In [4]:
```
import pandas as pd
import yfinance as yf
from datetime import date,timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import date,timedelta
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from pmdarima import auto_arima
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.metrics import mean_squared_error
```

In [5]:
```
df = yf.download('PFE',
                     start= date.today()- timedelta(days = 365),
                     end=date.today()+ timedelta(days = 1),
                     progress=False)
df.tail()
```

Out[5]:

Date	Open	High	Low	Close	Adj Close	Volume
2021-11-02	45.080002	46.040001	43.049999	45.450001	45.054520	69248000
2021-11-03	45.520000	45.990002	44.480000	44.820000	44.430000	44593100
2021-11-04	44.290001	44.470001	43.310001	43.849998	43.849998	38172500
2021-11-05	48.090000	48.810001	46.549999	48.610001	48.610001	173753300
2021-11-08	48.610001	48.790001	47.599998	48.330002	48.330002	57370300

In [6]:
```
#Using tail command to see the sample of data from bottom
df.tail()
```

Out[6]:

Date	Open	High	Low	Close	Adj Close	Volume

Date	Open	High	Low	Close	Adj Close	Volume
2021-11-02	45.080002	46.040001	43.049999	45.450001	45.054520	69248000
2021-11-03	45.520000	45.990002	44.480000	44.820000	44.430000	44593100
2021-11-04	44.290001	44.470001	43.310001	43.849998	43.849998	38172500
2021-11-05	48.090000	48.810001	46.549999	48.610001	48.610001	173753300
2021-11-08	48.610001	48.790001	47.599998	48.330002	48.330002	57370300

In [7]:
```
#Using shape command to find the number of rows and columns
df.shape
```

Out[7]:
```
(252, 6)
```

In [8]:
```
#Using info command to find the datatypes of columns,index range,not null count
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2020-11-09 to 2021-11-08
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       252 non-null    float64
 1   High       252 non-null    float64
 2   Low        252 non-null    float64
 3   Close      252 non-null    float64
 4   Adj Close  252 non-null    float64
 5   Volume     252 non-null    int64  
dtypes: float64(5), int64(1)
memory usage: 13.8 KB
```

In [9]:
```
#Removing the columns which are not used for the futher operations
df=df.drop(['Adj Close'], axis= 1)
```

In [10]:
```
#Daily returns tells us the returns that we obtain in a day after the stock price closes
#The daily return measures the dollar change in a stock's price as a percentage of the previous day's closing price.
```

```
#A positive return means the stock has grown in value, while a negative return means it has lost value.
df['Daily Returns'] = (df['Close']/df['Close'].shift(1)) -1
df.head()
```

Out[10]:

Date	Open	High	Low	Close	Volume	Daily Returns
2020-11-09	39.715370	39.838711	36.413662	37.191650	230153864	NaN
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495

In [11]:

```
df.tail()
```

Out[11]:

Date	Open	High	Low	Close	Volume	Daily Returns
2021-11-02	45.080002	46.040001	43.049999	45.450001	69248000	0.041476
2021-11-03	45.520000	45.990002	44.480000	44.820000	44593100	-0.013861
2021-11-04	44.290001	44.470001	43.310001	43.849998	38172500	-0.021642
2021-11-05	48.090000	48.810001	46.549999	48.610001	173753300	0.108552
2021-11-08	48.610001	48.790001	47.599998	48.330002	57370300	-0.005760

In [12]:

```
df['Daily_Returns_PCT']=df['Close'].pct_change(1)
df.head()
```

Out[12]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_Returns_PCT
2020-11-09	39.715370	39.838711	36.413662	37.191650	230153864	NaN	NaN

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_Returns_PCT
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265	-0.013265
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654	-0.004654
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675	-0.024675
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495	0.028495

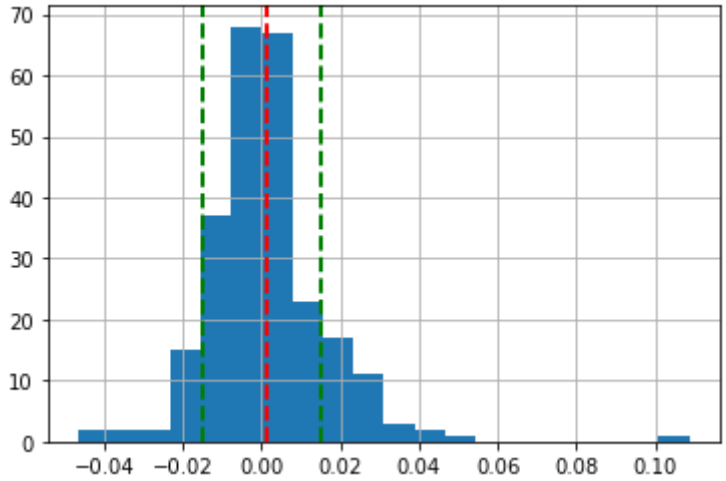
In [13]:
```
#Removing the columns which are not used for the futher operations
df=df.drop(['Daily_Returns_PCT'], axis= 1)
```

In [18]:
```
mean = df['Daily Returns'].mean()
std = df['Daily Returns'].std()
print('mean =',mean)
print('Std deviation =',std)
```

```
mean = 0.0011575092203006008
Std deviation = 0.015200594182836086
```

In [19]:
```
df['Daily Returns'].hist(bins=20)
plt.axvline(mean,color='red',linestyle='dashed',linewidth=2)
#to plot the std line we plot both the positive and negative values
plt.axvline(std,color='g',linestyle='dashed',linewidth=2)
plt.axvline(-std,color='g',linestyle='dashed',linewidth=2)
```

Out[19]:
```
<matplotlib.lines.Line2D at 0x147ebdff340>
```



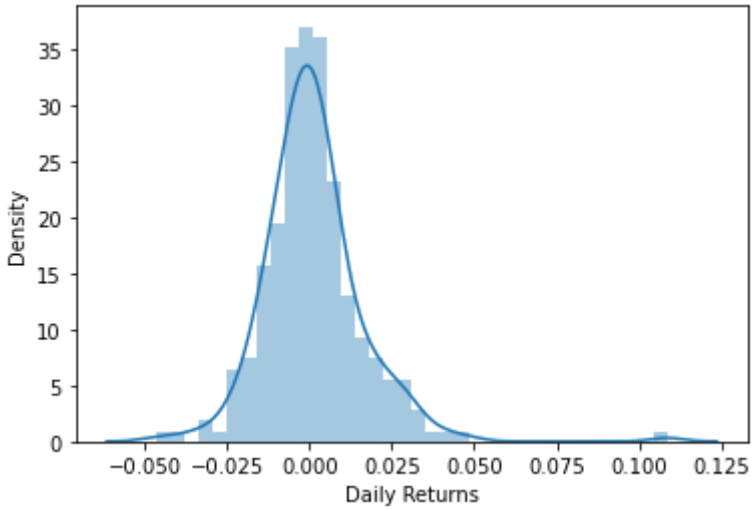
In [20]:
```
#Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.
#That is, data sets with high kurtosis tend to have heavy tails, or outliers.
#Data sets with low kurtosis tend to have light tails, or lack of outliers.
#A uniform distribution would be the extreme case.
#here we have relatively low kurtosis

df['Daily Returns'].kurtosis()
```

Out[20]: 9.913877978038844

In [21]:
```
sns.distplot(df['Daily Returns'])
```

```
C:\Users\amalj\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[21]: <AxesSubplot:xlabel='Daily Returns', ylabel='Density'>



In [22]:
```
#Log returns are time additive. Most technical analysis requires normalizing the time series.Log is the good way.
#Log return has slight change from Daily return
df['Daily_return_log']=np.log(df['Close']/df['Close'].shift(1))
df.head()
```

Out[22]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log
2020-11-09	39.715370	39.838711	36.413662	37.191650	230153864	NaN	NaN
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265	-0.013354
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654	-0.004664
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675	-0.024985
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495	0.028097

In [23]:
```
df['Cumilative return']=np.cumsum(df['Daily Returns'])
df.head()
```

Out[23]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return
2020-11-09	39.715370	39.838711	36.413662	37.191650	230153864	NaN	NaN	NaN
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265	-0.013354	-0.013265
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654	-0.004664	-0.017919
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675	-0.024985	-0.042594
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495	0.028097	-0.014099

In [24]:
```
df.tail()
```

Out[24]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return
2021-11-02	45.080002	46.040001	43.049999	45.450001	69248000	0.041476	0.040639	0.223247
2021-11-03	45.520000	45.990002	44.480000	44.820000	44593100	-0.013861	-0.013958	0.209385
2021-11-04	44.290001	44.470001	43.310001	43.849998	38172500	-0.021642	-0.021880	0.187743
2021-11-05	48.090000	48.810001	46.549999	48.610001	173753300	0.108552	0.103055	0.296295
2021-11-08	48.610001	48.790001	47.599998	48.330002	57370300	-0.005760	-0.005777	0.290535

In [25]:
```
#The compound return is the rate of return, usually expressed as a percentage, that represents the cumulative effect that
#a series of gains or losses has on an original amount of capital over a period of time.

df['Cumulative Compounded Return']= (1+ df['Daily Returns']).cumprod()
df.head()
```

Out[25]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
2020-11-09	39.715370	39.838711	36.413662	37.191650	230153864	NaN	NaN	NaN	NaN
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265	-0.013354	-0.013265	0.986735

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654	-0.004664	-0.017919	0.982143
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675	-0.024985	-0.042594	0.957908
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495	0.028097	-0.014099	0.985204

In [26]:
```
df.tail()
```

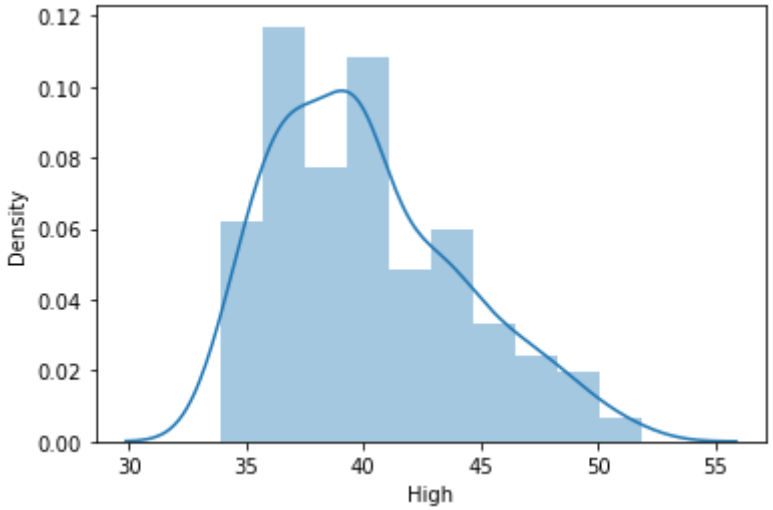
Out[26]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
2021-11-02	45.080002	46.040001	43.049999	45.450001	69248000	0.041476	0.040639	0.223247	1.222049
2021-11-03	45.520000	45.990002	44.480000	44.820000	44593100	-0.013861	-0.013958	0.209385	1.205109
2021-11-04	44.290001	44.470001	43.310001	43.849998	38172500	-0.021642	-0.021880	0.187743	1.179028
2021-11-05	48.090000	48.810001	46.549999	48.610001	173753300	0.108552	0.103055	0.296295	1.307014
2021-11-08	48.610001	48.790001	47.599998	48.330002	57370300	-0.005760	-0.005777	0.290535	1.299485

In [27]:
```
'''''''''Today's high refers to a security's intraday highest trading price.
It is represented by the highest point on a day's stock chart.
This can be contrasted with today's low, which is the trading day's intraday low price.'''''''''
sns.distplot(df['High'])
```

```
C:\Users\amalj\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

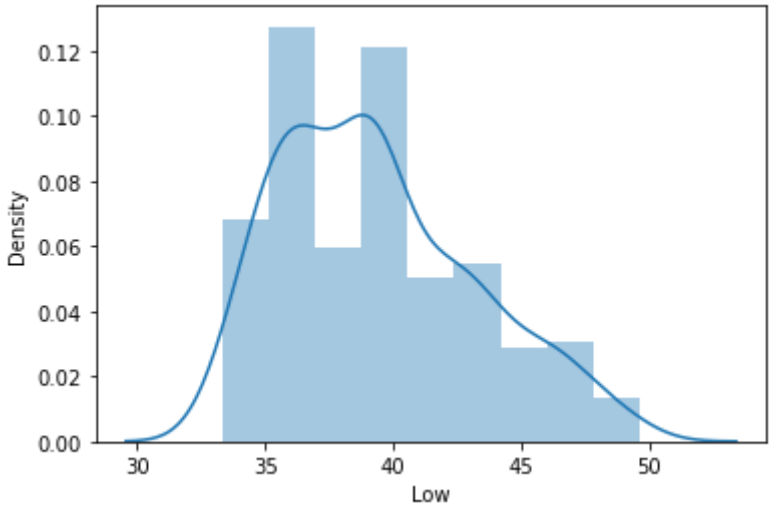
Out[27]: <AxesSubplot:xlabel='High', ylabel='Density'>



In [28]:
```
'''''''''Today's high refers to a security's intraday highest trading price.
It is represented by the highest point on a day's stock chart.
This can be contrasted with today's low, which is the trading day's intraday low price.'''''''''
sns.distplot(df['Low'])
```

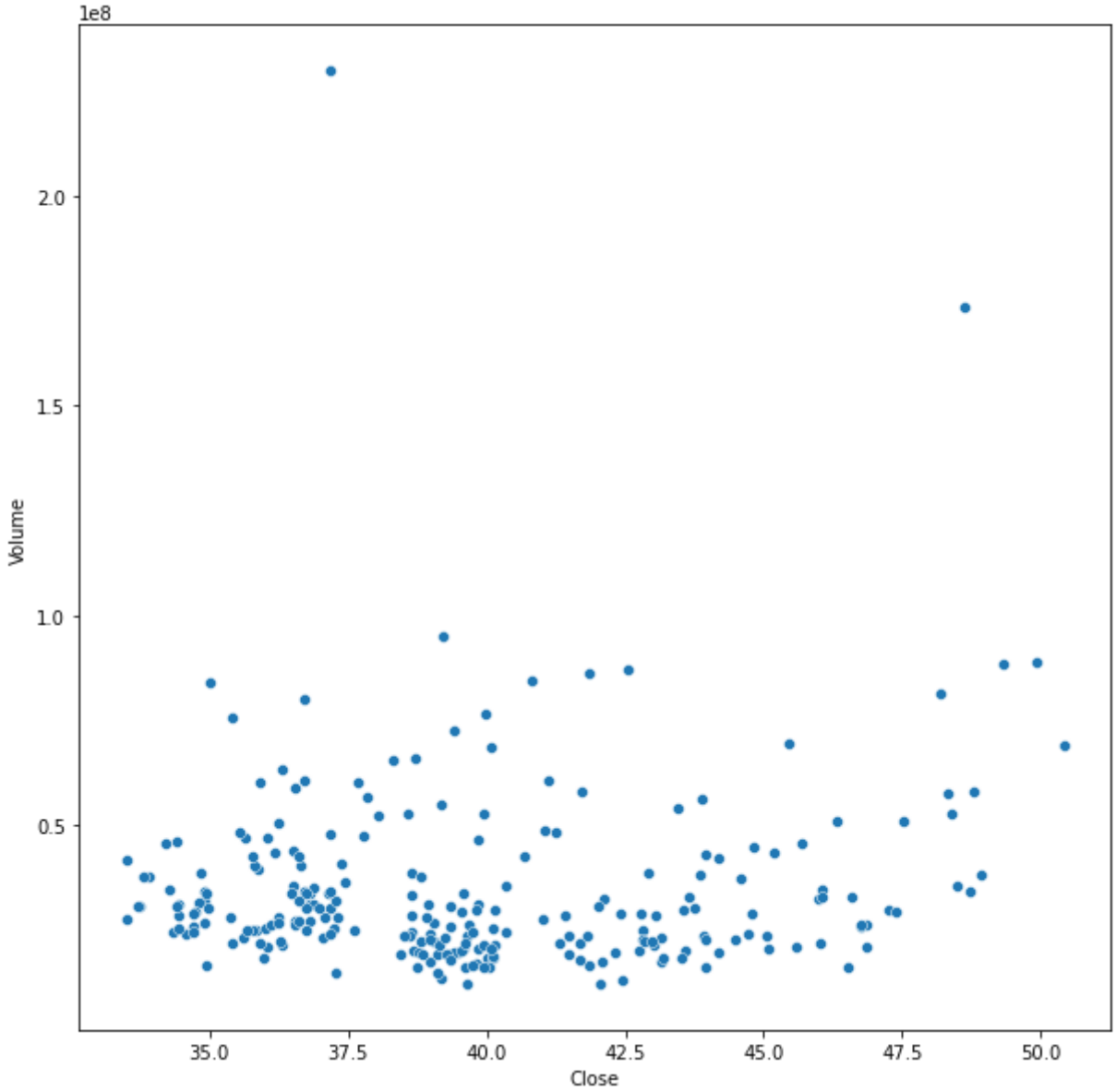
```
C:\Users\amalj\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[28]: <AxesSubplot:xlabel='Low', ylabel='Density'>



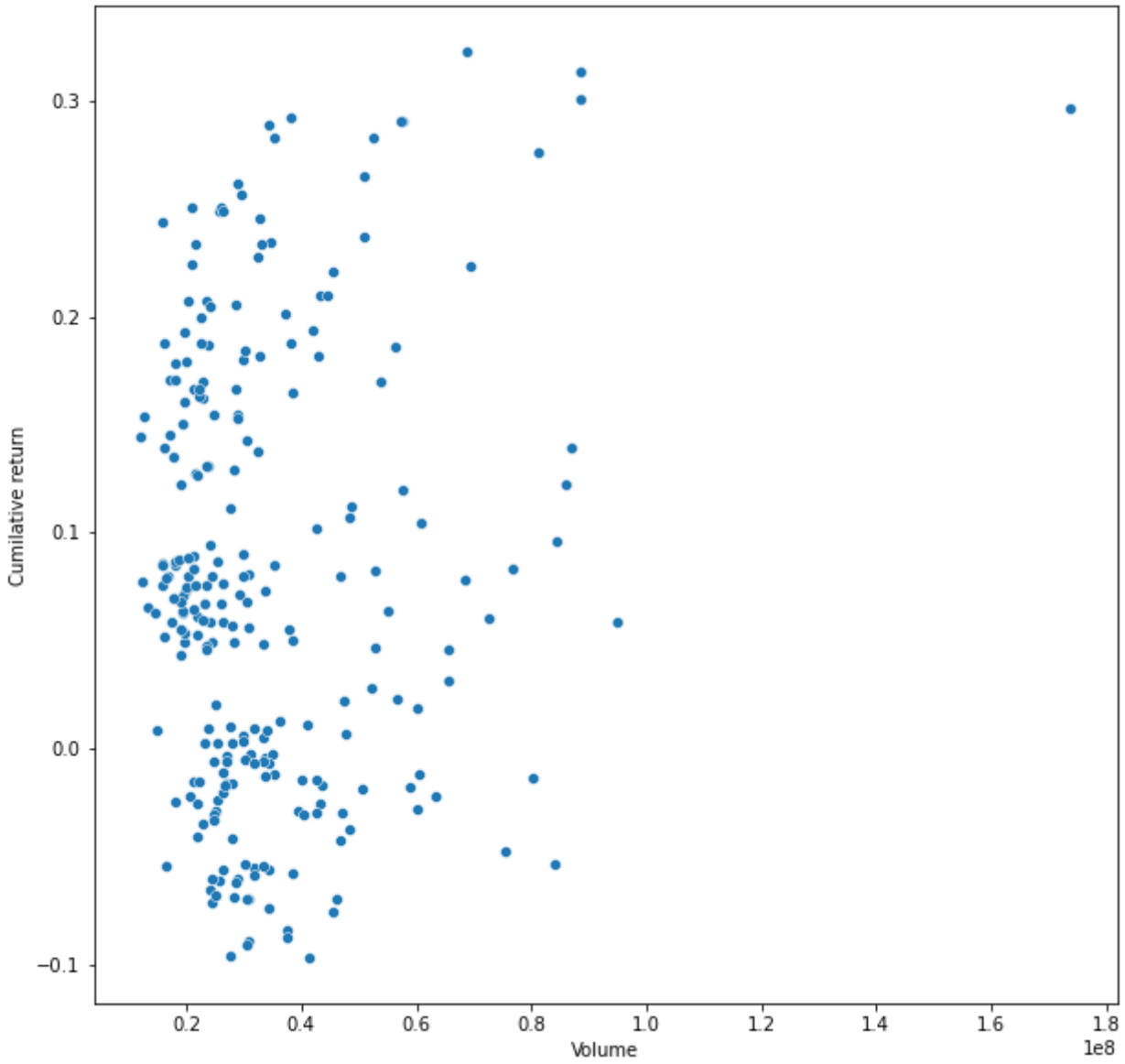
In [29]:
```
#No signifact relationship between volume and close price
plt.figure(figsize=(10,10))
sns.scatterplot(x=df['Close'],y=df['Volume'])
```

Out[29]: <AxesSubplot:xlabel='Close', ylabel='Volume'>



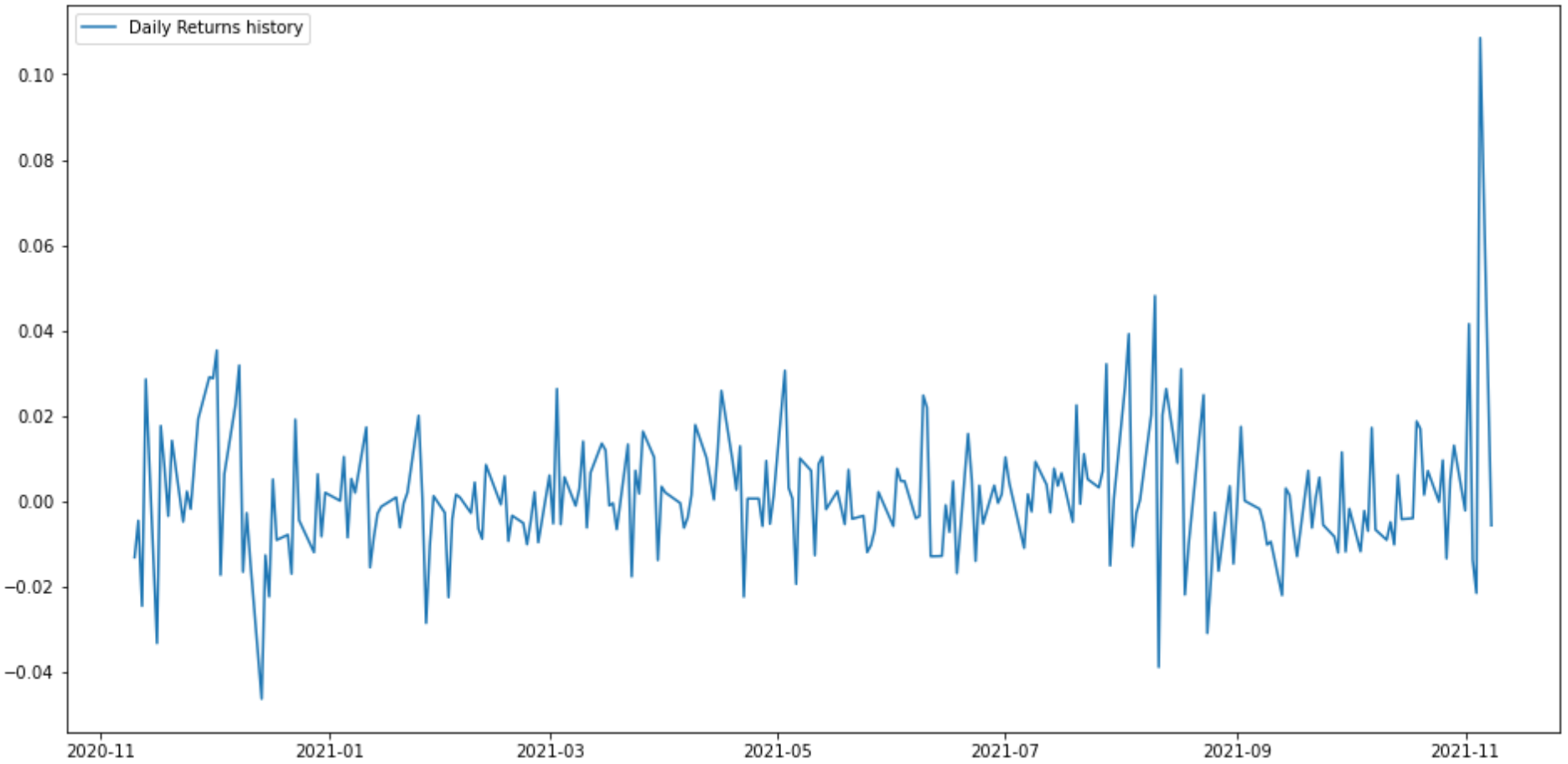
In [30]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x=df['Volume'],y=df['Cumilative return'])
```

Out[30]: <AxesSubplot:xlabel='Volume', ylabel='Cumilative return'>



In [31]:
```
plt.figure(figsize=(16,8))
plt.plot(df["Daily Returns"],label='Daily Returns history')
plt.legend(loc='upper left', fontsize=10)
```

Out[31]: <matplotlib.legend.Legend at 0x147ebe2ba30>



In [32]:
```
plt.figure(figsize=(16,8))
plt.plot(df["Close"],label='Close Price history')
plt.legend(loc='upper left', fontsize=10)
```

Out[32]: <matplotlib.legend.Legend at 0x147f2224ca0>

In [34]:
```
df.shape
```

Out[34]: (252, 9)

In [35]:
```
#Using dropna() function drop rows with at least one Null value
df.dropna()
```

Out[35]:

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return

Date	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
2020-11-10	38.377609	38.462997	36.489563	36.698292	80091668	-0.013265	-0.013354	-0.013265	0.986735
2020-11-11	36.888046	38.140415	35.958256	36.527515	58980997	-0.004654	-0.004664	-0.017919	0.982143
2020-11-12	36.318787	36.375713	35.332069	35.626186	46767877	-0.024675	-0.024985	-0.042594	0.957908
2020-11-13	35.929790	36.679317	35.777988	36.641365	40175634	0.028495	0.028097	-0.014099	0.985204
2020-11-16	35.920303	36.005692	34.962048	35.417458	75530167	-0.033402	-0.033973	-0.047501	0.952296
...	...	...	...	...	...	...	...	...	...
2021-11-02	45.080002	46.040001	43.049999	45.450001	69248000	0.041476	0.040639	0.223247	1.222049
2021-11-03	45.520000	45.990002	44.480000	44.820000	44593100	-0.013861	-0.013958	0.209385	1.205109
2021-11-04	44.290001	44.470001	43.310001	43.849998	38172500	-0.021642	-0.021880	0.187743	1.179028
2021-11-05	48.090000	48.810001	46.549999	48.610001	173753300	0.108552	0.103055	0.296295	1.307014
2021-11-08	48.610001	48.790001	47.599998	48.330002	57370300	-0.005760	-0.005777	0.290535	1.299485

251 rows × 9 columns

In [36]:
```
df.corr()
```

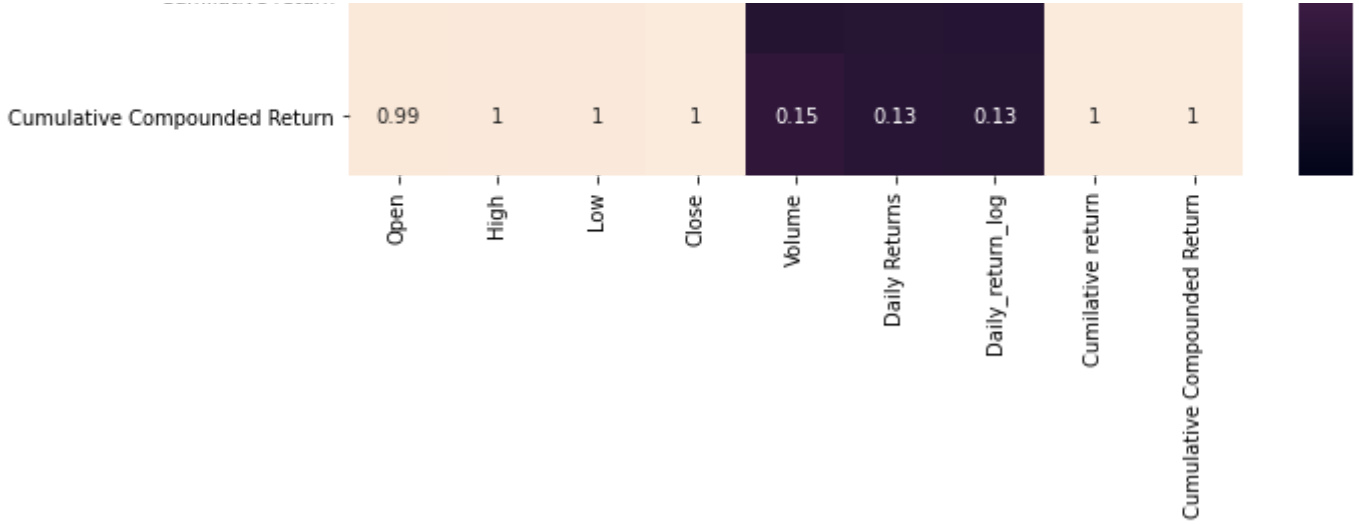
Out[36]:

	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
Open	1.000000	0.996406	0.993917	0.990913	0.136708	0.034342	0.029137	0.989965	0.991713
High	0.996406	1.000000	0.993838	0.995235	0.156724	0.086231	0.081223	0.992473	0.995887
Low	0.993917	0.993838	1.000000	0.996073	0.066395	0.077649	0.073629	0.995235	0.996095
Close	0.990913	0.995235	0.996073	1.000000	0.104743	0.133997	0.129660	0.998007	1.000000
Volume	0.136708	0.156724	0.066395	0.104743	1.000000	0.218806	0.204823	0.121188	0.152763
Daily Returns	0.034342	0.086231	0.077649	0.133997	0.218806	1.000000	0.999755	0.128168	0.133997
Daily_return_log	0.029137	0.081223	0.073629	0.129660	0.204823	0.999755	1.000000	0.124057	0.129660

	Open	High	Low	Close	Volume	Daily Returns	Daily_return_log	Cumilative return	Cumulative Compounded Return
Cumilative return	0.989965	0.992473	0.995235	0.998007	0.121188	0.128168	0.124057	1.000000	0.998007
Cumulative Compounded Return	0.991713	0.995887	0.996095	1.000000	0.152763	0.133997	0.129660	0.998007	1.000000

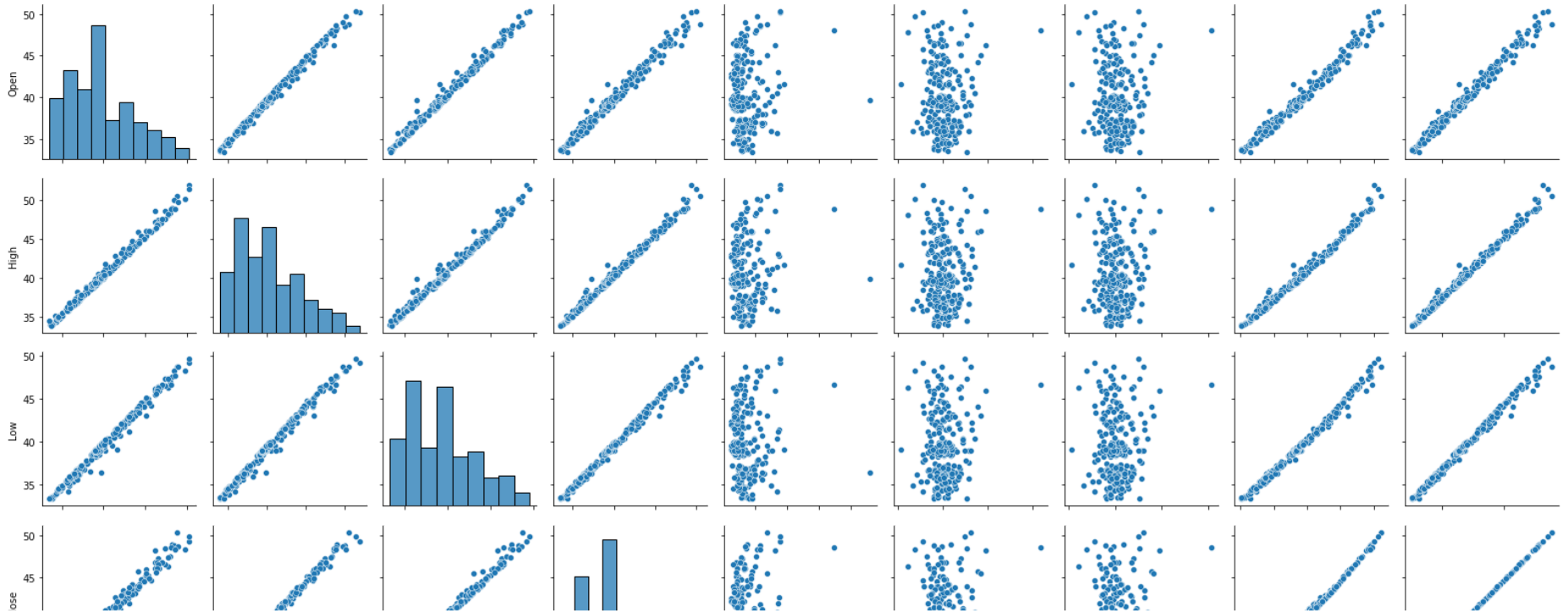
In [37]:
```
#Heatmap of correlation Matrix
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True)
```

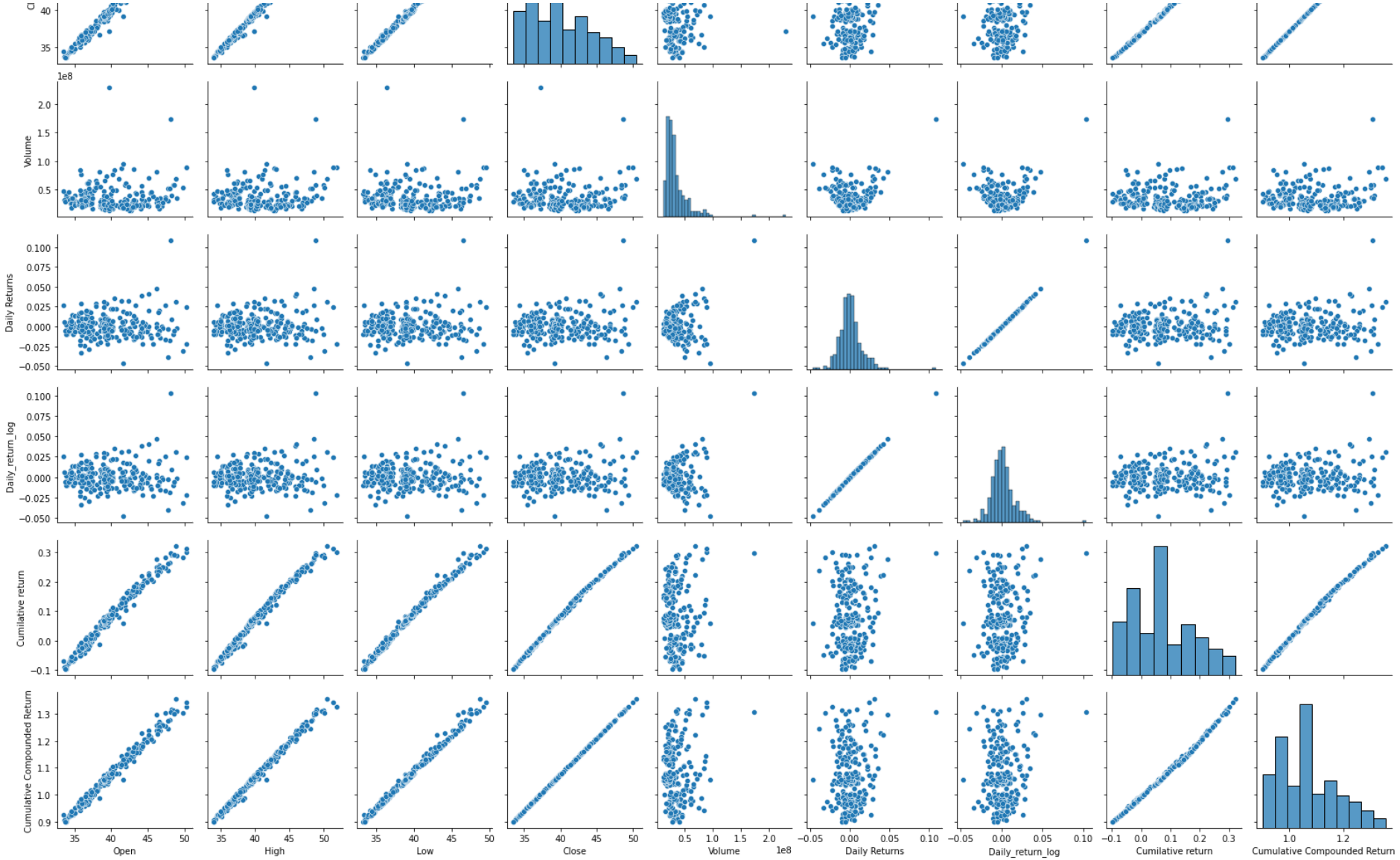
Out[37]: <AxesSubplot:>



In [38]:
```
sns.pairplot(df)
```

Out[38]: <seaborn.axisgrid.PairGrid at 0x147f2e8cdc0>





In [39]:

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge
import pyramid as pm
import pmdarima as pmd
from pmdarima import auto_arima
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
```

```
import math
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [47]:
```
#split into train and validation
#time series data
train = df[:190]

test = df[190:]
```

In [48]:
```
y_train = train[['Close']]
x_train = train.drop(['Close','Daily Returns','Daily_return_log','Cumilative return','Cumulative Compounded Return'], axis=1)

x_test = test.drop(['Close','Daily Returns','Daily_return_log','Cumilative return','Cumulative Compounded Return'], axis=1)
y_test = test[['Close']]
```

In [49]:
```
y_train.shape
```

Out[49]: (190, 1)

In [50]:
```
x_train.shape
```

Out[50]: (190, 4)

In [51]:
```
y_test.shape
```

Out[51]: (62, 1)

In [52]:
```
x_test.shape
```

Out[52]: (62, 4)

In [53]:
```
x_test.head()
```

Out[53]:

Date	Open	High	Low	Volume
2021-08-12	46.500000	47.400002	46.340000	29676400
2021-08-13	47.410000	48.500000	47.320000	35341200
2021-08-16	48.230000	48.970001	47.509998	38045800
2021-08-17	48.779999	50.490002	48.709999	68817000
2021-08-18	50.230000	51.860001	49.169998	88598000

In [54]:
```
model=LinearRegression()
```

In [55]:
```
reg=model.fit(x_train,y_train)
```

In [56]:
```
Y_pred =reg.predict(x_test)
```

In [57]:
```
print('intercept=',reg.intercept_[0])
```

```
intercept= -0.21807288658575175
```

In [58]:
```
print("coefficients=",reg.coef_)
```

```
coefficients= [[-5.48123396e-01  7.11739374e-01  8.42511713e-01  8.44669230e-10]]
```

In [59]:
```
print('Accuracy of model is:'+str(r2_score(y_test,Y_pred)*100)+' %')
```

```
Accuracy of model is:97.26309060272457 %
```

In [62]:
```
test['Predictions'] = 0
test['Predictions'] = Y_pred


train.index = df[:190].index
test.index = df[190:].index
```

```
plt.figure(figsize=(10,10))
plt.plot(train["Close"],label='Training')
plt.plot(test[['Close']],label='Actual')
plt.plot(test[['Predictions']],label='Predicted')
plt.legend(loc='upper left', fontsize=10)
plt.title("Actual Vs Predicted Close price",size=30)

#Orange line is actual close whereas Green line represent predicted value
```

```
<ipython-input-62-44d86367e041>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  test['Predictions'] = 0
<ipython-input-62-44d86367e041>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  test['Predictions'] = Y_pred
```

Out[62]: Text(0.5, 1.0, 'Actual Vs Predicted Close price')

In [63]:
```
training = train['Close']
validation = test['Close']
```

In [64]:

```
#model = auto_arima(training, trace=True, error_action='ignore',seasonal=True, suppress_warnings=True)
model = auto_arima(training, start_p=1, start_q=1,max_p=3, max_q=3, m=12,start_P=0, seasonal=True,d=1, D=1, trace=True,error_actio
model.fit(training)
```

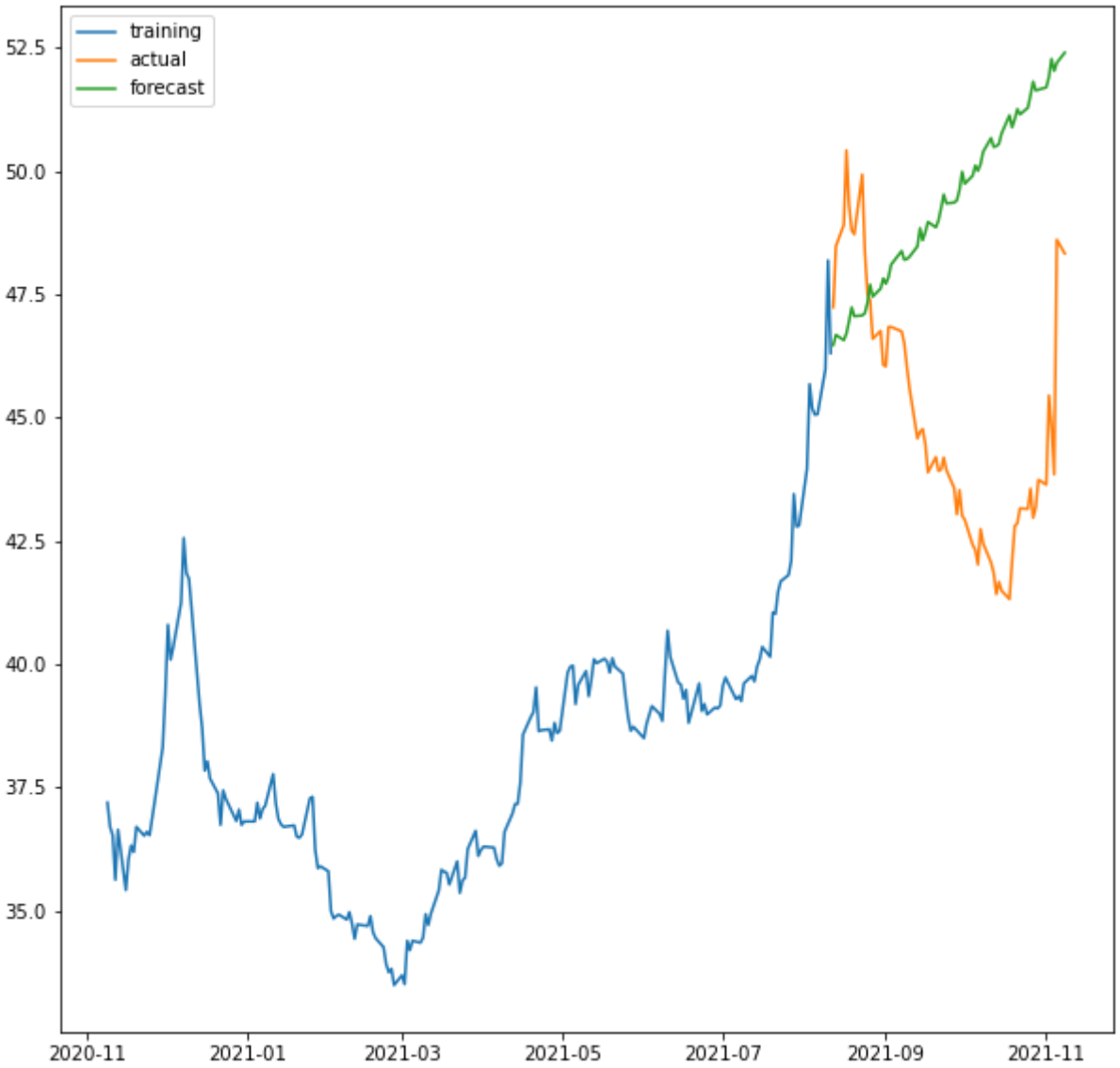
```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=325.006, Time=0.52 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=389.760, Time=0.02 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=344.594, Time=0.07 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=324.107, Time=0.19 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=391.348, Time=0.03 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=325.749, Time=0.26 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=325.764, Time=0.59 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=344.595, Time=0.08 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=inf, Time=1.60 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=322.124, Time=0.13 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=323.768, Time=0.21 sec
 ARIMA(0,1,0)(0,1,2)[12]             : AIC=323.783, Time=0.32 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=342.684, Time=0.06 sec
 ARIMA(0,1,0)(1,1,2)[12]             : AIC=inf, Time=1.73 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=324.105, Time=0.19 sec
 ARIMA(0,1,0)(0,1,1)[12] intercept   : AIC=inf, Time=0.24 sec

Best model:  ARIMA(0,1,0)(0,1,1)[12]
Total fit time: 6.275 seconds
```

Out[64]:
```
ARIMA(order=(0, 1, 0), scoring_args={}, seasonal_order=(0, 1, 1, 12),
      suppress_warnings=True, with_intercept=False)
```

In [65]:
```
forecast = model.predict(n_periods=len(test))
forecast = pd.DataFrame(forecast,index = test.index,columns=['Prediction'])
```

In [69]:
```
#plot
plt.figure(figsize=(10,10))
plt.plot(train['Close'],label='training')
plt.plot(test['Close'], label='actual')
plt.plot(forecast['Prediction'],label='forecast')
plt.legend(loc='upper left', fontsize=10)
```

Out[69]:
```
<matplotlib.legend.Legend at 0x147f702c820>
```



In [67]:
```
print(model.summary())
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                  190
Model:             SARIMAX(0, 1, 0)x(0, 1, [1], 12)   Log Likelihood                -159.062
Date:                              Tue, 09 Nov 2021   AIC                            322.124
```

```
Time:                                  12:09:16   BIC                            328.476
Sample:                                       0   HQIC                           324.700
                                          - 190
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L12      -0.8543      0.101     -8.452      0.000      -1.052      -0.656
sigma2         0.3234      0.029     11.172      0.000       0.267       0.380
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                21.34
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               1.04   Skew:                             0.23
Prob(H) (two-sided):                  0.89   Kurtosis:                         4.64
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [70]:

```
rmse=sqrt(mean_squared_error(validation, forecast))
print(rmse)
```

```
5.921153426576269
```