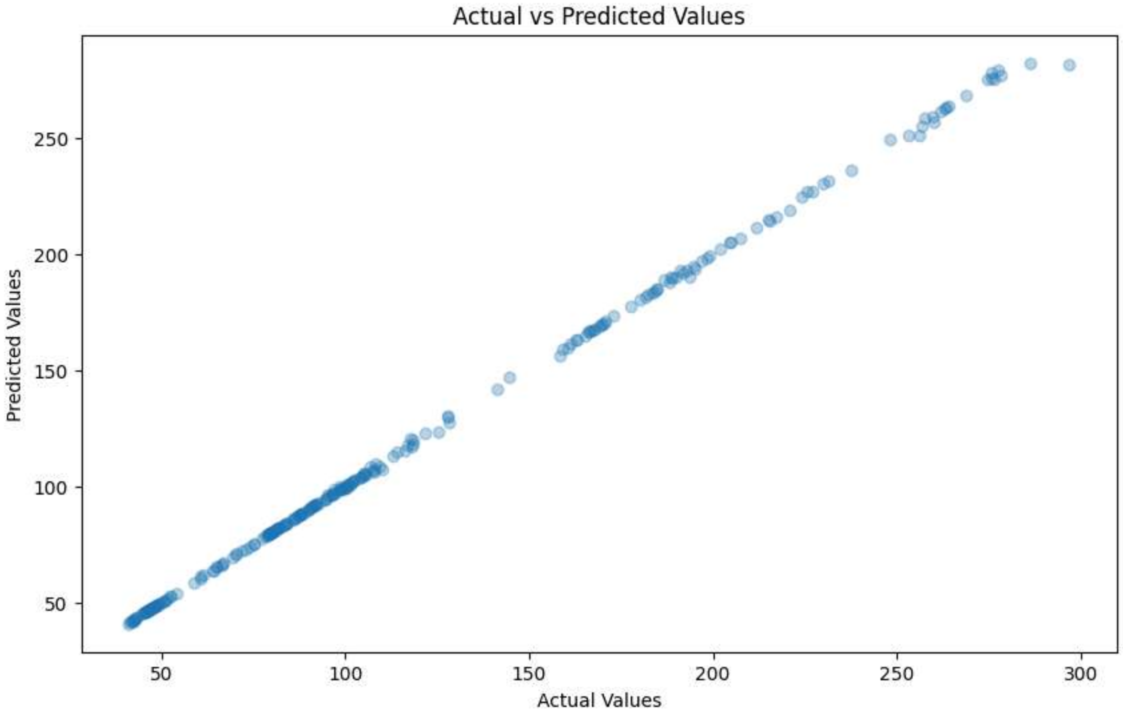
```
# Step 9: Cross-Validation
cv_scores = cross_val_score(best_rf_model, X, y, cv=5)
print(f'Cross-Validation Scores: {cv_scores}')
print(f'Average CV Score: {np.mean(cv_scores)}')
```

```
Cross-Validation Scores: [ 0.99851044  0.99809576  0.9965745   0.83336902 -1.31982439]
Average CV Score: 0.501345067260613
```

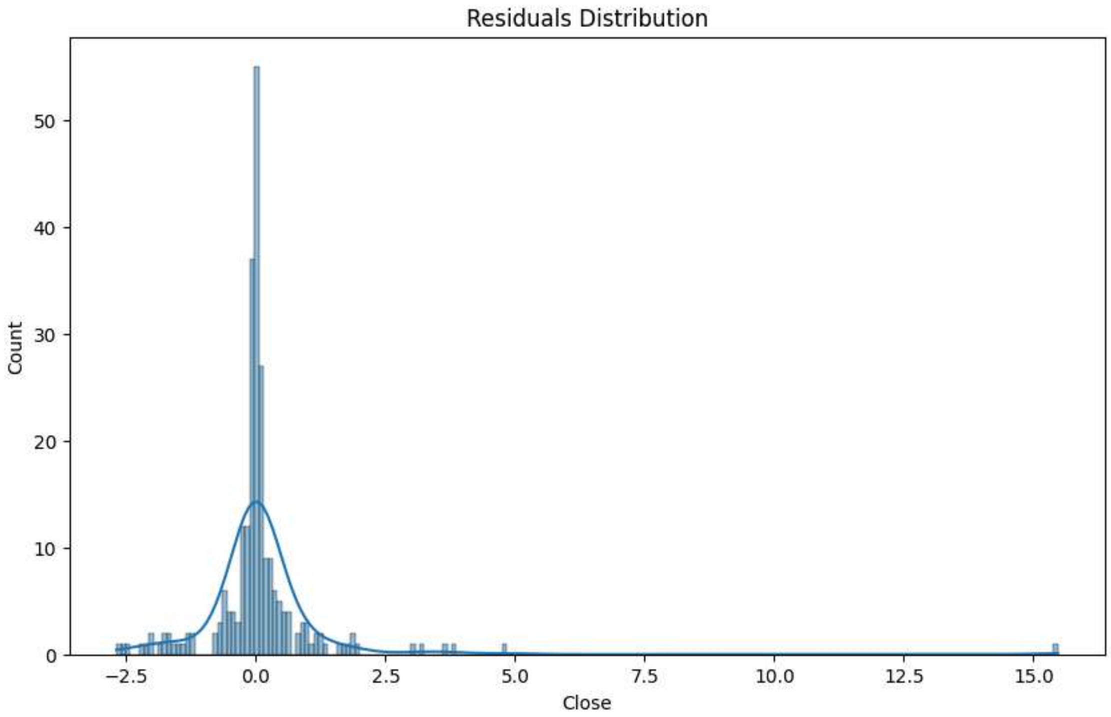
```
# Step 10: Visualization of Results
# Example: Feature importances visualization
feature_importances = pd.Series(best_rf_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh')
plt.title('Feature Importances')
plt.show()
```




```
# Example: Actual vs Predicted plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, best_rf_pred, alpha=0.3)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.show()
```



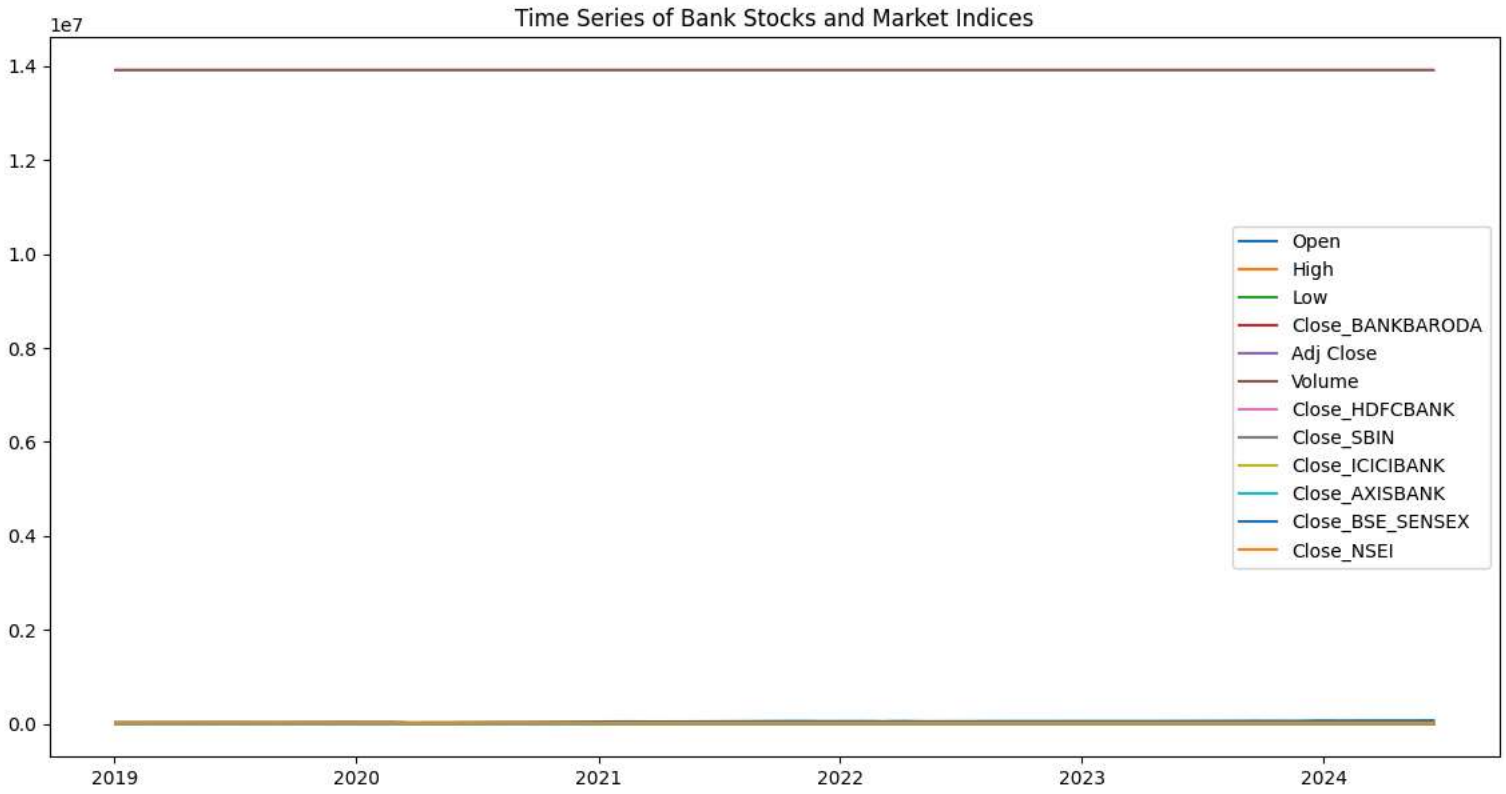
```
# Example: Residual plot
residuals = y_test - best_rf_pred
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.show()
```

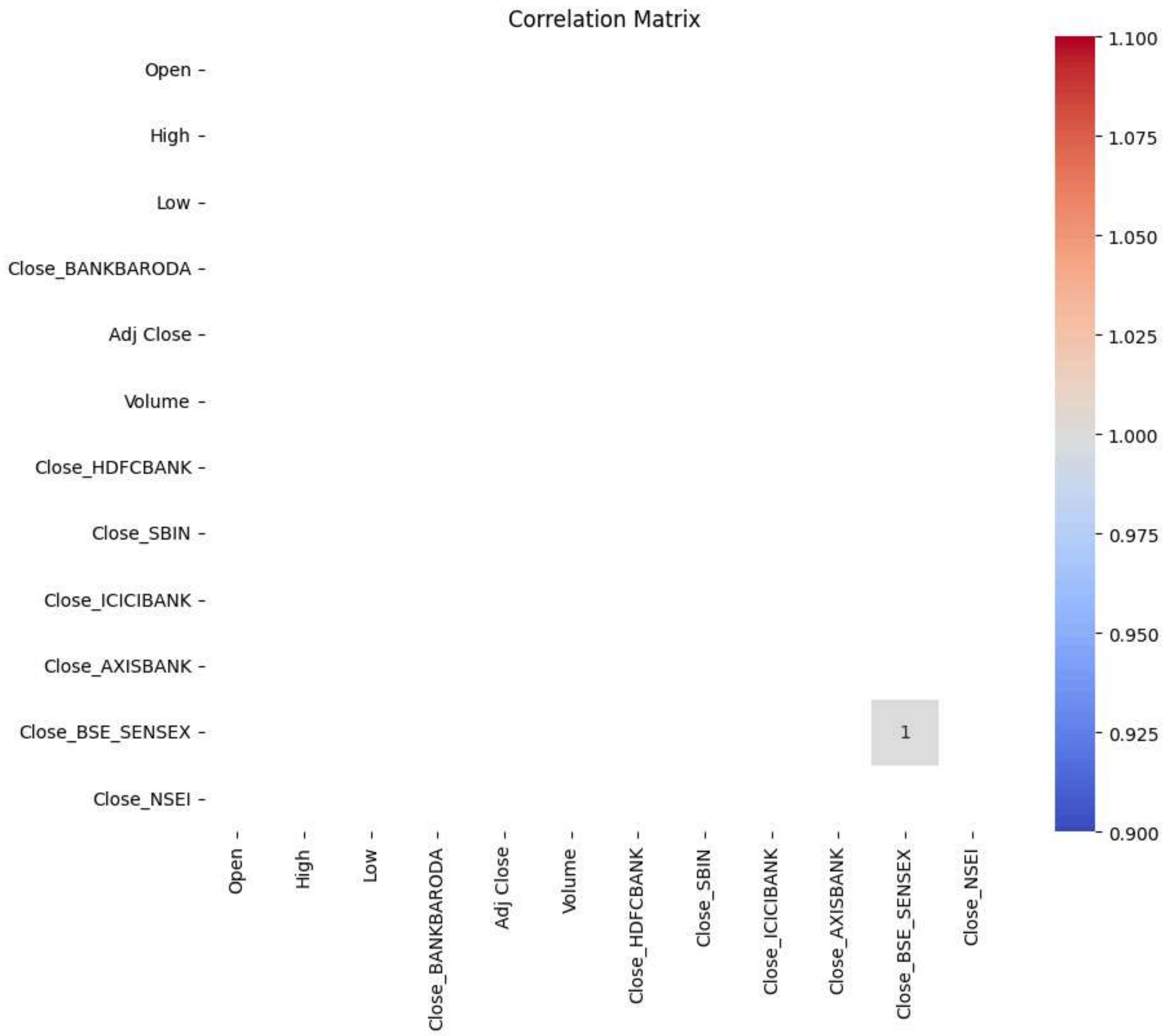



```
import pandas as pd

# Define paths to the dataset files
datasets = {
    "BANKBARODA": "/dbfs/mnt/riskpredict-data/BANKBARODA.NS.csv",
    "HDFCBANK": "/dbfs/mnt/riskpredict-data/HDFCBANK.csv",
    "SBIN": "/dbfs/mnt/riskpredict-data/SBIN.csv",
    "ICICIBANK": "/dbfs/mnt/riskpredict-data/ICICIBANK.csv",
    "AXISBANK": "/dbfs/mnt/riskpredict-data/AXISBANK.csv",
    "BSE_SENSEX": "/dbfs/mnt/riskpredict-data/BSE_Sensex.csv",
    "NSEI": "/dbfs/mnt/riskpredict-data/Nifty50.csv"
}

# Load datasets into DataFrames and ensure 'Date' is in datetime format
dataframes = {}
for name, path in datasets.items():
    df = pd.read_csv(path)
    df['Date'] = pd.to_datetime(df['Date'])
    # Rename 'Close' column to reflect the dataset name
    df.rename(columns={"Close": f"Close_{name}"}, inplace=True)
    dataframes[name] = df

# Merge all DataFrames on 'Date'
merged_df = pd.DataFrame()
for name, df in dataframes.items():
    if merged_df.empty:
        merged_df = df
    else:
        merged_df = pd.merge(merged_df, df[['Date', f"Close_{name}"]], on='Date', how='outer')

# Handle missing values with forward fill
merged_df.fillna(method='ffill', inplace=True)

# Drop rows that still have missing values after forward fill
merged_df.dropna(inplace=True)

# Save the prepared dataset to a new CSV file for further analysis
merged_df.to_csv("/dbfs/mnt/riskpredict-data/merged_dataset.csv", index=False)

print("Merged dataset prepared and saved.")
```

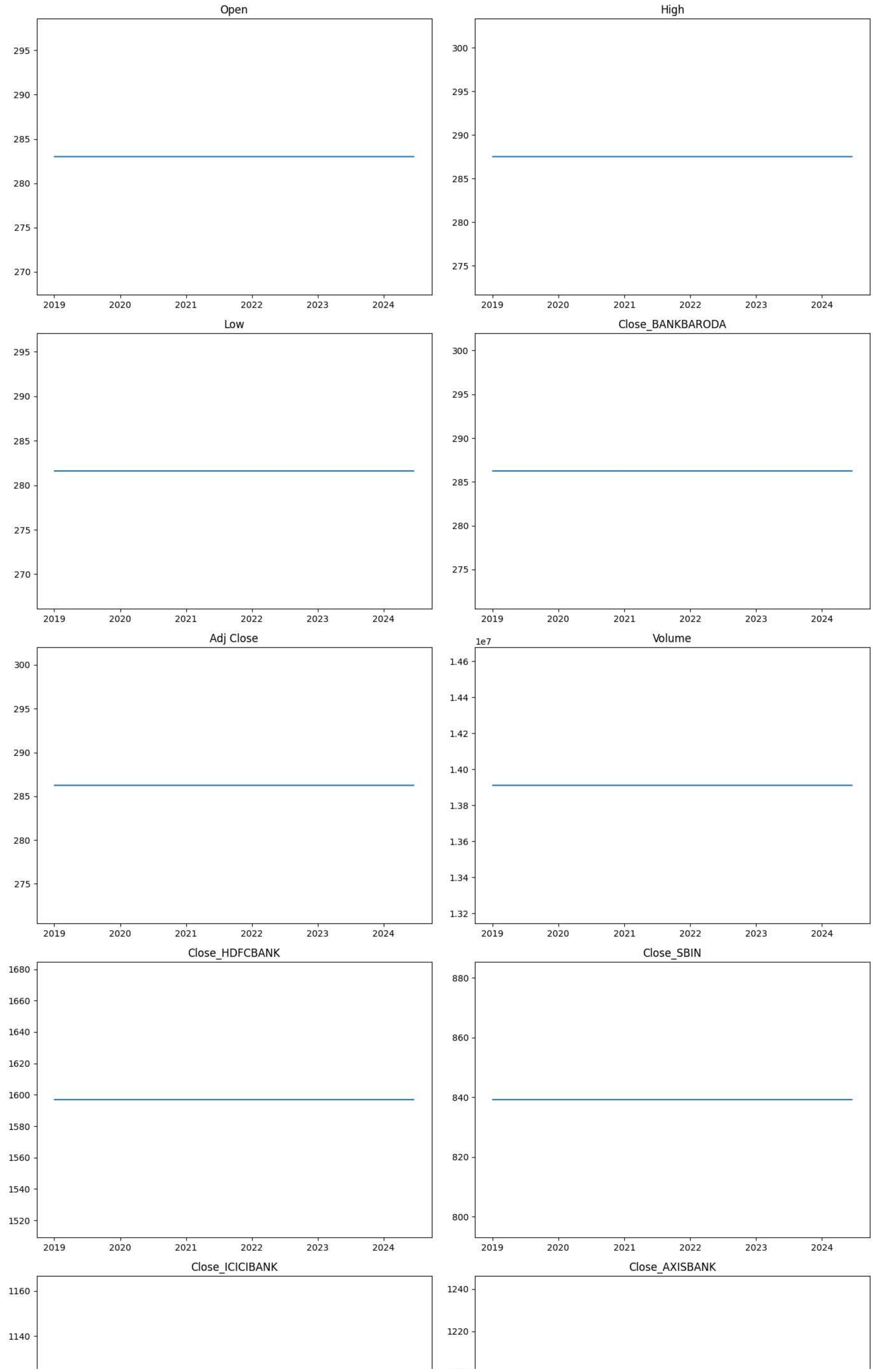
```
Merged dataset prepared and saved.
```

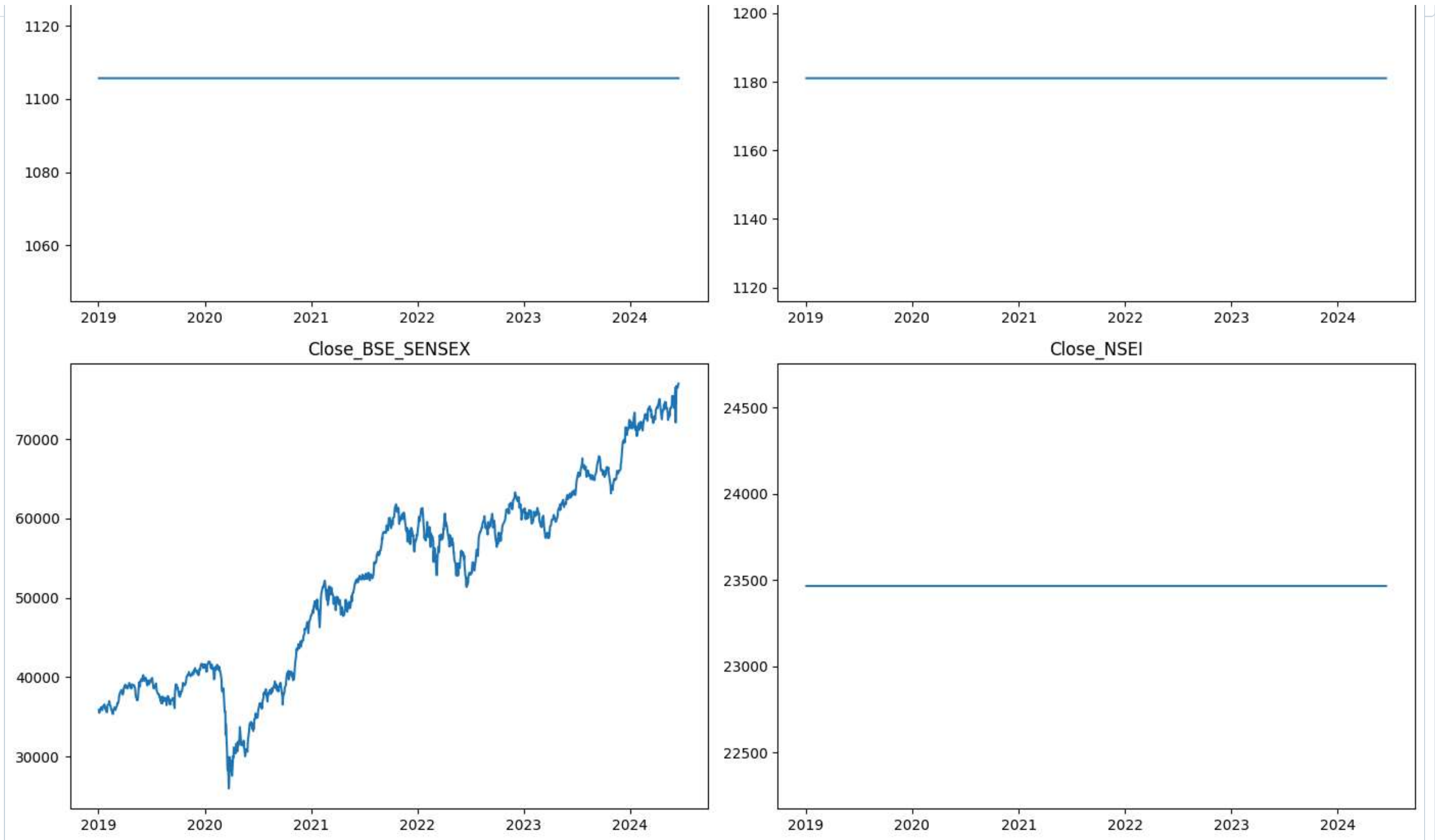
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the merged dataset
df = pd.read_csv("/dbfs/mnt/riskpredict-data/merged_dataset.csv")
df['Date'] = pd.to_datetime(df['Date'])

# Plot time series
plt.figure(figsize=(14, 7))
for column in df.columns[1:]:  # Skip 'Date' column
    plt.plot(df['Date'], df[column], label=column)
plt.title('Time Series of Bank Stocks and Market Indices')
plt.legend()
plt.show()

# Correlation matrix
corr_matrix = df.iloc[:, 1:].corr()  # Exclude 'Date' column
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Statistical summary
print(df.describe())
```






```
        Open    High  ...  Close_BSE_SENSEX    Close_NSEI
count  1344.0  1344.0  ...       1344.000000  1.344000e+03
mean    283.0   287.5  ...      52183.375618  2.346560e+04
std       0.0     0.0  ...      12290.557728  3.639333e-12
min     283.0   287.5  ...      25981.240000  2.346560e+04
25%     283.0   287.5  ...      39666.442500  2.346560e+04
50%     283.0   287.5  ...      54230.530000  2.346560e+04
75%     283.0   287.5  ...      60846.017500  2.346560e+04
max     283.0   287.5  ...      76992.770000  2.346560e+04

[8 rows x 12 columns]
```


Open
295
290
285
280
275
270
2019 2020 2021 2022 2023 2024
High
300
295
290
285
280
275
2019 2020 2021 2022 2023 2024
Low
295
290
285
280
275
270
2019 2020 2021 2022 2023 2024
Close_BANKBARODA
300
295
290
285
280
275
2019 2020 2021 2022 2023 2024
Adj Close
300
295
290
285
280
275
2019 2020 2021 2022 2023 2024
Volume
1e7
1.46
1.44
1.42
1.40
1.38
1.36
1.34
1.32
2019 2020 2021 2022 2023 2024
Close_HDFCBANK
1680
1660
1640
1620
1600
1580
1560
1540
1520
2019 2020 2021 2022 2023 2024
Close_SBIN
880
860
840
820
800
2019 2020 2021 2022 2023 2024
Close_ICICIBANK
1160
1140
Close_AXISBANK
1240
1220


1120
1100
1080
1060
2019
2020
2021
2022
2023
2024
1200
1180
1160
1140
1120
2019
2020
2021
2022
2023
2024
Close_BSE_SENSEX
70000
60000
50000
40000
30000
2019
2020
2021
2022
2023
2024
Close_NSEI
24500
24000
23500
23000
22500
2019
2020
2021
2022
2023
2024