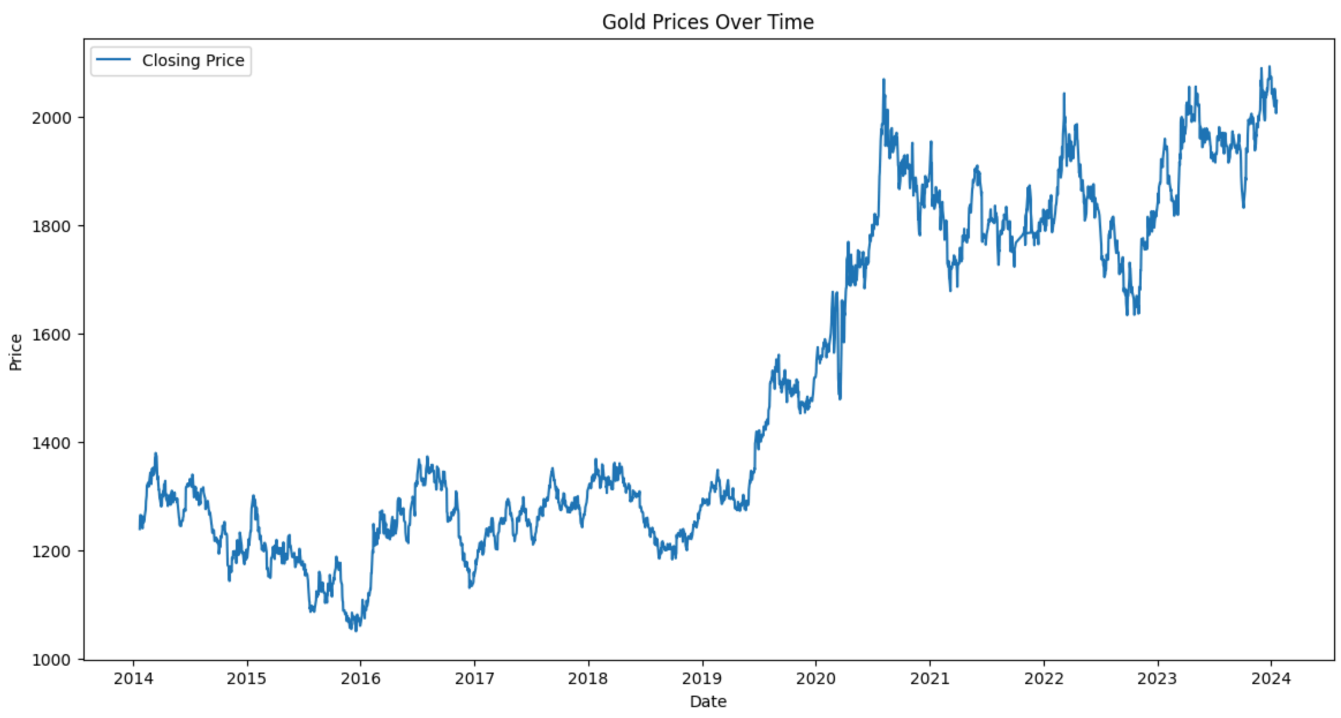
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/goldstock.csv'
df = pd.read_csv(file_path)

# Drop the unnecessary index column if exists
if 'Unnamed: 0' in df.columns:
    df.drop(columns=['Unnamed: 0'], inplace=True)

# Convert 'Date' column to datetime format and set it as the index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Display the first few rows and summary information of the dataset
print(df.head())
print(df.info())
```

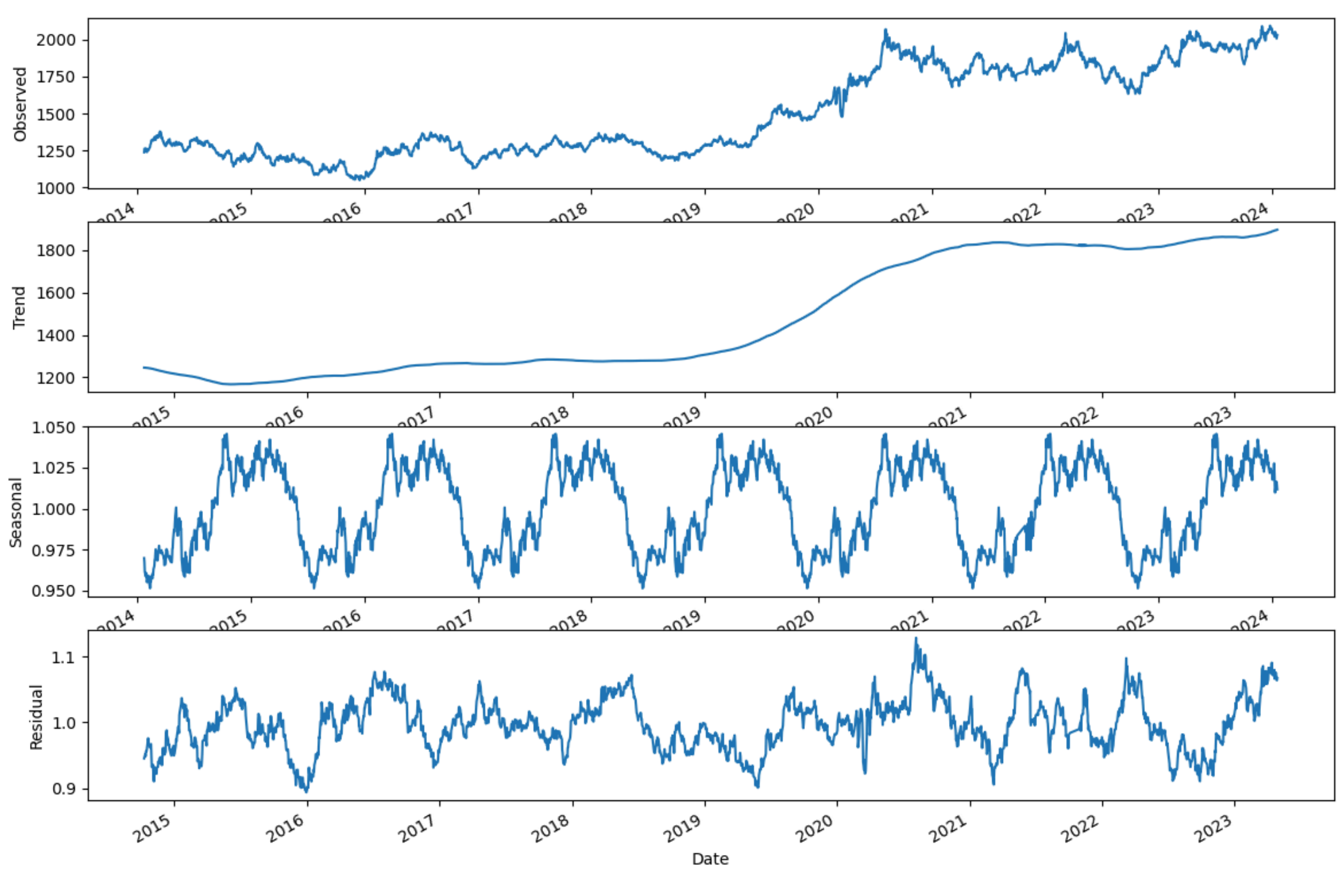
```
             Close    Volume    Open    High     Low
Date
2024-01-19  2029.3  166078.0  2027.4  2041.9  2022.2
2024-01-18  2021.6  167013.0  2009.1  2025.6  2007.7
2024-01-17  2006.5  245194.0  2031.7  2036.1  2004.6
2024-01-16  2030.2  277995.0  2053.4  2062.8  2027.6
2024-01-12  2051.6  250946.0  2033.2  2067.3  2033.1
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2511 entries, 2024-01-19 to 2014-01-22
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Close   2511 non-null   float64
 1   Volume  2511 non-null   float64
 2   Open    2511 non-null   float64
 3   High    2511 non-null   float64
 4   Low     2511 non-null   float64
dtypes: float64(5)
memory usage: 117.7 KB
None
```

```
#initial EDA
# Plot the closing price over time
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Closing Price')
plt.title('Gold Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

# Display summary statistics
summary_stats = df.describe()
print(summary_stats)
```



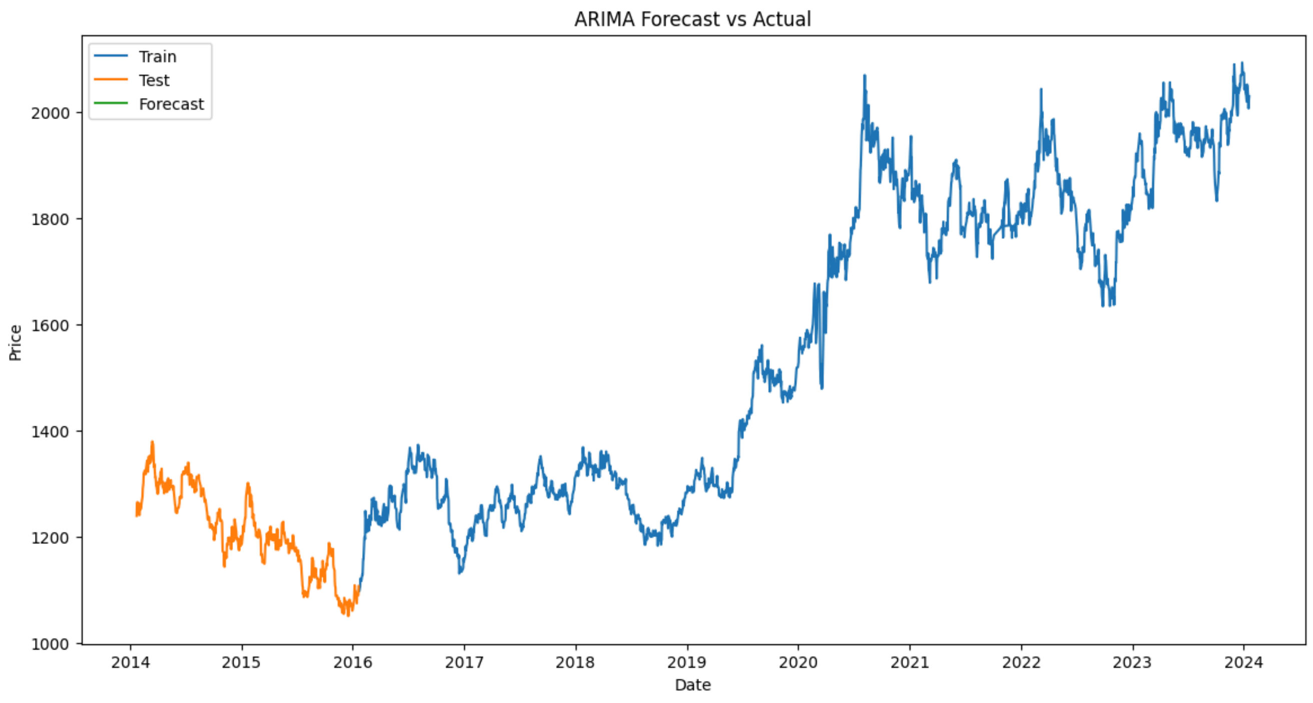
```
             Close         Volume         Open         High          Low
count  2511.000000    2511.000000  2511.000000  2511.000000  2511.000000
mean   1498.726085  185970.770609  1498.725528  1508.451454  1488.869932
std     298.824811   97600.769382   299.118187   301.262244   296.417703
min    1049.600000       1.000000  1051.500000  1062.700000  1045.400000
25%    1249.850000  126693.500000  1249.500000  1257.300000  1242.350000
50%    1332.800000  175421.000000  1334.000000  1342.400000  1326.600000
75%    1805.850000  234832.000000  1805.600000  1815.450000  1793.050000
max    2093.100000  787217.000000  2094.400000  2098.200000  2074.600000
```

```
#time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df['Close'], model='multiplicative', period=365)

# Plot decomposed components
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(14, 10))
result.observed.plot(ax=ax1, legend=False)
ax1.set_ylabel('Observed')
result.trend.plot(ax=ax2, legend=False)
ax2.set_ylabel('Trend')
result.seasonal.plot(ax=ax3, legend=False)
ax3.set_ylabel('Seasonal')
result.resid.plot(ax=ax4, legend=False)
ax4.set_ylabel('Residual')
plt.show()
```


Observed
2000
1750
1500
1250
1000
Trend
1800
1600
1400
1200
Seasonal
1.050
1.025
1.000
0.975
0.950
Residual
1.1
1.0
0.9
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Date

```
#advanced modeling
#ARIMA model
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import numpy as np

# Split data into train and test
train_size = int(len(df) * 0.8)
train, test = df['Close'][:train_size], df['Close'][train_size:]

# Fit ARIMA model
model = ARIMA(train, order=(5, 1, 0))
model_fit = model.fit()

# Forecast
forecast = model_fit.forecast(steps=len(test))

# Ensure the forecast length matches the test data length
forecast = forecast[:len(test)]

# Calculate confidence intervals
fc_series = pd.Series(forecast, index=test.index)
lower_series = pd.Series(model_fit.get_forecast(steps=len(test)).conf_int()['lower Close'], index=test.index)
upper_series = pd.Series(model_fit.get_forecast(steps=len(test)).conf_int()['upper Close'], index=test.index)

# Plot forecast vs actual
plt.figure(figsize=(14, 7))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Test')
plt.plot(fc_series.index, fc_series, label='Forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.15)
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('ARIMA Forecast vs Actual')
plt.legend()
plt.show()

# Evaluate
rmse = mean_squared_error(test, forecast, squared=False)
print(f'RMSE: {rmse}')
```

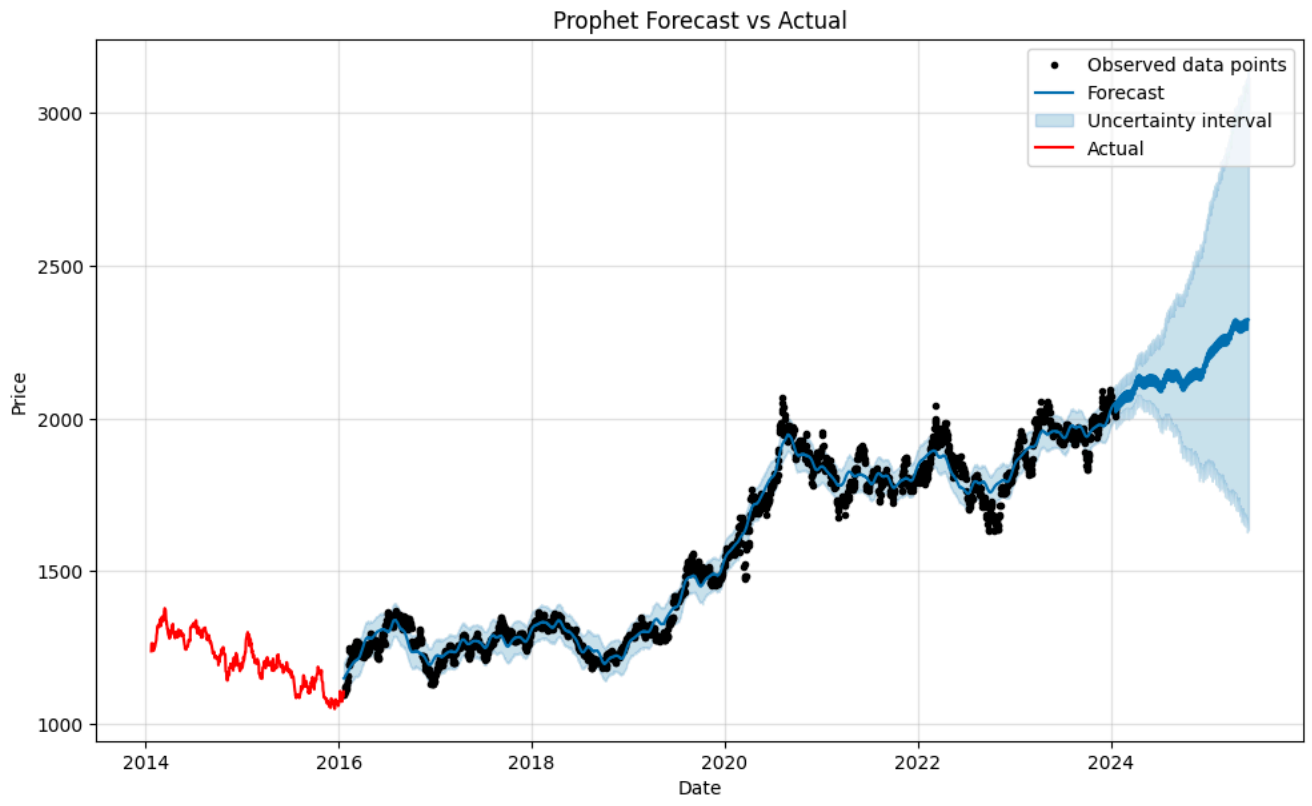
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Pre
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported index is available. In
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Pre
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Pre
  return get_prediction_index(
```



```
RMSE: 135.8767871226694
```

```
#Prophet model
!pip install prophet

from prophet import Prophet
from sklearn.metrics import mean_squared_error

# Prepare data for Prophet
df_prophet = df.reset_index()[['Date', 'Close']].rename(columns={'Date': 'ds', 'Close': 'y'})

# Split data into train and test
train_prophet = df_prophet.iloc[:int(0.8*len(df_prophet))]
test_prophet = df_prophet.iloc[int(0.8*len(df_prophet)):]

# Fit Prophet model
model_prophet = Prophet()
model_prophet.fit(train_prophet)

# Forecast
future = model_prophet.make_future_dataframe(periods=len(test_prophet))
forecast = model_prophet.predict(future)

# Plot forecast vs actual
fig = model_prophet.plot(forecast)
plt.plot(test_prophet['ds'], test_prophet['y'], label='Actual', color='r')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Prophet Forecast vs Actual')
plt.legend()
plt.show()

# Evaluate
forecast_test = forecast.iloc[-len(test_prophet):]
rmse = mean_squared_error(test_prophet['y'], forecast_test['yhat'], squared=False)
print(f'RMSE: {rmse}')
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.3)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.25.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (2.0.3)
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/dist-packages (from prophet) (0.50)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from prophet) (4.66.4)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from prophet) (6.4.0)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.2
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (4.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.1
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->holidays>=0.25->prophet
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpqlei239z/boe573is.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpqlei239z/ym0vg2yj.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=7
14:06:58 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
14:07:00 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```



```
RMSE: 961.5901380556295
```

```
#trading strategy development
#Moving Average Crossover Strategy
# Define the strategy
import numpy as np

# Define the strategy
short_window = 40
long_window = 100

signals = pd.DataFrame(index=df.index)
signals['signal'] = 0.0

# Create short simple moving average
signals['short_mavg'] = df['Close'].rolling(window=short_window, min_periods=1, center=False).mean()

# Create long simple moving average
signals['long_mavg'] = df['Close'].rolling(window=long_window, min_periods=1, center=False).mean()

# Create signals
signals['signal'][short_window:] = np.where(signals['short_mavg'][short_window:] > signals['long_mavg'][short_window:], 1.0, 0.0)

# Generate trading orders
signals['positions'] = signals['signal'].diff()

# Plot signals
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Close Price')
plt.plot(signals['short_mavg'], label='40-day SMA')
plt.plot(signals['long_mavg'], label='100-day SMA')
```