```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()

# For time stamps
from datetime import datetime


# The tech stocks we'll use for this analysis
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Set up End and Start times for data grab
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']
```



```
for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end)


company_list = [AAPL, GOOG, MSFT, AMZN]
company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.tail(10)
```

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

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2023-04-10	100.959999	102.199997	99.570000	102.169998	102.169998	37261200	AMAZON
2023-04-11	100.800003	101.000000	99.010002	99.919998	99.919998	60417800	AMAZON

```
AAPL.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	149.249961	151.298526	147.419124	149.456016	149.071479	7.996793e+07
std	10.637265	10.514574	10.768822	10.689015	10.675815	2.481820e+07
min	126.010002	127.769997	124.169998	125.019997	124.829399	3.519590e+07
25%	142.110001	143.854996	139.949997	142.464996	141.973434	6.406770e+07
50%	148.869995	150.919998	147.240005	149.350006	148.910004	7.482960e+07
75%	156.275002	158.154999	154.164993	156.779999	155.852943	8.904005e+07
						20e+08

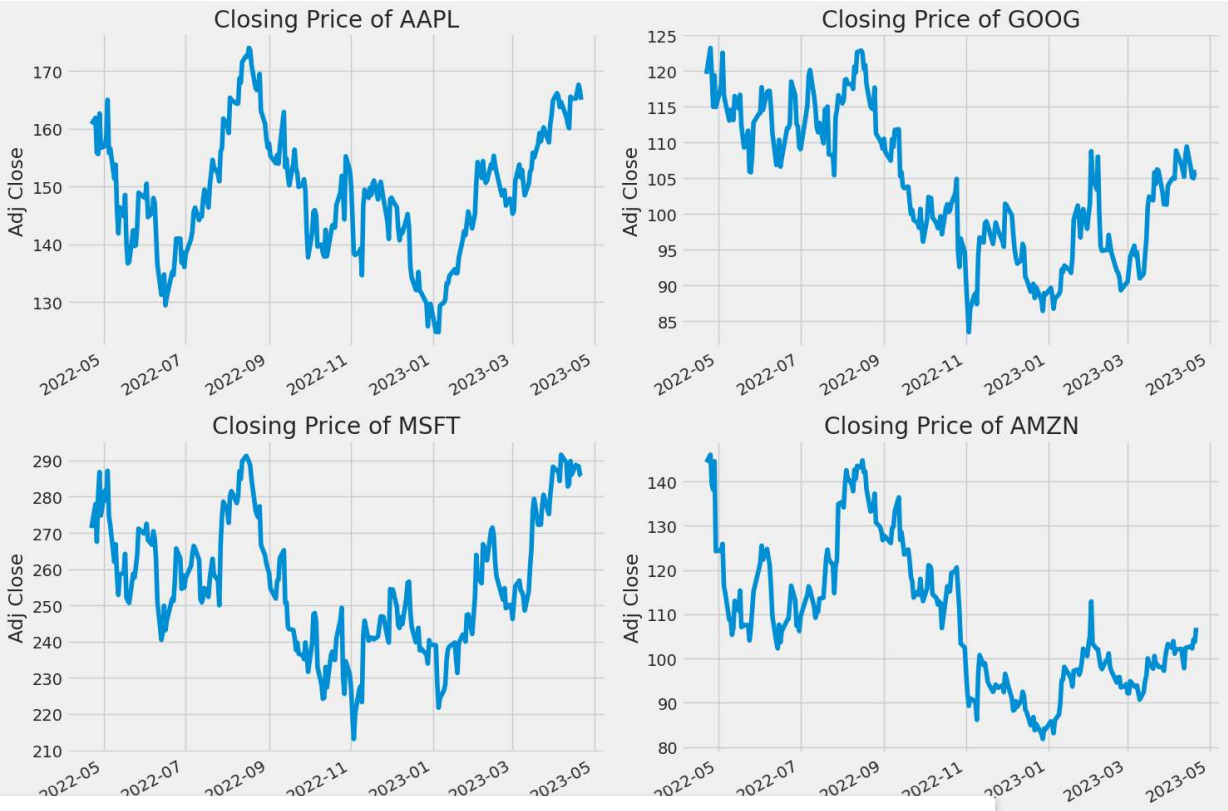
Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
AAPL.info()
```

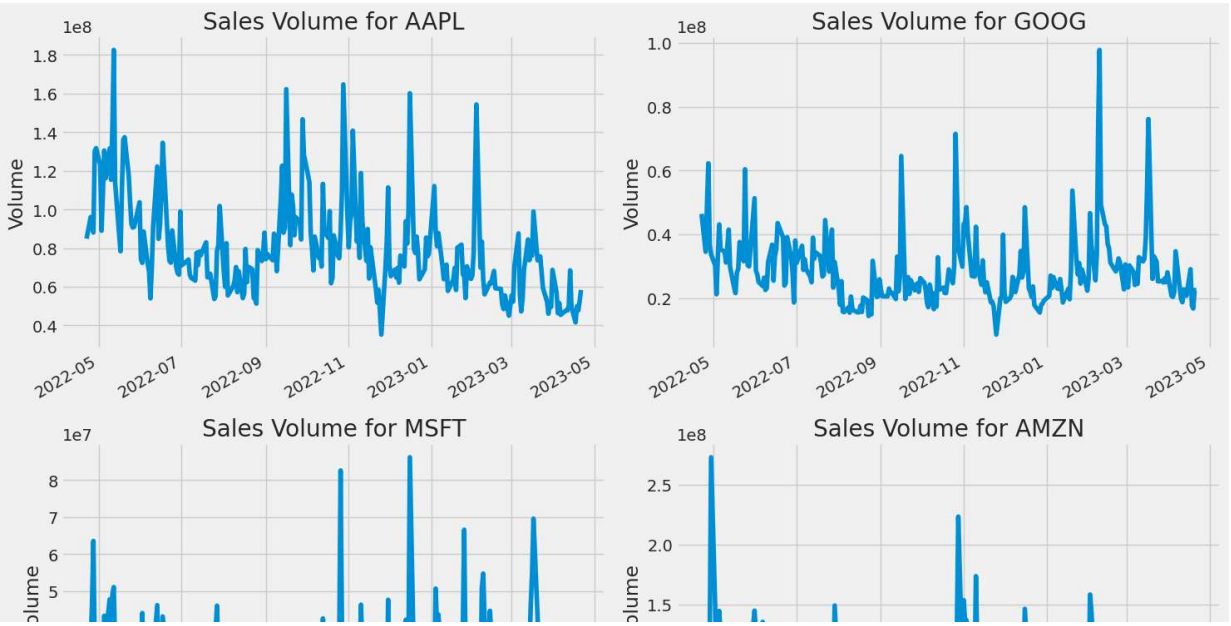
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-04-22 to 2023-04-21
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          251 non-null    float64
 1   High          251 non-null    float64
 2   Low           251 non-null    float64
 3   Close         251 non-null    float64
 4   Adj Close     251 non-null    float64
 5   Volume        251 non-null    int64  
 6   company_name  251 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.7+ KB
```

```
# Let's see a historical view of the closing price
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
```

```
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```




```
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Volume'].plot()
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```




```
ma_day = [10, 20, 50]

for ma in ma_day:
    for company in company_list:
        column_name = f"MA for {ma} days"
        company[column_name] = company['Adj Close'].rolling(ma).mean()
```

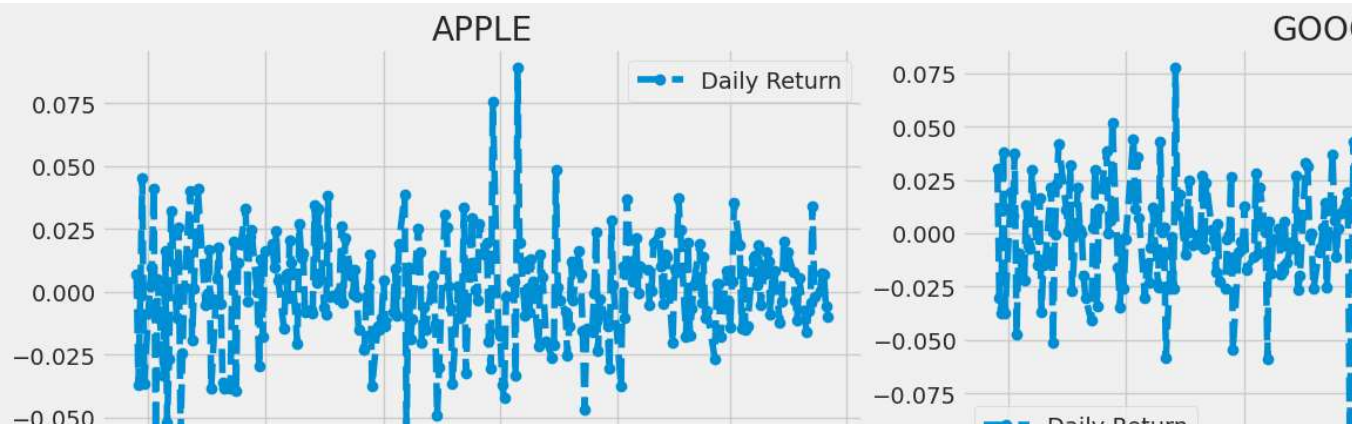


```
AAPL[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,0])
axes[0,0].set_title('APPLE')

GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,1])
axes[0,1].set_title('GOOGLE')

MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,0])
axes[1,0].set_title('MICROSOFT')

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,1])
axes[1,1].set_title('AMAZON')

fig.tight_layout()
```

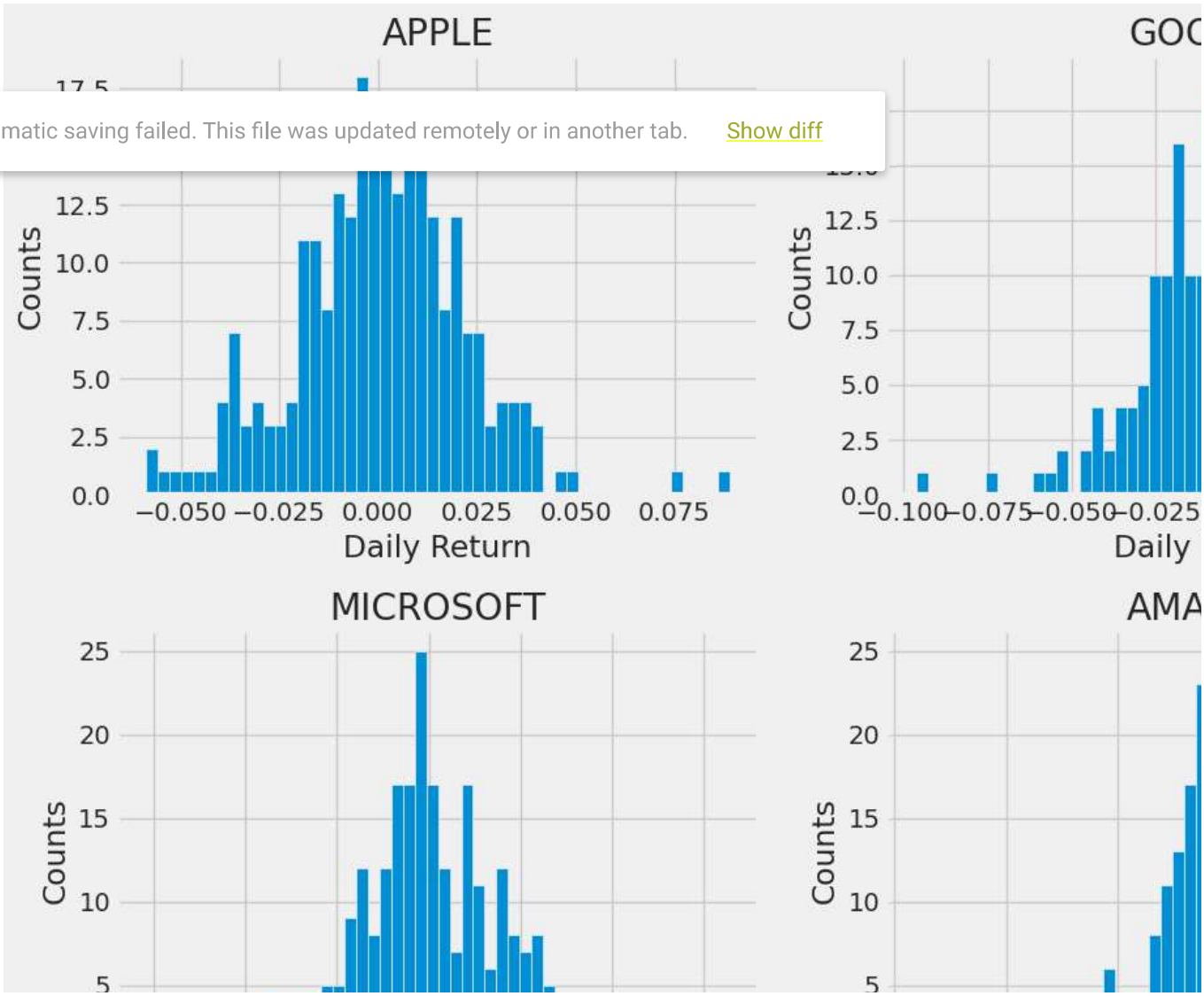

```
for company in company_list:
    company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

AAPL['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--', marker='o')
```



```
GOOG['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--', marker='o')
axes[0,1].set_title('GOOGLE')

MSFT['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--', marker='o')
axes[1,0].set_title('MICROSOFT')

AMZN['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='--', marker='o')
axes[1,1].set_title('AMAZON')

fig.tight_layout()
```




```
plt.figure(figsize=(12, 9))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Daily Return'].hist(bins=50)
    plt.xlabel('Daily Return')
    plt.ylabel('Counts')
    plt.title(f'{company_name[i - 1]}')

plt.tight_layout()
```



```
# Grab all the closing prices for the tech stock list into one DataFrame

closing_df = pdr.get_data_yahoo(tech_list, start=start, end=end)['Adj Close']

# Make a new tech returns DataFrame
tech_rets = closing_df.pct_change()
tech_rets.head()
```

```
[**********100%**********]  4 of 4 completed
```

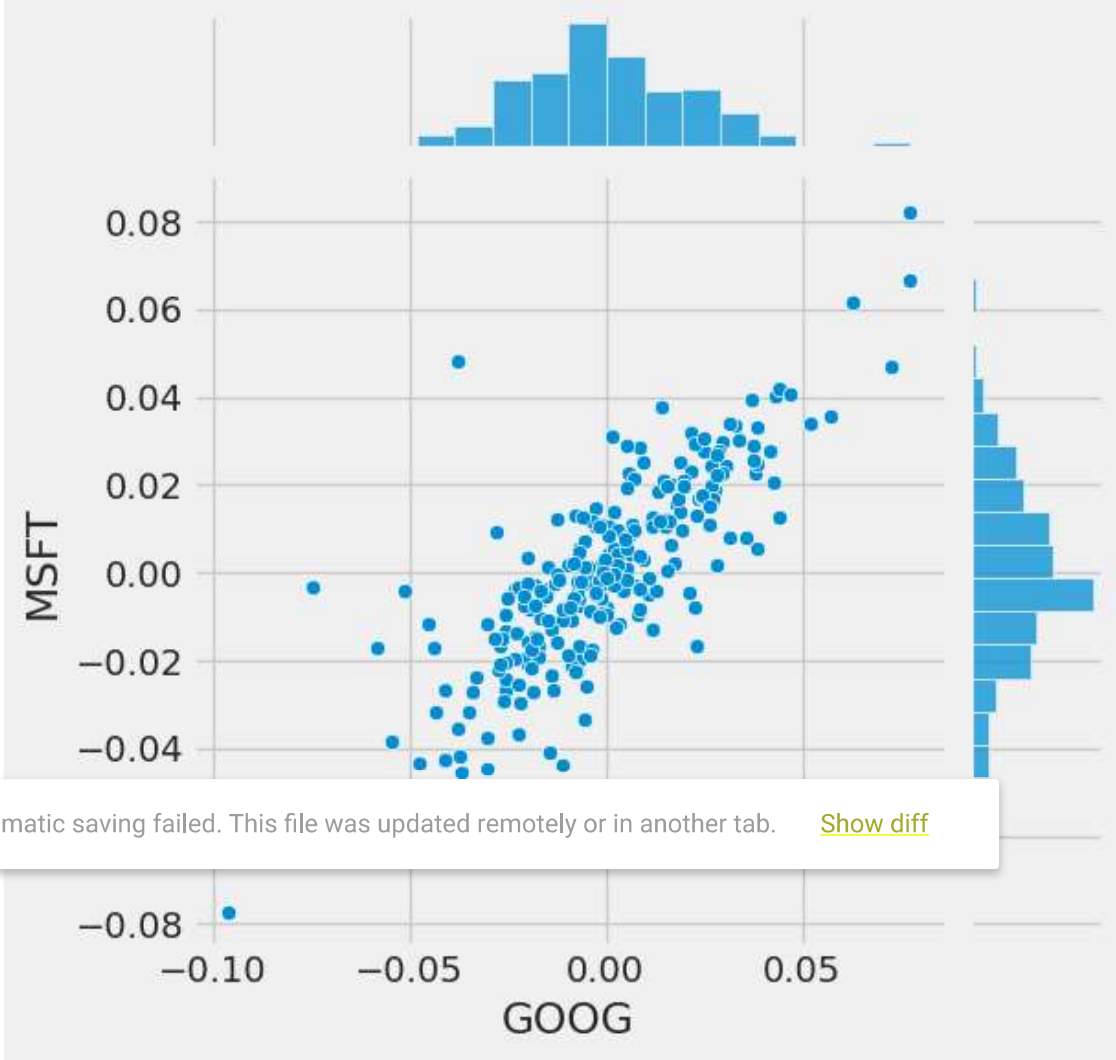
Date	AAPL	AMZN	GOOG	MSFT
2022-04-22	NaN	NaN	NaN	NaN
2022-04-25	0.006737	0.011943	0.030398	0.024414
2022-04-26	-0.037328	-0.045751	-0.030377	-0.037404
2022-04-27	-0.001467	-0.008781	-0.037534	0.048109
2022-04-28	0.045155	0.046534	0.038176	0.022633

```
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```

```
<seaborn.axisgrid.JointGrid at 0x7fccbeb2a490>
```

```
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7fccbc68cca0>
```

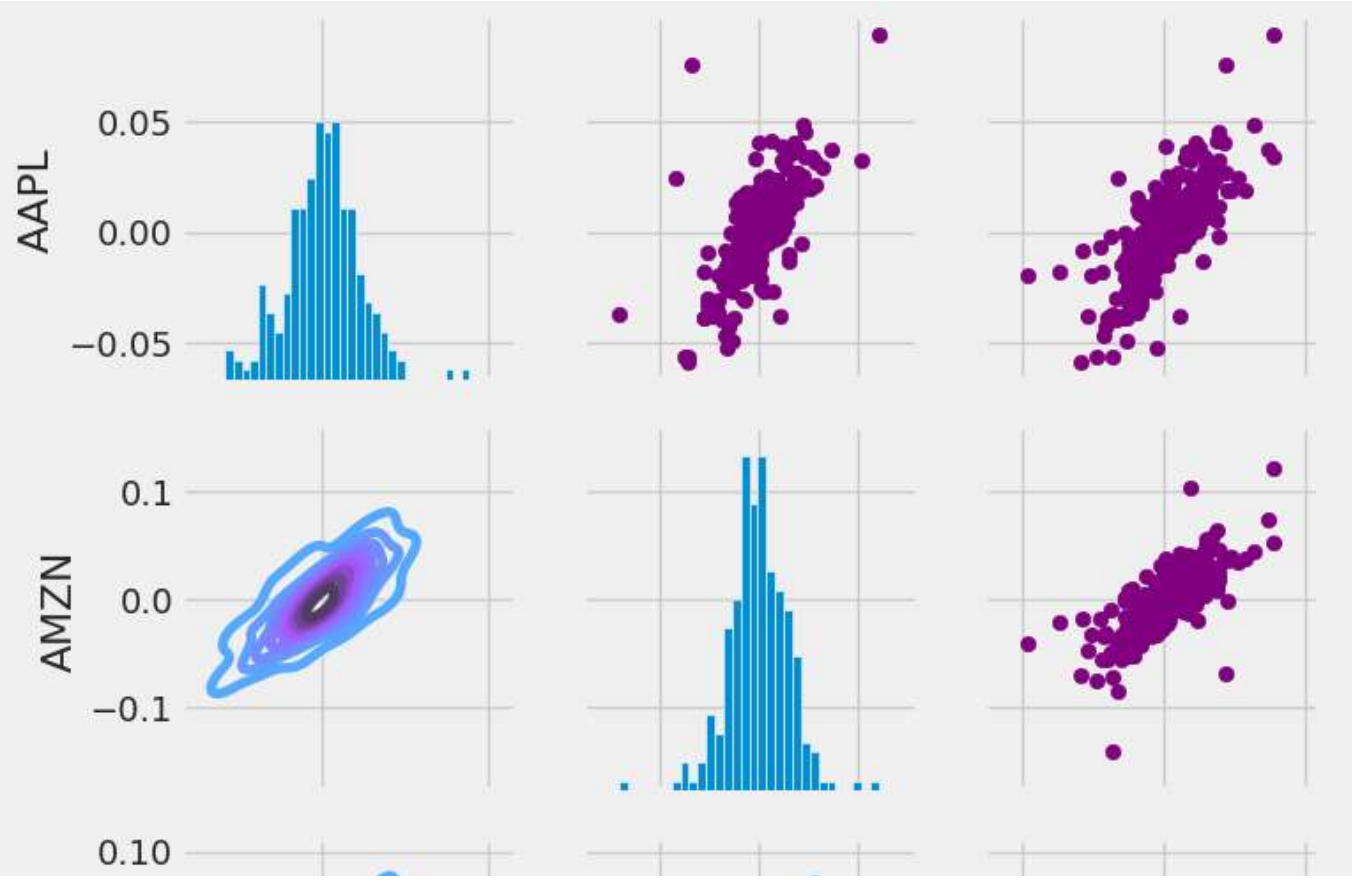



```
sns.pairplot(tech_rets, kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0x7fccbe9e09d0>
```

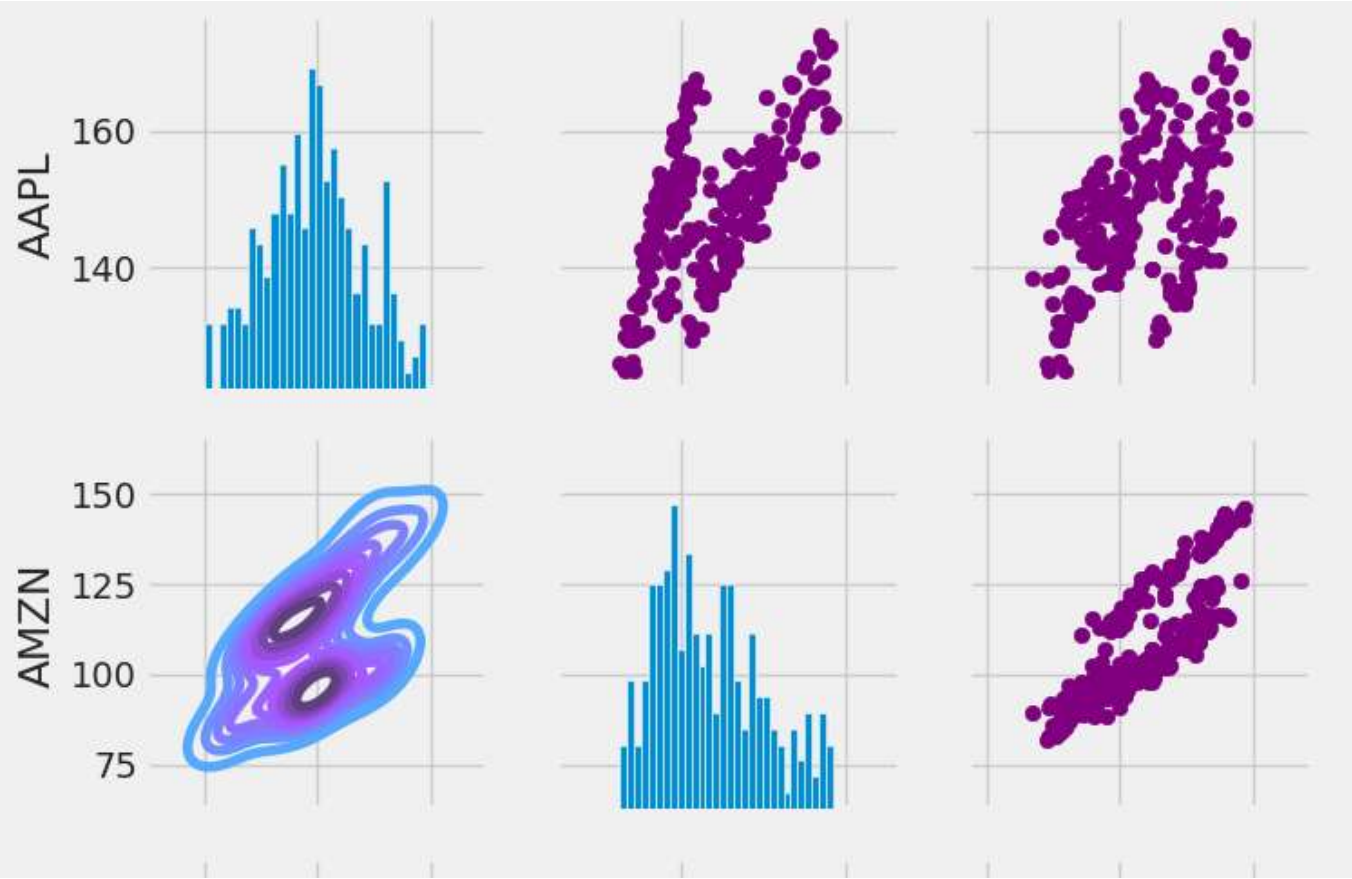

```
# Using map_upper we can specify what the upper triangle will look like.
return_fig.map_upper(plt.scatter, color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde)
# or the color map (BluePurple)
return_fig.map_lower(sns.kdeplot, cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
return_fig.map_diag(plt.hist, bins=30)
```

```
<seaborn.axisgrid.PairGrid at 0x7fccbc420e80>
```




```
# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

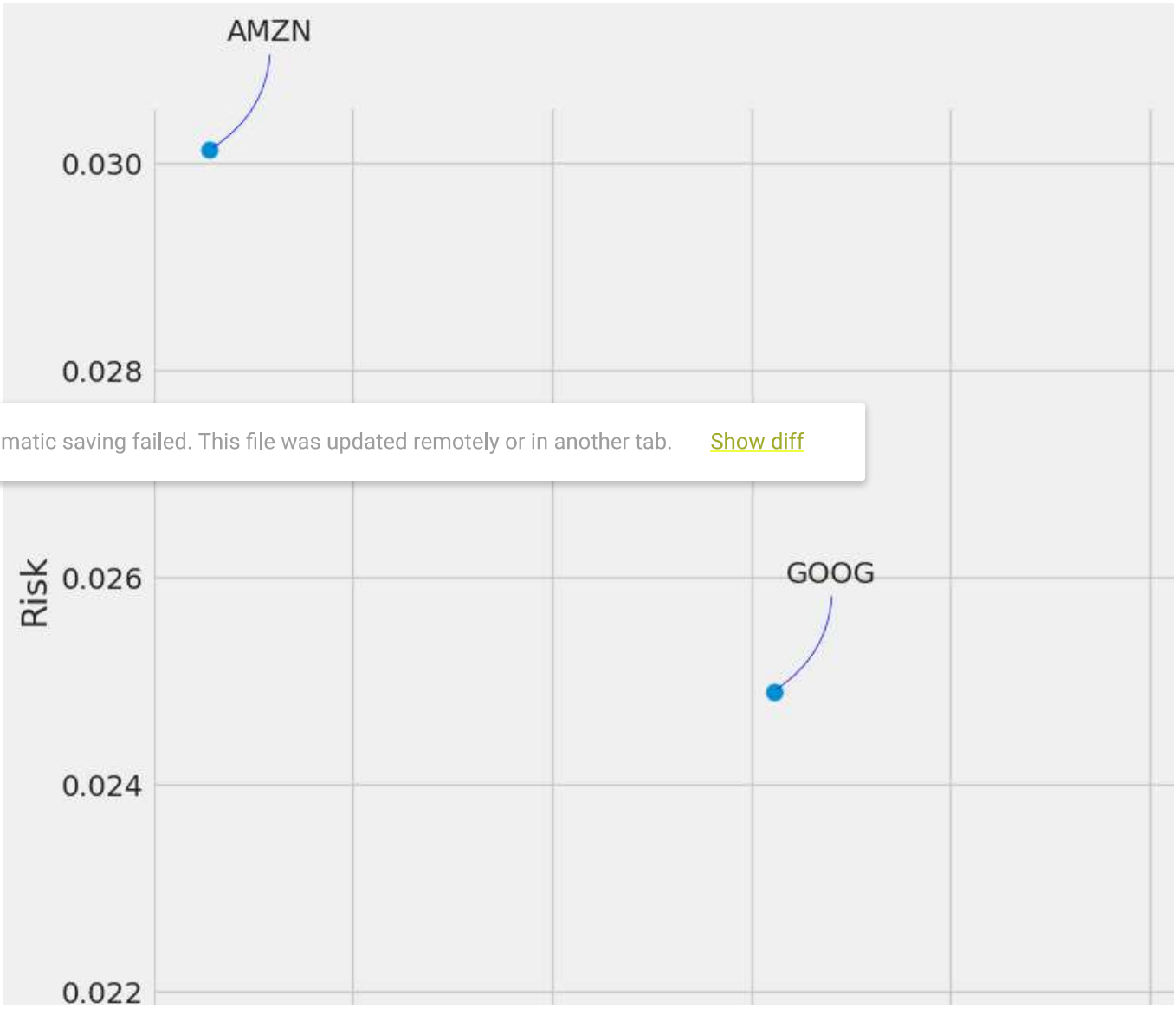
```
<seaborn.axisgrid.PairGrid at 0x7fccb8201cd0>
```




```
plt.subplot(2, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock return')

plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock closing price')
```

```
Text(0.5, 1.0, 'Correlation of stock closing price')
```

```
rets = tech_rets.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 8))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottor
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
```



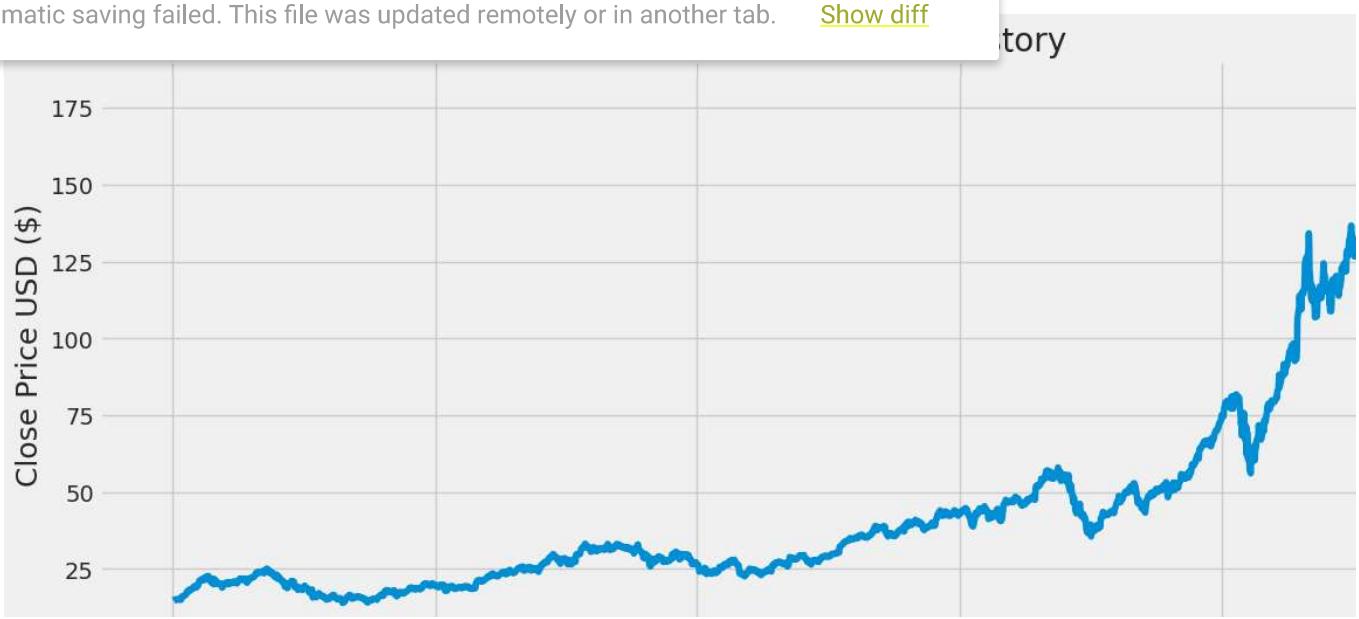

```
df = pdr.get_data_yahoo('AAPL', start='2012-01-01', end=datetime.now())
# Show teh data
df
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.500193	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.567370	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.706894	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.839728	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.819361	394024400
...	...	...	...	...	...	...
2023-04-17	165.089996	165.389999	164.029999	165.229996	165.229996	41516200
2023-04-18	166.100006	167.410004	165.649994	166.470001	166.470001	49923000
2023-04-19	165.800003	168.160004	165.539993	167.630005	167.630005	47720200
2023-04-20	166.089996	167.869995	165.559998	166.649994	166.649994	52456400

```
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff




```
data = df.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len
```

```
2702
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data

     array([[0.00439887],
            [0.00486851],
            [0.00584391],
            ...,
            [0.91443665],
            [0.90860542],
            [0.89890671]])


train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)
```



```
# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
# x_train.shape
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)

     [array([0.00439887, 0.00486851, 0.00584391, 0.00677256, 0.00663019,
            0.00695107, 0.00680444, 0.00655793, 0.00622217, 0.00726133,
            0.00819848, 0.00790947, 0.0063263 , 0.00783722, 0.00634968,
            0.01192796, 0.01149658, 0.01205972, 0.01327737, 0.01401476,
            0.01395314, 0.01372576, 0.01469479, 0.01560643, 0.01663922,
            0.01830739, 0.02181161, 0.02186474, 0.02381555, 0.02527333,
```

```
       0.0227679 , 0.02373267, 0.02371354, 0.02641875, 0.02603411,
       0.026746  , 0.02802528, 0.02873719, 0.03078787, 0.03228178,
       0.03271317, 0.03286405, 0.03030973, 0.02969346, 0.02978484,
       0.03218616, 0.03286193, 0.03431335, 0.03773469, 0.04229932,
       0.04144504, 0.04144716, 0.04474738, 0.04578017, 0.04504489,
       0.04437338, 0.04367423, 0.04599691, 0.04759072, 0.04825798])]
[0.04660893460974819]

[array([0.00439887, 0.00486851, 0.00584391, 0.00677256, 0.00663019,
       0.00695107, 0.00680444, 0.00655793, 0.00622217, 0.00726133,
       0.00819848, 0.00790947, 0.0063263 , 0.00783722, 0.00634968,
       0.01192796, 0.01149658, 0.01205972, 0.01327737, 0.01401476,
       0.01395314, 0.01372576, 0.01469479, 0.01560643, 0.01663922,
       0.01830739, 0.02181161, 0.02186474, 0.02381555, 0.02527333,
       0.0227679 , 0.02373267, 0.02371354, 0.02641875, 0.02603411,
       0.026746  , 0.02802528, 0.02873719, 0.03078787, 0.03228178,
       0.03271317, 0.03286405, 0.03030973, 0.02969346, 0.02978484,
       0.03218616, 0.03286193, 0.03431335, 0.03773469, 0.04229932,
       0.04144504, 0.04144716, 0.04474738, 0.04578017, 0.04504489,
       0.04437338, 0.04367423, 0.04599691, 0.04759072, 0.04825798]), array([0.00486851, 0.005843
       0.00680444, 0.00655793, 0.00622217, 0.00726133, 0.00819848,
       0.00790947, 0.0063263 , 0.00783722, 0.00634968, 0.01192796,
       0.01149658, 0.01205972, 0.01327737, 0.01401476, 0.01395314,
       0.01372576, 0.01469479, 0.01560643, 0.01663922, 0.01830739,
       0.02181161, 0.02186474, 0.02381555, 0.02527333, 0.0227679 ,
       0.02373267, 0.02371354, 0.02641875, 0.02603411, 0.026746  ,
       0.02802528, 0.02873719, 0.03078787, 0.03228178, 0.03271317,
       0.03286405, 0.03030973, 0.02969346, 0.02978484, 0.03218616,
       0.03286193, 0.03431335, 0.03773469, 0.04229932, 0.04144504,
```



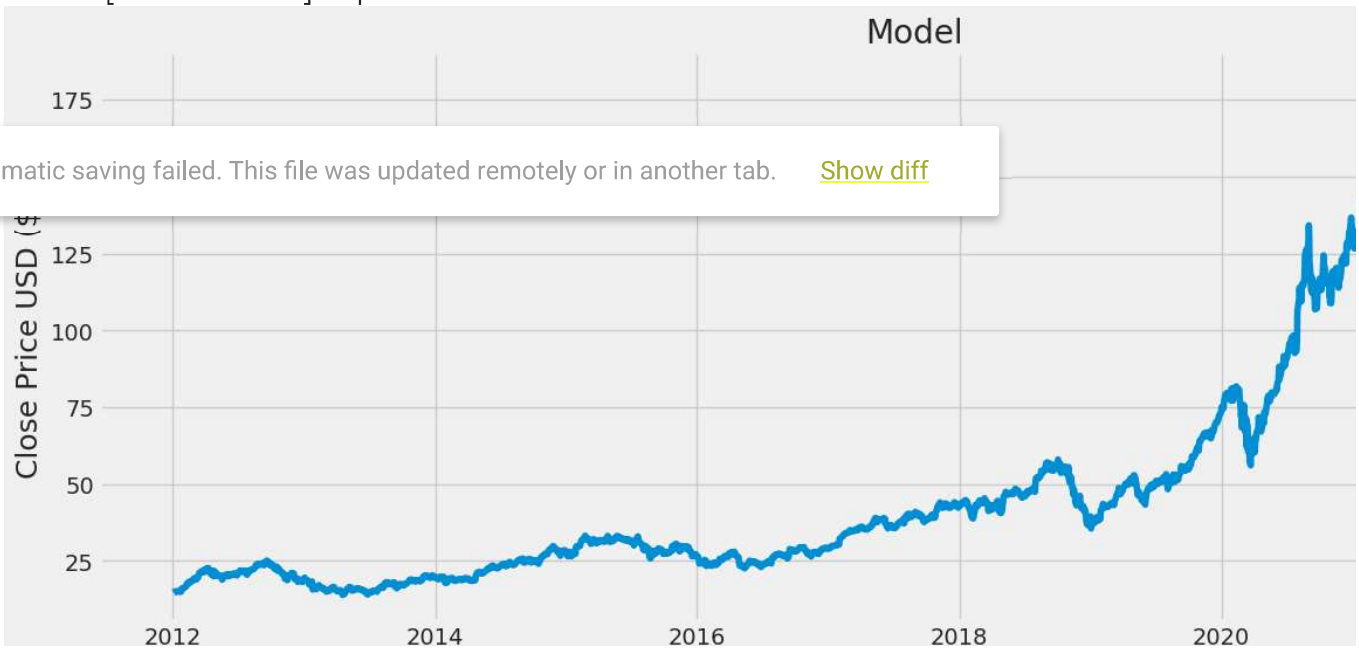
```
2642/2642 [==============================] - 105s 39ms/step - loss: 0.0013
<keras.callbacks.History at 0x7fcc537aad90>
```

```
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
rmse
```

```
5/5 [==============================] - 1s 34ms/step
9.305078389608145
```

```
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

```
<ipython-input-25-bac245597b66>:3: 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/in
  valid['Predictions'] = predictions
```




```
# Show the valid and predicted prices
valid
```

Date	Close	Predictions
2022-09-28	149.839996	161.110489
2022-09-29	142.479996	160.471436
2022-09-30	138.199997	158.337860
2022-10-03	142.449997	155.207245
2022-10-04	146.100006	153.229111
...	...	...
2023-04-17	165.229996	172.455887
2023-04-18	166.470001	173.155518
2023-04-19	167.630005	174.006805
2023-04-20	166.649994	174.983276
2023-04-21	165.020004	175.583923

142 rows × 2 columns