time-analysis-finance-2

September 24, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
```

1 Data Cleaning

[2]:
```
datasets = ['AAPL']


for dataset in datasets :
    Ticker = yf.Ticker(dataset)
    data = Ticker.history(start='2023-01-01' , end='2023-09-23')
    filename = f"{dataset}_data.csv"
    data.to_csv(filename)
    print(f"Download data for {dataset} and saved as {filename}")
```

```
Download data for AAPL and saved as AAPL_data.csv
```

[3]:
```
Ticker = 'AAPL'
start_date = '2019-01-01'
end_date = '2023-09-23'
```

[4]:
```
stock_data = yf.download(Ticker, start=start_date, end=end_date)
```

```
[*********************100%%**********************]  1 of 1 completed
```

[5]:
```
stock_data
```

[5]:
```
                 Open       High        Low      Close  Adj Close  \
Date
2019-01-02  38.722500  39.712502  38.557499  39.480000  37.943260
2019-01-03  35.994999  36.430000  35.500000  35.547501  34.163837
2019-01-04  36.132500  37.137501  35.950001  37.064999  35.622250
2019-01-07  37.174999  37.207500  36.474998  36.982498  35.542976
2019-01-08  37.389999  37.955002  37.130001  37.687500  36.220524
…                 …          …          …          …          …
```

```
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

               Volume
Date
2019-01-02  148158800
2019-01-03  365248800
2019-01-04  234428400
2019-01-07  219111200
2019-01-08  164101200
…                 …
2023-09-18   67257600
2023-09-19   51826900
2023-09-20   58436200
2023-09-21   63047900
2023-09-22   56682928

[1190 rows x 6 columns]
```

[12]:
```
# Data Cleaning
# Remove duplicate row if any

stock_data = stock_data.drop_duplicates()
```

[13]:
```
stock_data
```

[13]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2019-01-02   38.722500   39.712502   38.557499   39.480000   37.943249
2019-01-03   35.994999   36.430000   35.500000   35.547501   34.163818
2019-01-04   36.132500   37.137501   35.950001   37.064999   35.622253
2019-01-07   37.174999   37.207500   36.474998   36.982498   35.542973
2019-01-08   37.389999   37.955002   37.130001   37.687500   36.220531
…                  …          …          …         …          …
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

               Volume
Date
2019-01-02  148158800
2019-01-03  365248800
```

```
2019-01-04  234428400
2019-01-07  219111200
2019-01-08  164101200
…                 …
2023-09-18   67257600
2023-09-19   51826900
2023-09-20   58436200
2023-09-21   63047900
2023-09-22   55110610

[1190 rows x 6 columns]
```

[14]:
```python
#handling Missing Value
#Forward fill missing value in case of gaps in data
stock_data['Close'].fillna(method='ffill', inplace=True)
```

[15]:
```python
stock_data
```

[15]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2019-01-02   38.722500   39.712502   38.557499   39.480000   37.943249
2019-01-03   35.994999   36.430000   35.500000   35.547501   34.163818
2019-01-04   36.132500   37.137501   35.950001   37.064999   35.622253
2019-01-07   37.174999   37.207500   36.474998   36.982498   35.542973
2019-01-08   37.389999   37.955002   37.130001   37.687500   36.220531
…                  …          …           …           …          …
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

               Volume
Date
2019-01-02  148158800
2019-01-03  365248800
2019-01-04  234428400
2019-01-07  219111200
2019-01-08  164101200
…                 …
2023-09-18   67257600
2023-09-19   51826900
2023-09-20   58436200
2023-09-21   63047900
2023-09-22   55110610

[1190 rows x 6 columns]
```

[16]:
```
#calculate Daily Returns
stock_data['Daily Return'] = stock_data['Close'].pct_change() * 100
```

[17]:
```
#Calculate log Return
stock_data['Log_Return'] = (stock_data['Close']/ stock_data['Close'].shift(1)).
 ↪apply(lambda x: None if pd.isnull(x) else (100*(np.log(x))))
```

[18]:
```
stock_data
```

[18]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2019-01-02   38.722500   39.712502   38.557499   39.480000   37.943249
2019-01-03   35.994999   36.430000   35.500000   35.547501   34.163818
2019-01-04   36.132500   37.137501   35.950001   37.064999   35.622253
2019-01-07   37.174999   37.207500   36.474998   36.982498   35.542973
2019-01-08   37.389999   37.955002   37.130001   37.687500   36.220531
…                  …           …           …           …           …
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

               Volume  Daily Return  Log_Return
Date
2019-01-02  148158800           NaN         NaN
2019-01-03  365248800     -9.960737  -10.492436
2019-01-04  234428400      4.268930    4.180324
2019-01-07  219111200     -0.222583   -0.222831
2019-01-08  164101200      1.906312    1.888370
…                 …             …         …
2023-09-18   67257600      1.691336    1.677192
2023-09-19   51826900      0.618085    0.616183
2023-09-20   58436200     -1.999219   -2.019474
2023-09-21   63047900     -0.888947   -0.892922
2023-09-22   55110610      0.494452    0.493234

[1190 rows x 8 columns]
```

[19]:
```
stock_data.dropna(subset=['Daily Return', 'Log_Return'], inplace=True)
```

[20]:
```
stock_data['Cumulative Return'] = (1+ stock_data['Daily Return'] / 100).
 ↪cumprod() - 1
```

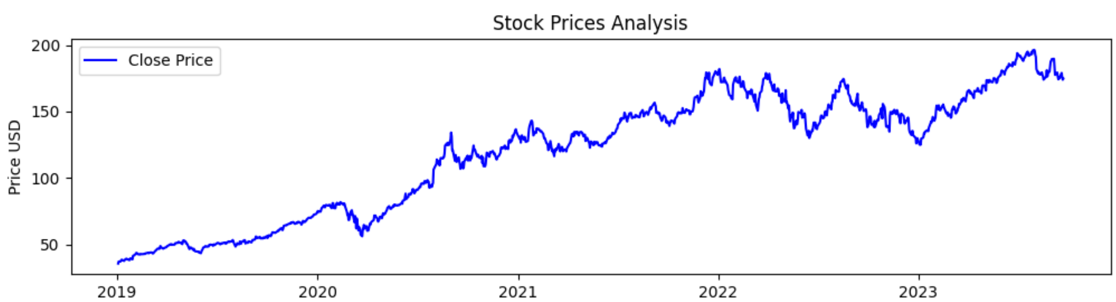
[21]:
```
print(stock_data.head())
```

```
                 Open       High        Low      Close  Adj Close     Volume  \
```

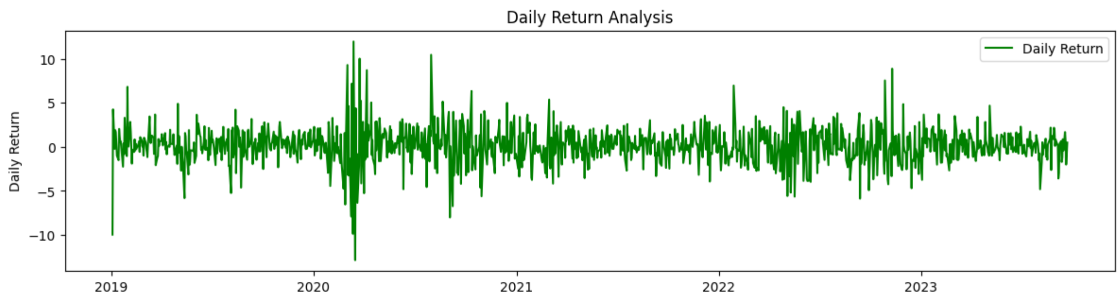
```
Date
2019-01-03  35.994999  36.430000  35.500000  35.547501  34.163818  365248800
2019-01-04  36.132500  37.137501  35.950001  37.064999  35.622253  234428400
2019-01-07  37.174999  37.207500  36.474998  36.982498  35.542973  219111200
2019-01-08  37.389999  37.955002  37.130001  37.687500  36.220531  164101200
2019-01-09  37.822498  38.632500  37.407501  38.327499  36.835617  180396400

            Daily Return  Log_Return  Cumulative Return
Date
2019-01-03     -9.960737  -10.492436          -0.099607
2019-01-04      4.268930    4.180324          -0.061170
2019-01-07     -0.222583   -0.222831          -0.063260
2019-01-08      1.906312    1.888370          -0.045403
2019-01-09      1.698174    1.683916          -0.029192
```

[22]:
```
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1 )
plt.plot(stock_data['Close'], label='Close Price', color='Blue')
plt.title('Stock Prices Analysis')
plt.ylabel('Price USD')
plt.legend()
```

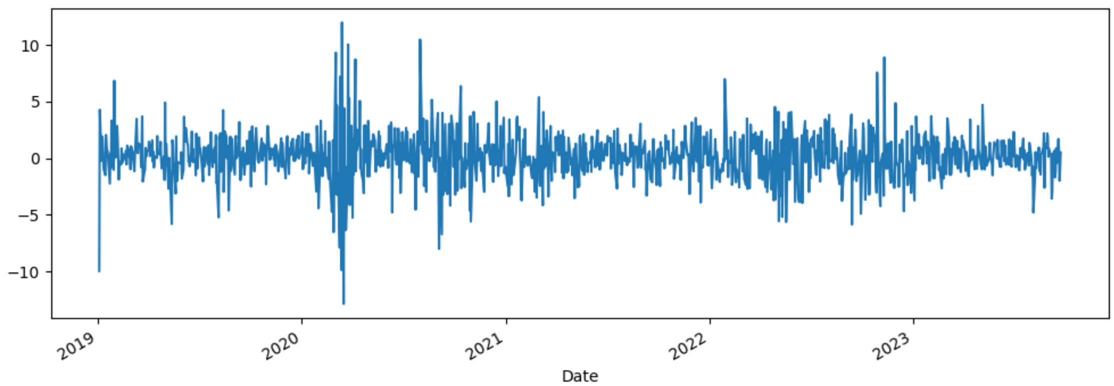
[22]: <matplotlib.legend.Legend at 0x7f37e9ccf2e0>



[23]:
```
#Return Analysis
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 2)
plt.plot(stock_data['Daily Return'], label='Daily Return', color='green')
plt.title('Daily Return Analysis')
plt.ylabel('Daily Return')
plt.legend()
plt.tight_layout()
```



[24]:
```
fig ,ax = plt.subplots(figsize=(12, 4))
stock_data['Daily Return'].plot(ax=ax);
```



2 Time Series Decompositions

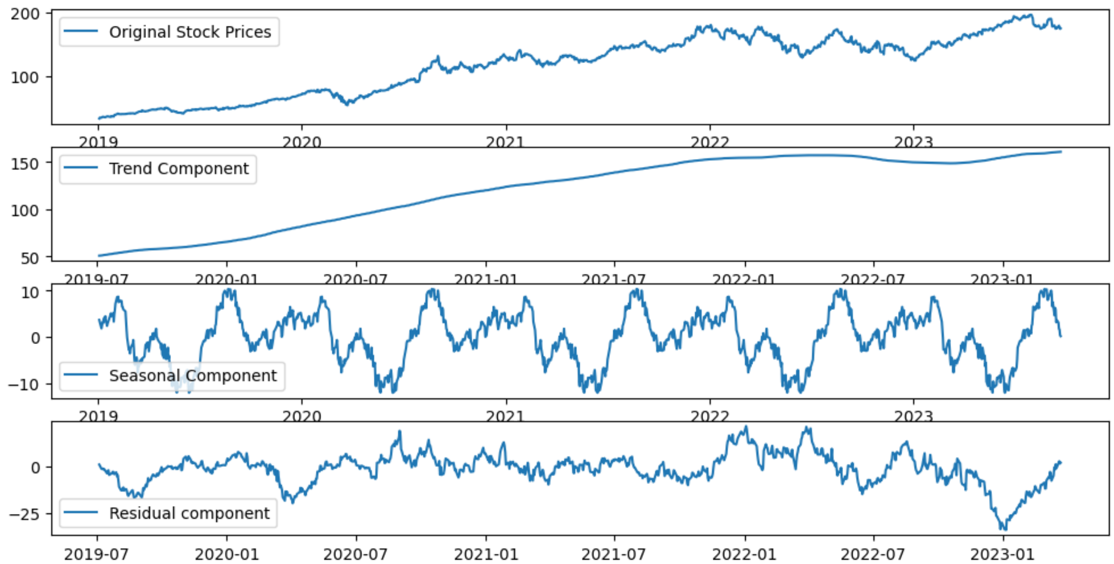
[19]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

[20]:
```
stock_prices = stock_data['Adj Close']
```

[21]:
```
result= seasonal_decompose(stock_prices, model='addictive', period=252)
```

[22]:
```
plt.figure(figsize=(12, 6))

plt.subplot(411)
plt.plot(stock_prices, label='Original Stock Prices')
plt.legend()

plt.subplot(412)
plt.plot(result.trend, label='Trend Component')
plt.legend()
```

```
plt.subplot(413)
plt.plot(result.seasonal, label='Seasonal Component')
plt.legend()

plt.subplot(414)
plt.plot(result.resid, label='Residual component')
plt.legend()

plt.show()
plt.tight_layout()
```




```
<Figure size 640x480 with 0 Axes>
```

3 *Volatility Garch*

[23]:
```
!pip install arch
```

```
Collecting arch
  Downloading
arch-6.1.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (916 kB)
                          916.4/916.4 kB
25.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.19 in
/usr/local/lib/python3.10/dist-packages (from arch) (1.23.5)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.10/dist-
packages (from arch) (1.11.2)
Requirement already satisfied: pandas>=1.1 in /usr/local/lib/python3.10/dist-
```

```
packages (from arch) (1.5.3)
Requirement already satisfied: statsmodels>=0.12 in
/usr/local/lib/python3.10/dist-packages (from arch) (0.14.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.1->arch) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1->arch) (2023.3.post1)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-
packages (from statsmodels>=0.12->arch) (0.5.3)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels>=0.12->arch) (23.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.2->statsmodels>=0.12->arch) (1.16.0)
Installing collected packages: arch
Successfully installed arch-6.1.0
```

[24]:
```
from arch import arch_model
```

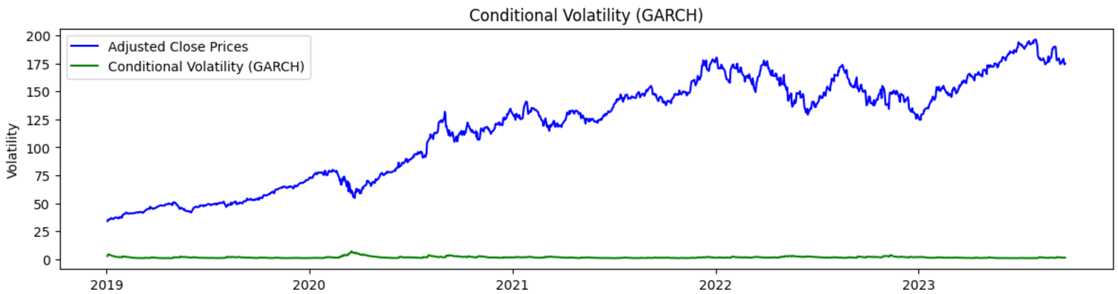
[25]:
```
model = arch_model(stock_data['Daily Return'], vol='Garch', p=1, q=1)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 7094.188659020681
Iteration:      2,   Func. Count:     16,   Neg. LLF: 180592995930.79388
Iteration:      3,   Func. Count:     24,   Neg. LLF: 2800.9432418003344
Iteration:      4,   Func. Count:     31,   Neg. LLF: 2784.186029104576
Iteration:      5,   Func. Count:     38,   Neg. LLF: 2419.5217245842377
Iteration:      6,   Func. Count:     44,   Neg. LLF: 2419.0228627426386
Iteration:      7,   Func. Count:     49,   Neg. LLF: 2419.0220910105472
Iteration:      8,   Func. Count:     54,   Neg. LLF: 2419.0220837354263
Iteration:      9,   Func. Count:     58,   Neg. LLF: 2419.0220837353813
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2419.0220837354263
            Iterations: 9
            Function evaluations: 58
            Gradient evaluations: 9
```

[26]:
```
conditional_volatility = results.conditional_volatility
```

[27]:
```
plt.figure(figsize=(12, 6))


#Plot Stock Price
plt.subplot(2, 1, 2)
plt.plot(stock_prices, label='Adjusted Close Prices', color='Blue')
plt.title('Historical Stock Prices')
plt.ylabel('Prices')
plt.legend()
```

```
#Plot Conditional Volatility
plt.subplot(2, 1, 2)
plt.plot(conditional_volatility, label='Conditional Volatility (GARCH)',␣
↪color='green')
plt.title('Conditional Volatility (GARCH)')
plt.ylabel('Volatility')
plt.legend()

plt.tight_layout()
plt.show()
```



4 *Statistical Descriptions*

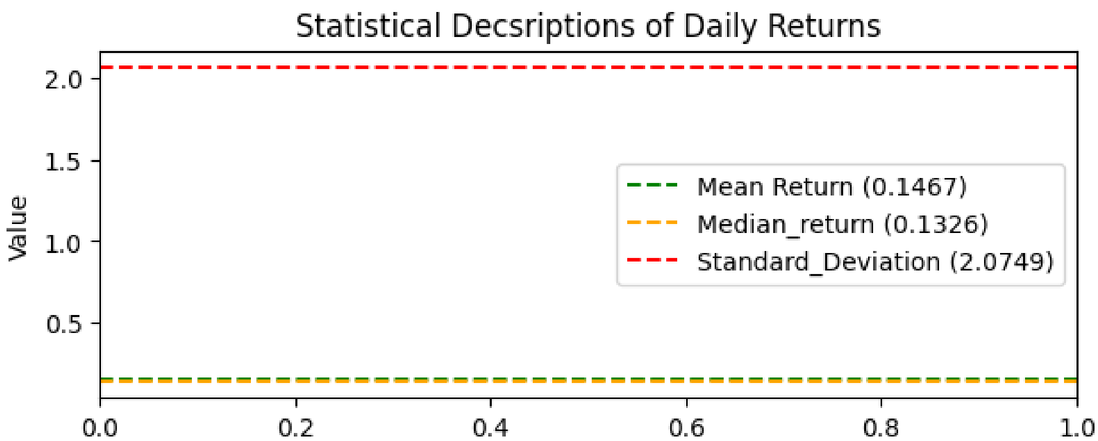
[28]:
```
mean_return = stock_data['Daily Return'].mean()
median_return = stock_data['Daily Return'].median()
std_deviation = stock_data['Daily Return'].std()
skewness = stock_data['Daily Return'].skew()
kurtosis = stock_data['Daily Return'].kurtosis()
```

[29]:
```
mean_return
```

[29]: 0.14673047261990926

[30]:
```
plt.subplot(2, 1, 2)
plt.axhline(mean_return, color='green', linestyle='--', label=f'Mean Return␣
↪({mean_return:.4f})')
plt.axhline(median_return, color='orange', linestyle='--',␣
↪label=f'Median_return ({median_return:.4f})')
plt.axhline(std_deviation, color='red', linestyle='--',␣
↪label=f'Standard_Deviation ({std_deviation:.4f})')
plt.title('Statistical Decsriptions of Daily Returns')
plt.ylabel('Value')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



5 Correlations / cointegrations

[31]:
```
correlation_matrix = stock_data.corr()
```

[32]:
```
correlation_matrix
```

[32]:

	Open	High	Low	Close	Adj Close
Open	1.000000	0.999633	0.999555	0.999061	0.999003
High	0.999633	1.000000	0.999471	0.999552	0.999477
Low	0.999555	0.999471	1.000000	0.999574	0.999546
Close	0.999061	0.999552	0.999574	1.000000	0.999939
Adj Close	0.999003	0.999477	0.999546	0.999939	1.000000
Volume	-0.427227	-0.419438	-0.437098	-0.428742	-0.431130
Daily Return	-0.046858	-0.034352	-0.032134	-0.015809	-0.016420
Log_Return	-0.044253	-0.031982	-0.029379	-0.013254	-0.013865
Cumulative Return	0.999061	0.999552	0.999574	1.000000	0.999939

	Volume	Daily Return	Log_Return	Cumulative Return
Open	-0.427227	-0.046858	-0.044253	0.999061
High	-0.419438	-0.034352	-0.031982	0.999552
Low	-0.437098	-0.032134	-0.029379	0.999574
Close	-0.428742	-0.015809	-0.013254	1.000000
Adj Close	-0.431130	-0.016420	-0.013865	0.999939
Volume	1.000000	-0.034130	-0.048782	-0.428742
Daily Return	-0.034130	1.000000	0.999622	-0.015809
Log_Return	-0.048782	0.999622	1.000000	-0.013254

```
Cumulative Return -0.428742     -0.015809    -0.013254            1.000000
```

[33]:
```python
from statsmodels.tsa.stattools import coint
```

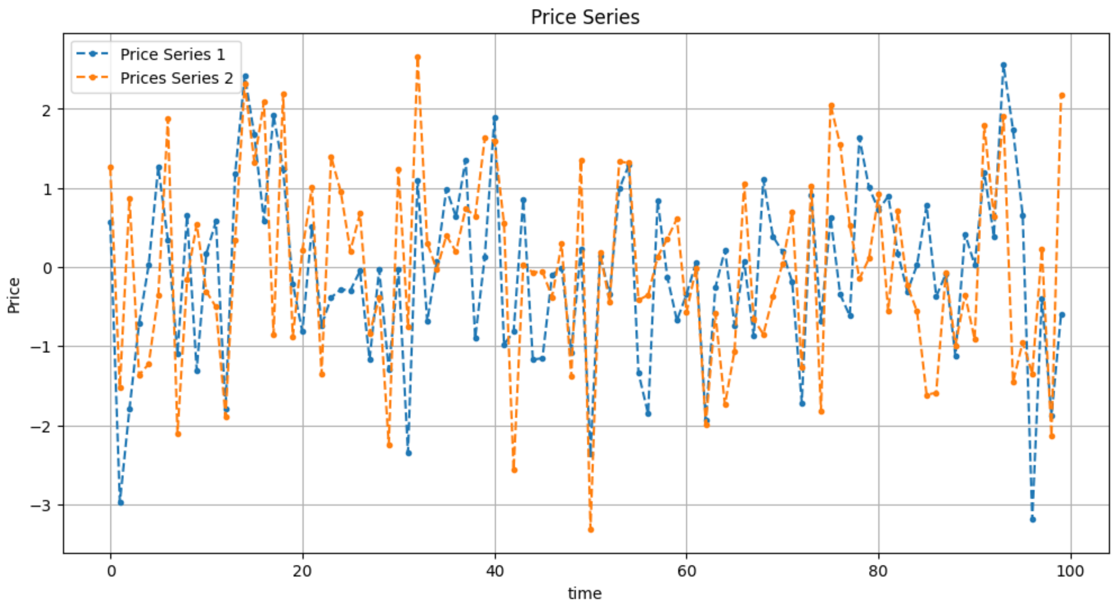
[35]:
```python
price_series_1 = np.random.randn(100)
price_series_2 = 0.5 * price_series_1 + np.random.randn(100)
```

[36]:
```python
cointegration_test = coint(price_series_1, price_series_2)
```

[37]:
```python
cointegration_test
```

[37]:
```
(-8.10242004427434,
 1.6696221220460294e-11,
 array([-4.01048603, -3.39854434, -3.08756793]))
```

[40]:
```python
plt.figure(figsize=(12, 6))
plt.plot(price_series_1, label='Price Series 1', linestyle='--', marker='o',␣
 ↪markersize=3)
plt.plot(price_series_2, label='Prices Series 2', linestyle='--', marker='o',␣
 ↪markersize=3)
plt.title('Price Series')
plt.xlabel('time')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



6 Econometrics Model

[63]:
```
import statsmodels.api as sm
```

[64]:
```
Stock_0 = '^GSPC'
start = '2019-01-01'
end = '2023-09-22'
```

[65]:
```
market_returns = yf.download(Stock_0, start, end)
```

```
[*********************100%%**********************]  1 of 1 completed
```

[66]:
```
market_returns
```

[66]:
```
                   Open         High          Low        Close    Adj Close  \
Date
2019-01-02  2476.959961  2519.489990  2467.469971  2510.030029  2510.030029
2019-01-03  2491.919922  2493.139893  2443.959961  2447.889893  2447.889893
2019-01-04  2474.330078  2538.070068  2474.330078  2531.939941  2531.939941
2019-01-07  2535.610107  2566.159912  2524.560059  2549.689941  2549.689941
2019-01-08  2568.110107  2579.820068  2547.560059  2574.409912  2574.409912
…                   …            …            …            …            …
2023-09-15  4497.979980  4497.979980  4447.209961  4450.319824  4450.319824
2023-09-18  4445.129883  4466.359863  4442.109863  4453.529785  4453.529785
2023-09-19  4445.410156  4449.850098  4416.609863  4443.950195  4443.950195
2023-09-20  4452.810059  4461.029785  4401.379883  4402.200195  4402.200195
2023-09-21  4374.359863  4375.700195  4329.169922  4330.000000  4330.000000

                Volume
Date
2019-01-02  3733160000
2019-01-03  3858830000
2019-01-04  4234140000
2019-01-07  4133120000
2019-01-08  4120060000
…                  …
2023-09-15  6932230000
2023-09-18  3161230000
2023-09-19  3614880000
2023-09-20  3308450000
2023-09-21  3662340000

[1189 rows x 6 columns]
```

[67]:
```
market_data = market_returns['Adj Close'].pct_change().dropna()
Ticker_data  = stock_data['Adj Close'].pct_change().dropna()
```

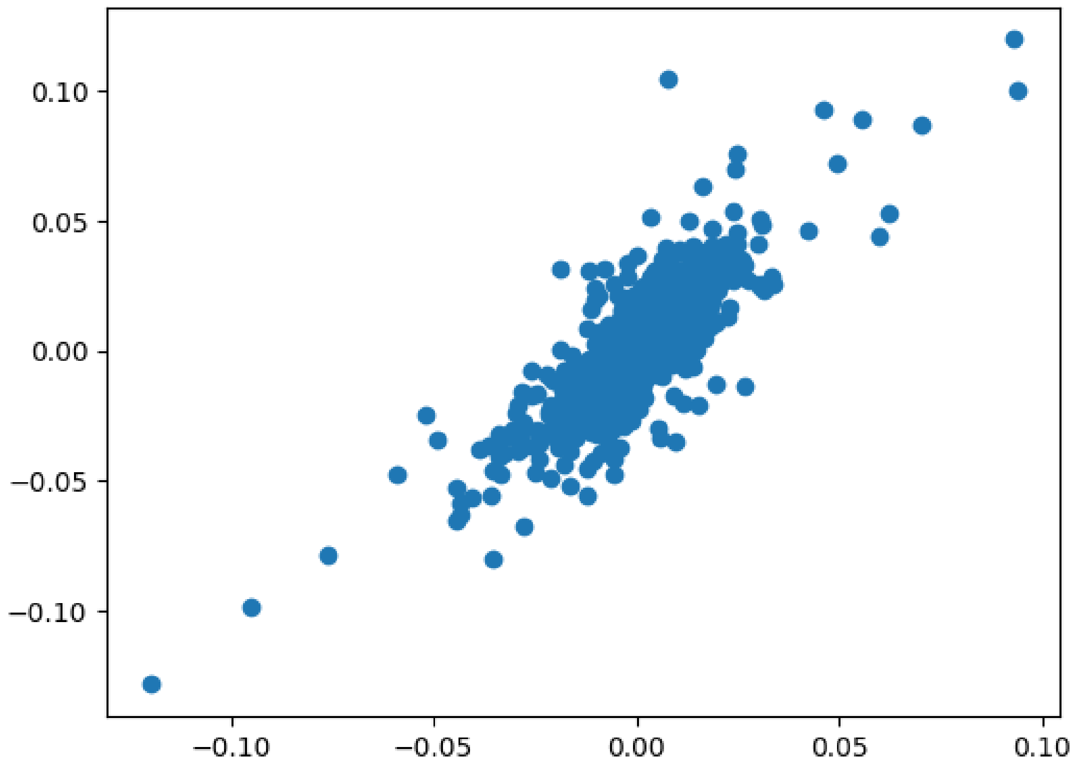
[68]:
```
X = sm.add_constant(market_data)
```

[69]:
```
Ticker_data = Ticker_data.reindex(X.index)
```

[70]:
```
model = sm.OLS(Ticker_data, X).fit()
```

[71]:
```
plt.scatter(market_data, Ticker_data, label='Data')
```

[71]: <matplotlib.collections.PathCollection at 0x7b527f8a0e20>



[79]:
```
prices = stock_data['Adj Close']
```

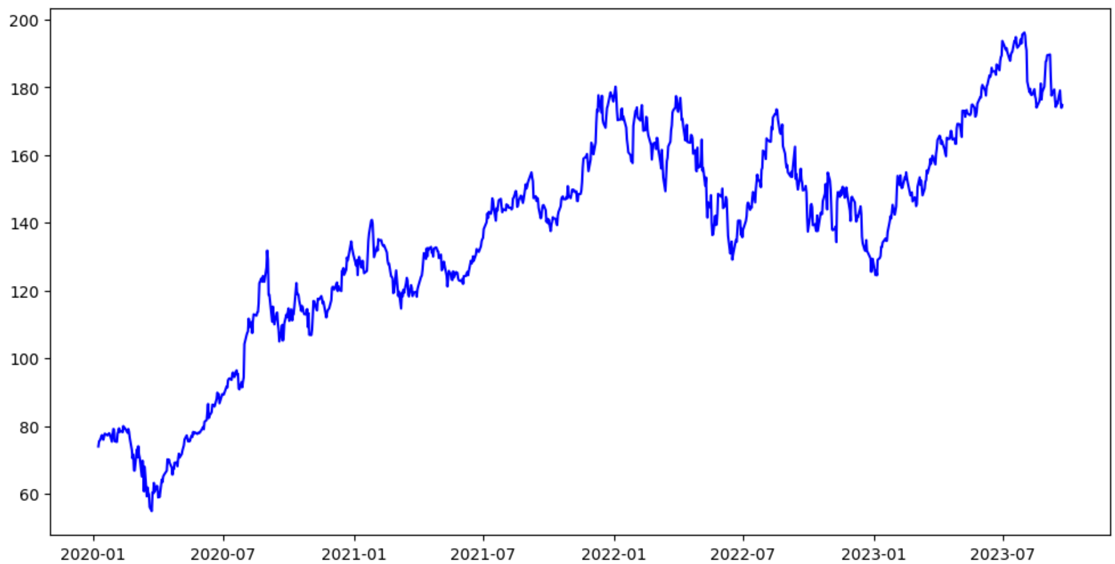
[80]:
```
model = sm.tsa.ARIMA(prices, order=(1, 1, 1)).fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  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 it has no associated frequency
information and so will be ignored when e.g. forecasting.
  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 it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

[81]:
```
plt.figure(figsize=(12,6))
plt.plot(prices, label='Original Price', color='blue')
```

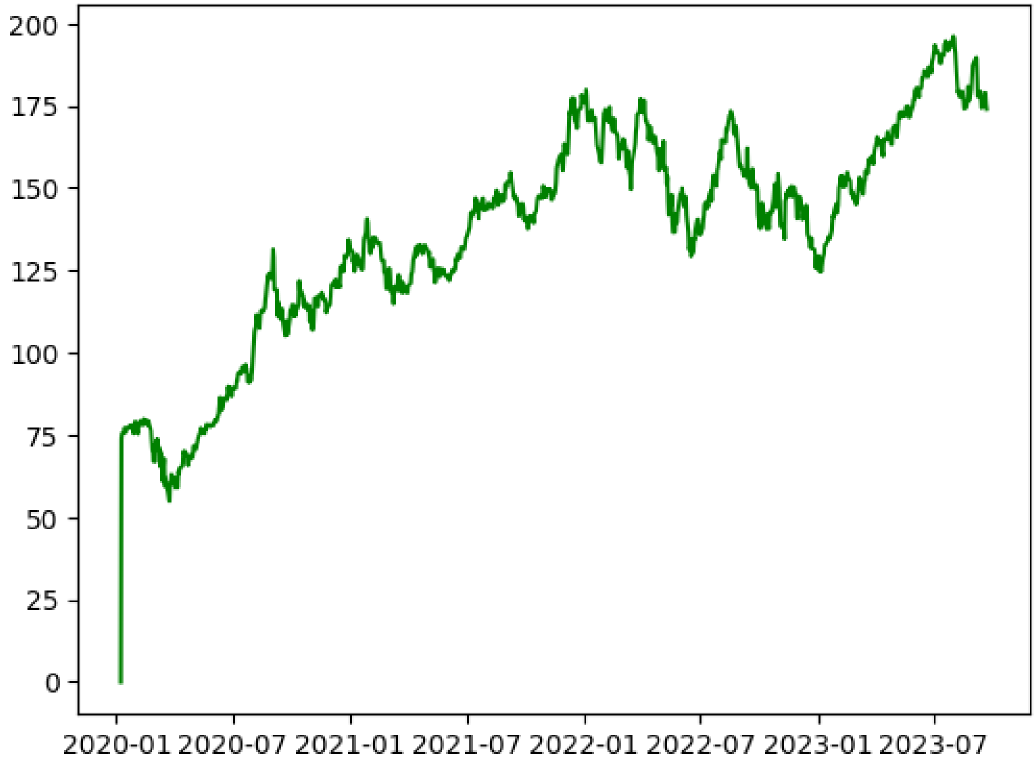
[81]: `[<matplotlib.lines.Line2D at 0x7b527ce0a530>]`



[82]:
```
lags = 5
```

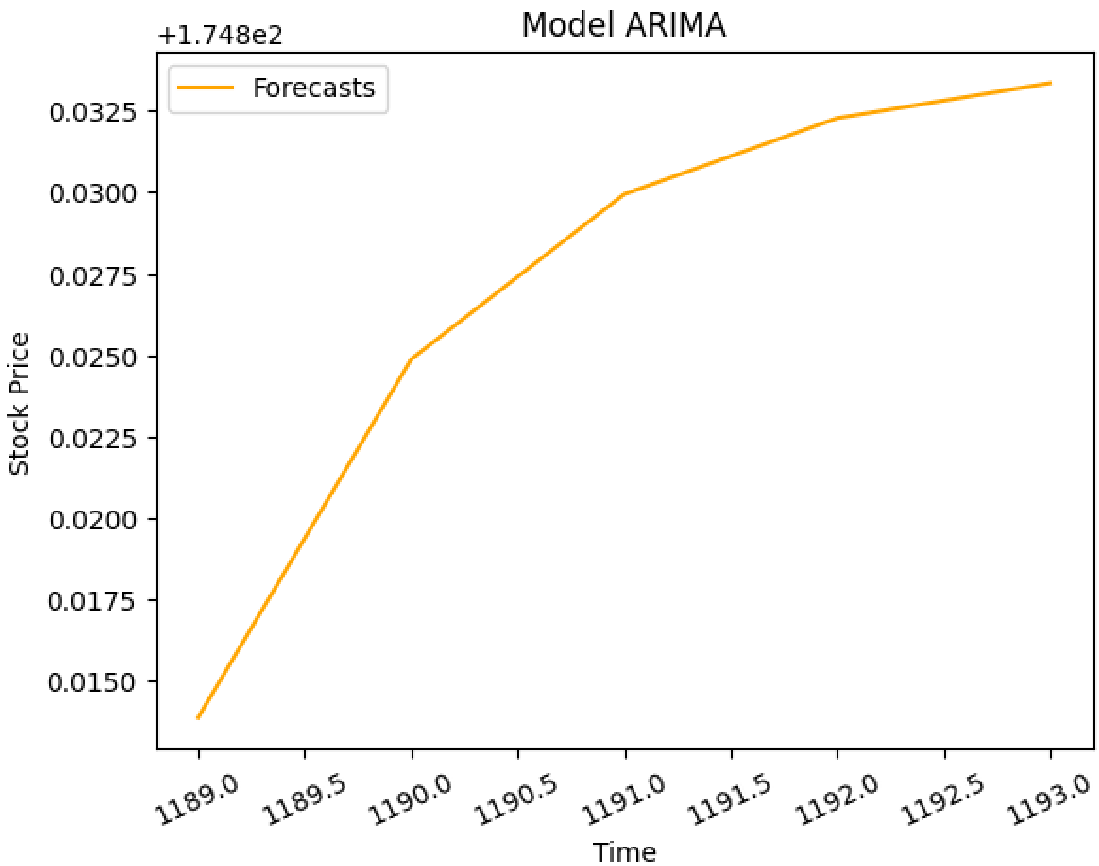
[84]:
```
plt.plot(model.fittedvalues, label='Fitted Value', color='green')
```

[84]: `[<matplotlib.lines.Line2D at 0x7b527cec2470>]`



[101]:
```
forecast = model.forecast(steps=5)
plt.plot(range(len(prices), len(prices) + 5), forecast, label='Forecasts',␣
↪color='orange')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.title('Model ARIMA')
plt.xticks(rotation = 25)
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  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 the next version, calling
this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```



7 Strategies Based on Simple Moving Averages

[11]:
```
data = pd.DataFrame(stock_data)
```

[14]:
```
data
```

[14]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2019-01-02   38.722500   39.712502   38.557499   39.480000   37.943260
2019-01-03   35.994999   36.430000   35.500000   35.547501   34.163837
2019-01-04   36.132500   37.137501   35.950001   37.064999   35.622250
2019-01-07   37.174999   37.207500   36.474998   36.982498   35.542976
2019-01-08   37.389999   37.955002   37.130001   37.687500   36.220524
…                  …          …          …          …          …
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
```

```
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

               Volume
Date
2019-01-02  148158800
2019-01-03  365248800
2019-01-04  234428400
2019-01-07  219111200
2019-01-08  164101200
…                 …
2023-09-18   67257600
2023-09-19   51826900
2023-09-20   58436200
2023-09-21   63047900
2023-09-22   56682928

[1190 rows x 6 columns]
```

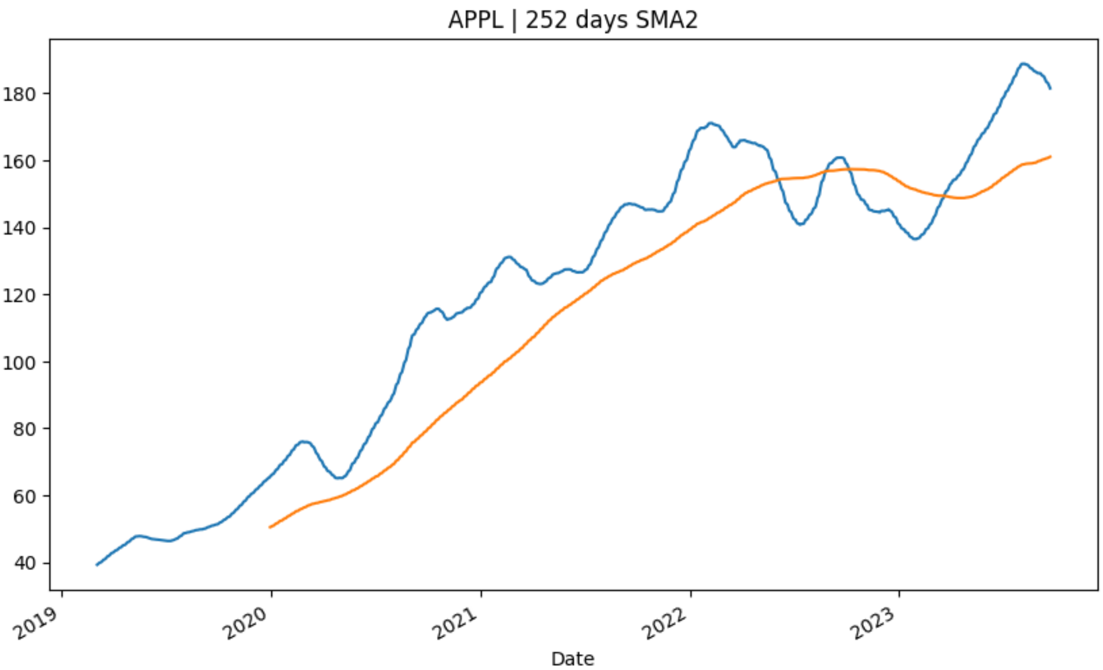
[15]:
```
stock_data['SMA1'] = stock_data['Adj Close'].rolling(42).mean()
stock_data['SMA2'] = stock_data['Adj Close'].rolling(252).mean()
```

[16]:
```
stock_data.tail()
```

[16]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2023-09-18  176.479996  179.380005  176.169998  177.970001  177.970001
2023-09-19  177.520004  179.630005  177.130005  179.070007  179.070007
2023-09-20  179.259995  179.699997  175.399994  175.490005  175.490005
2023-09-21  174.550003  176.300003  173.860001  173.929993  173.929993
2023-09-22  174.669998  177.078995  174.054993  174.789993  174.789993

              Volume        SMA1        SMA2
Date
2023-09-18  67257600  183.063979  160.650982
2023-09-19  51826900  182.735418  160.767088
2023-09-20  58436200  182.349915  160.854076
2023-09-21  63047900  181.908008  160.925327
2023-09-22  56682928  181.465892  161.012536
```

[19]:
```
stock_data['SMA1'].plot(title='APPL |  42 days SMA2', figsize=(10,6));
stock_data['SMA2'].plot (title='APPL | 252 days SMA2', figsize=(10,6))
```

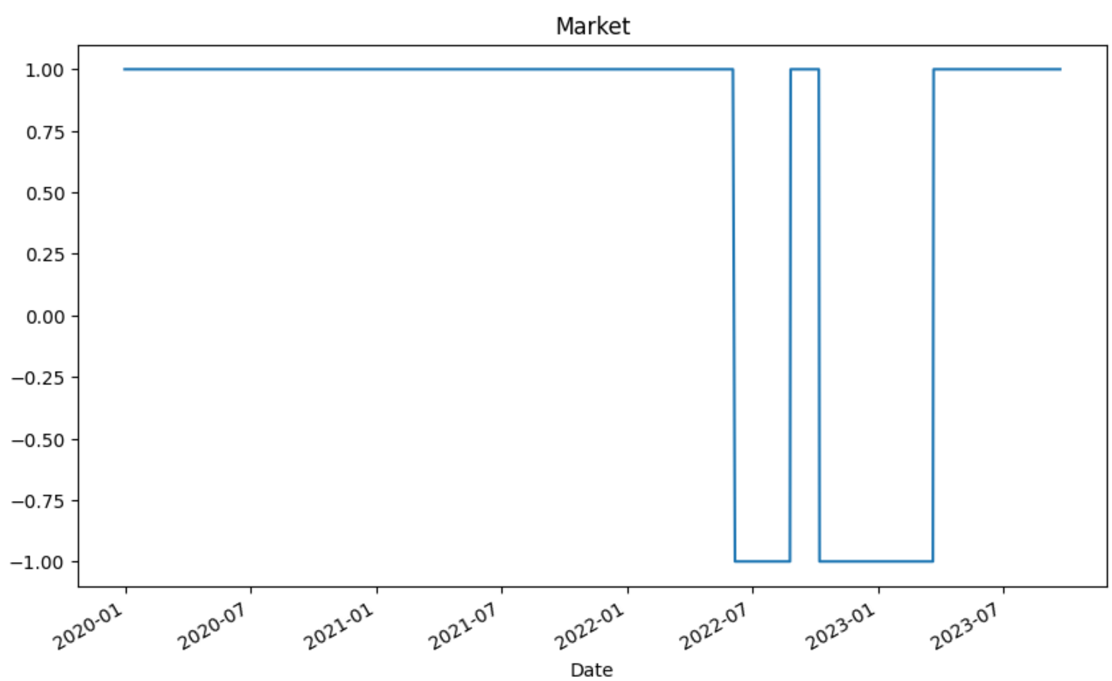
[19]:
```
<Axes: title={'center': 'APPL | 252 days SMA2'}, xlabel='Date'>
```



[20]:
```
stock_data['Position'] = np.where(stock_data['SMA1']> stock_data['SMA2'], 1, -1)
```

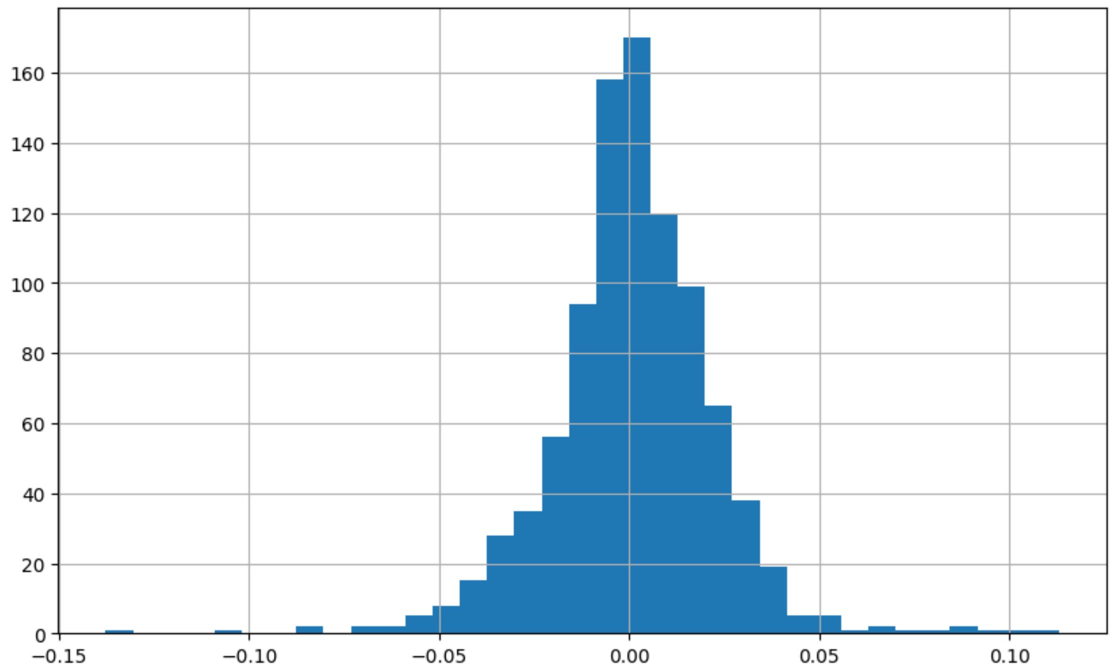
[21]:
```
stock_data.dropna(inplace=True)
```

[23]:
```
stock_data['Position'].plot(ylim=[-1.1, 1.1], title='Market', figsize=(10,6));
```



[25]:
```
stock_data['Return'] = np.log(stock_data['Adj Close'] / stock_data['Adj Close'].
  ↪shift(1))
```

[26]:
```
stock_data['Return'].hist(bins=35, figsize=(10, 6));
```



[27]:
```
stock_data['Strategy'] = stock_data['Position'].shift(1) * stock_data['Return']
```

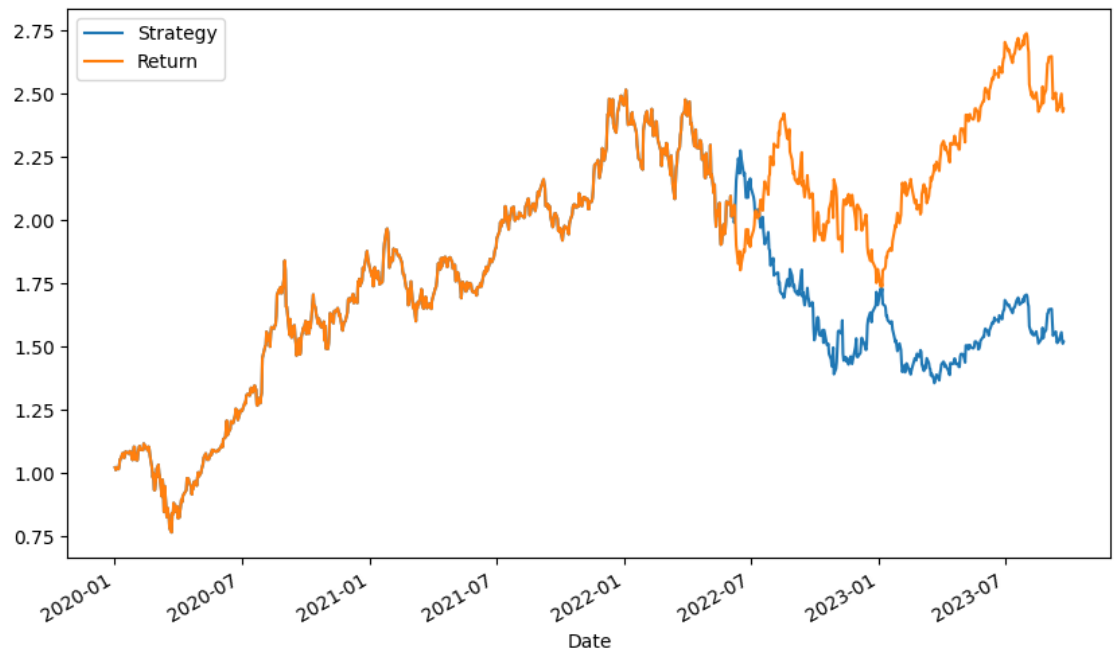
[29]:
```
stock_data[['Strategy', 'Return']].sum()
```

[29]:
```
Strategy    0.418708
Return      0.892280
dtype: float64
```

[31]:
```
stock_data[['Strategy', 'Return']].sum().apply(np.exp)
```

[31]:
```
Strategy    1.519997
Return      2.440689
dtype: float64
```

[32]:
```
stock_data[['Strategy', 'Return']].cumsum().apply(np.exp).plot(figsize=(10, 6));
```



[33]:
```
stock_data[['Strategy', 'Return']].mean() * 252
```

[33]:
```
Strategy    0.112489
Return      0.239717
dtype: float64
```

[34]:
```
np.exp(stock_data[['Strategy', 'Return']].mean() * 252) - 1
```

[34]:
```
Strategy    0.11906
Return      0.27089
dtype: float64
```

[35]:
```
stock_data[['Strategy','Return']].std() * 252 ** 0.5
```

[35]:
```
Strategy    0.345133
Return      0.344875
dtype: float64
```

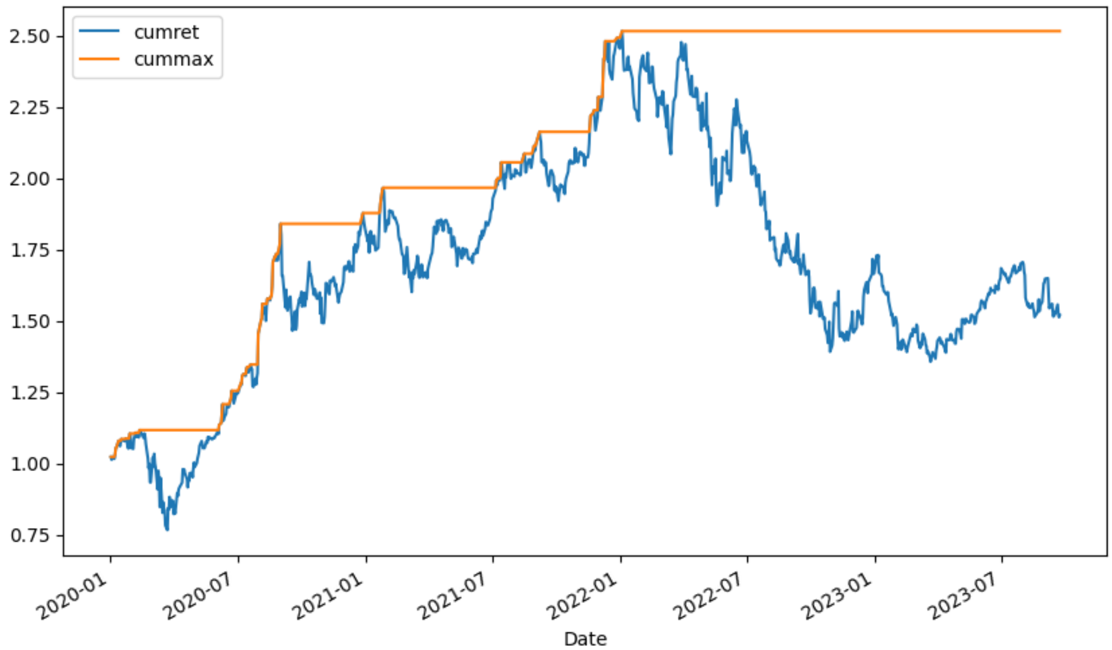
[36]:
```
(stock_data[['Strategy','Return']].apply(np.exp) - 1).std() * 252 ** 0.5
```

[36]:
```
Strategy    0.344744
Return      0.345029
dtype: float64
```

[37]:
```
stock_data['cumret'] = stock_data['Strategy'].cumsum().apply(np.exp)
```

[38]:
```
stock_data['cummax'] = stock_data['cumret'].cummax()
```

[39]:
```
stock_data[['cumret', 'cummax']].dropna().plot(figsize=(10, 6));
```



[40]:
```
drawdown = stock_data['cummax'] - stock_data['cumret']
```

[41]:
```
drawdown.max()
```

[41]: 1.1597999478192935

8 Predictions Stock

[47]:
```
X= stock_data['Close']
```

[49]:
```
y =  X + np.random.standard_normal(len(X))
```

[50]:
```
reg = np.polyfit(X, y, deg=1)
```

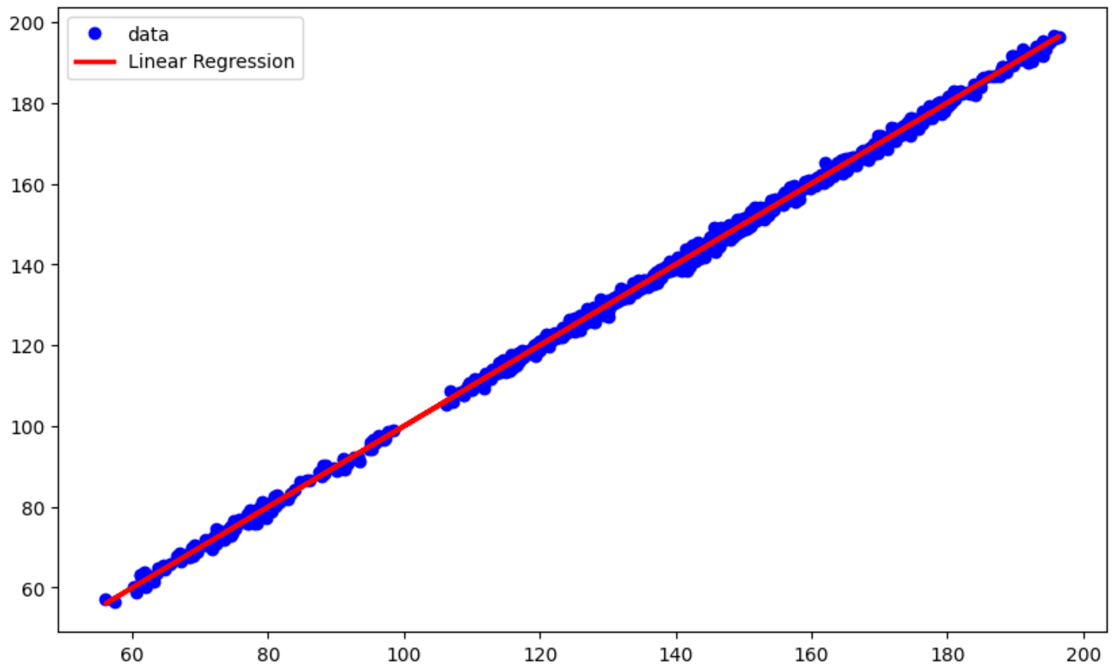
[51]:
```
reg
```

[51]: array([1.00090991, -0.1240214])

[52]:
```
plt.figure(figsize=(10, 6))
plt.plot(X, y, 'bo', label='data')
```

```
plt.plot(X, np.polyval(reg, X), 'r', lw=2.5, label='Linear Regression')
plt.legend(loc=0)
```

[52]: <matplotlib.legend.Legend at 0x7b528c7d7dc0>



[53]:
```
lags = 5
```

[54]:
```
cols = []
for lag in range(1, lags + 1):
    col =f'lag_{lag}'
    stock_data[col] = stock_data['Adj Close'].shift(lag)
    cols.append(col)
stock_data.dropna(inplace=True)
```

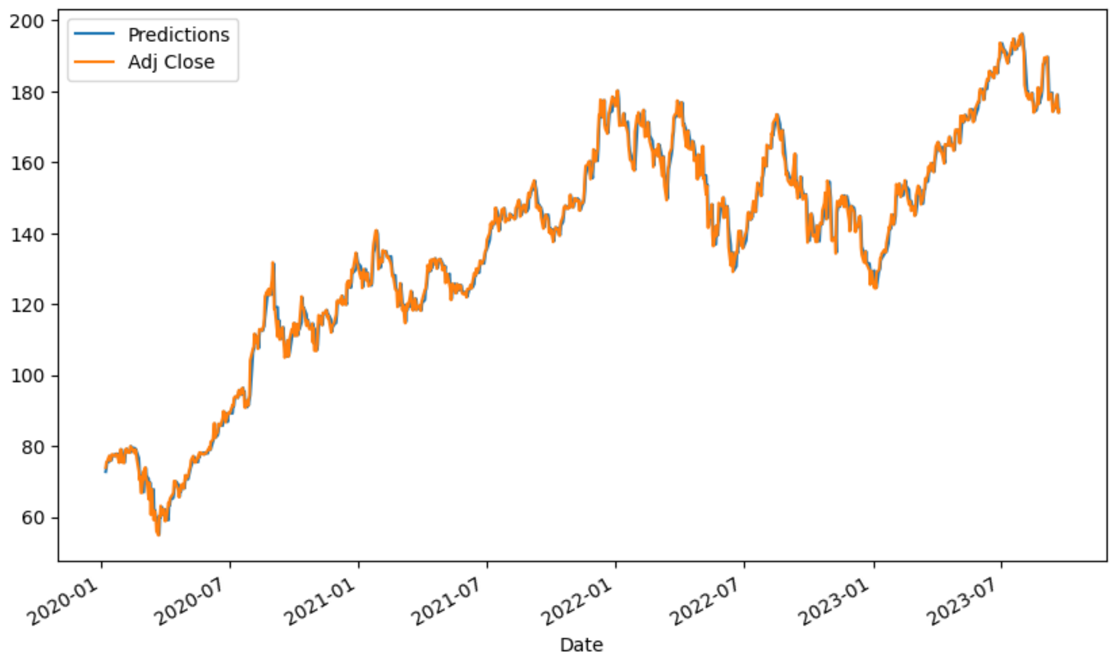
[55]:
```
reg = np.linalg.lstsq(stock_data[cols], stock_data['Adj Close'], rcond=None)[0]
```

[56]:
```
reg
```

[56]: array([0.95667901, 0.01357397, 0.00697459, 0.00724528, 0.01613076])

[57]:
```
stock_data['Predictions'] = np.dot(stock_data[cols], reg)
```

[58]:
```
stock_data[['Predictions', 'Adj Close']].plot(figsize=(10, 6));
```



[59]:
```
stock_data[['Predictions', 'Adj Close']].loc['2023-1-1':].plot(figsize=(10, 6));
```