```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'tesla-stock-price:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F1007%2F1814%2Fbundle%2Farchive.zip%3F

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading tesla-stock-price, 34499 bytes compressed
[==================================================] 34499 bytes downloaded
Downloaded and uncompressed: tesla-stock-price
Downloading sandp500, 20283917 bytes compressed
[==================================================] 20283917 bytes downloaded
Downloaded and uncompressed: sandp500
Downloading amzn-dpz-btc-ntfx-adjusted-may-2013may2019, 30437 bytes compressed
```

```
[==================================================] 30437 bytes downloaded
Downloaded and uncompressed: amzn-dpz-btc-ntfx-adjusted-may-2013may2019
Downloading apple-aapl-historical-stock-data, 50651 bytes compressed
[==================================================] 50651 bytes downloaded
Downloaded and uncompressed: apple-aapl-historical-stock-data
Data source import complete.
```

```
!pip install -q yfinance
```

```
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']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

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
```

	Open	High	Low	Close	Adj Close	Volume	company_na
Date							
2024-03-15	176.639999	177.929993	173.899994	174.419998	174.419998	72115500	AMAZ(
2024-03-18	175.800003	176.690002	174.279999	174.479996	174.479996	31250700	AMAZ(
2024-03-19	174.220001	176.089996	173.520004	175.899994	175.899994	26880900	AMAZ(
2024-03-20	176.139999	178.529999	174.639999	178.149994	178.149994	29947200	AMAZ(
2024-03-21	179.990005	181.419998	178.149994	178.149994	178.149994	32824300	AMAZ(
2024-03-22	177.750000	179.259995	176.750000	178.869995	178.869995	27964100	AMAZ(

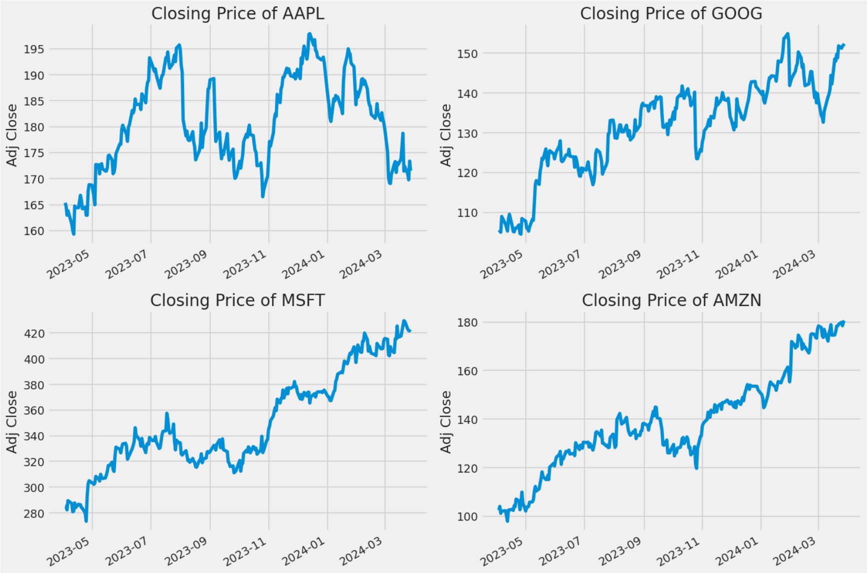
```
# Summary Stats
AAPL.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	249.000000	249.000000	249.000000	249.000000	249.000000	2.490000e+02
mean	180.897832	182.350161	179.553895	181.047109	180.589378	5.748903e+07
std	9.276276	9.139226	9.213128	9.156795	9.225381	1.727822e+07
min	161.220001	162.029999	159.779999	160.100006	159.250000	2.404830e+07
25%	173.160004	174.589996	172.050003	173.660004	173.229996	4.679290e+07
50%	180.869995	182.339996	178.970001	180.750000	180.242249	5.363130e+07
75%	189.330002	189.990005	187.779999	189.369995	188.969589	6.230330e+07
max	198.020004	199.619995	197.000000	198.110001	197.857529	1.366826e+08

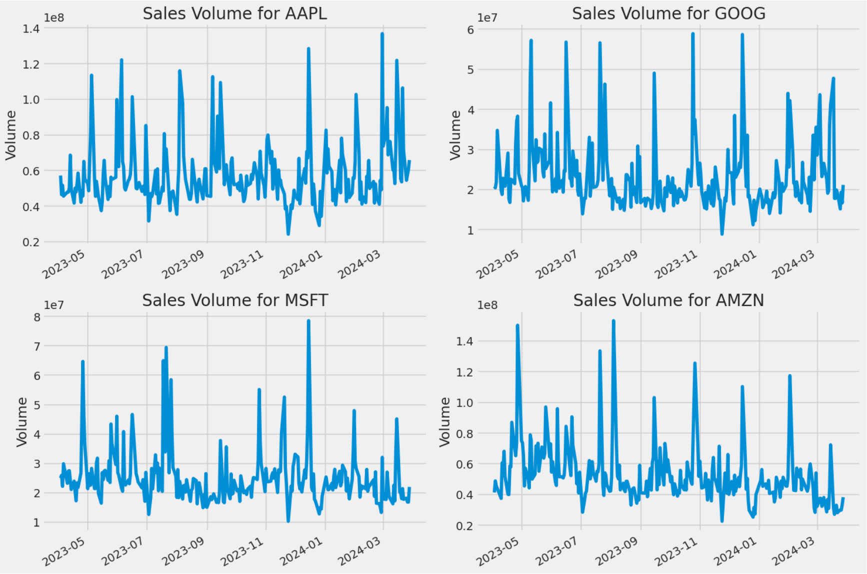
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 249 entries, 2023-04-03 to 2024-03-28
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          249 non-null    float64
 1   High          249 non-null    float64
 2   Low           249 non-null    float64
 3   Close         249 non-null    float64
 4   Adj Close     249 non-null    float64
 5   Volume        249 non-null    int64  
 6   company_name  249 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.6+ KB
```

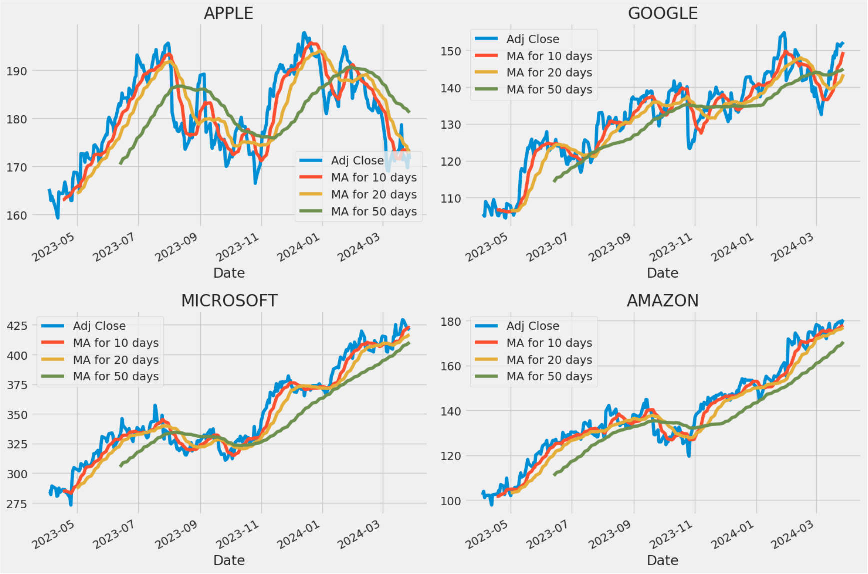
```
# Let's see a historical view of the closing price
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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()
```



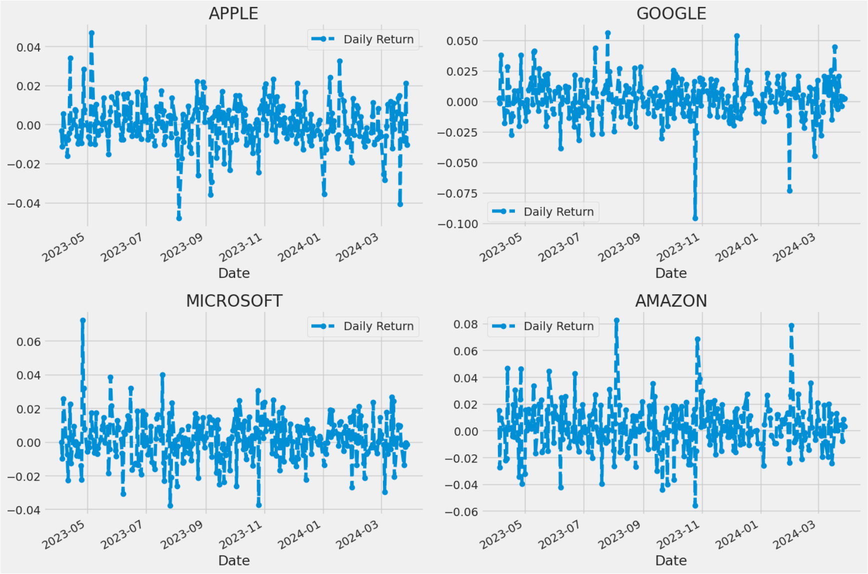
```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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()
```



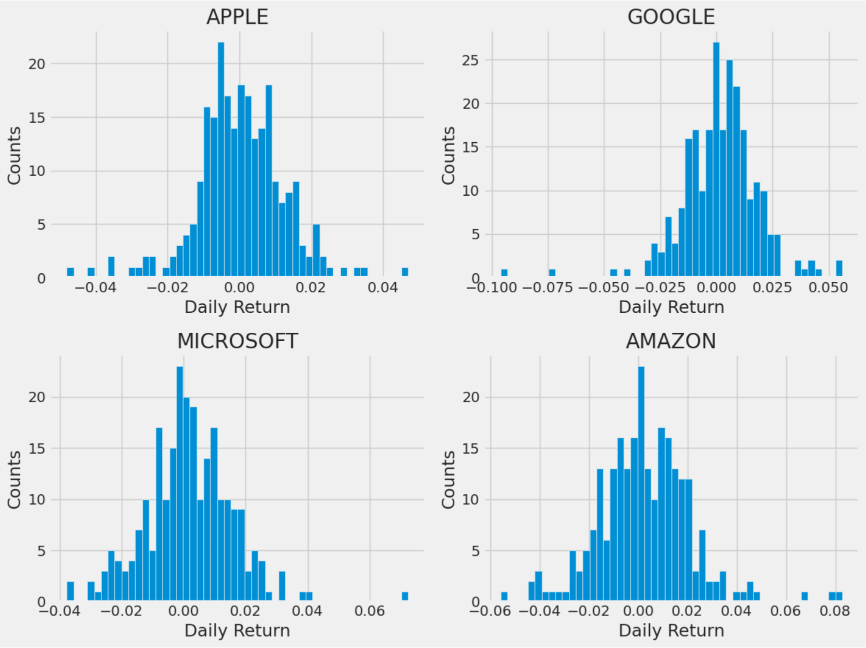
```
ma_day = [10, 20, 50]

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()


fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

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()
```



```
# We'll use pct_change to find the percent change for each day
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')
axes[0,0].set_title('APPLE')

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()
```

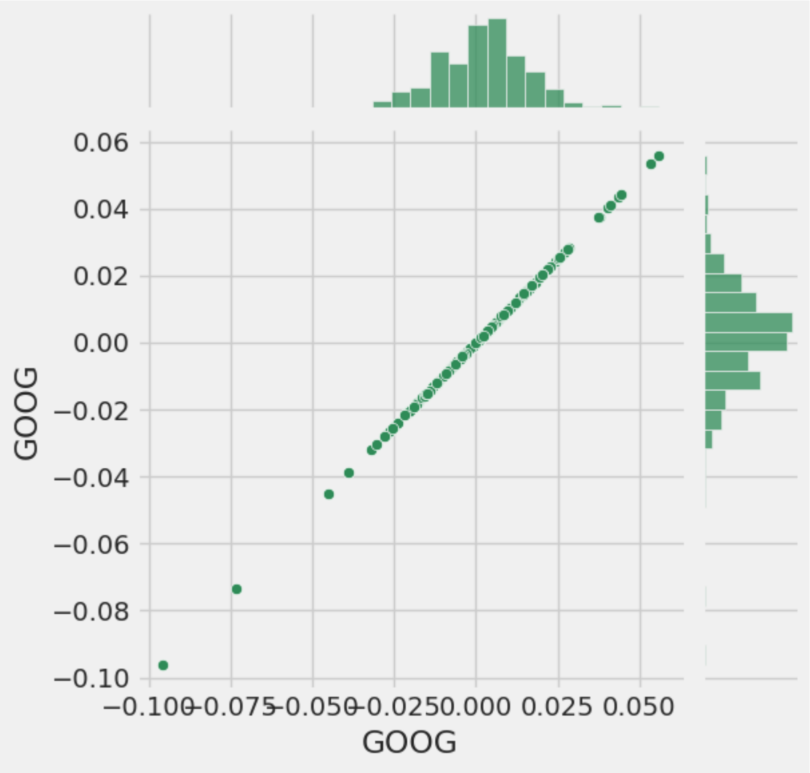
```
[*********************100%%**********************]  4 of 4 completed
```

Ticker	AAPL	AMZN	GOOG	MSFT
Date				
2023-04-03	NaN	NaN	NaN	NaN
2023-04-04	-0.003250	0.015038	0.002002	-0.000174
2023-04-05	-0.011290	-0.027417	-0.001617	-0.009889
2023-04-06	0.005496	0.009496	0.037637	0.025533
2023-04-10	-0.015972	0.001078	-0.017906	-0.007579

Next steps: Generate code with `tech_rets` View recommended plots

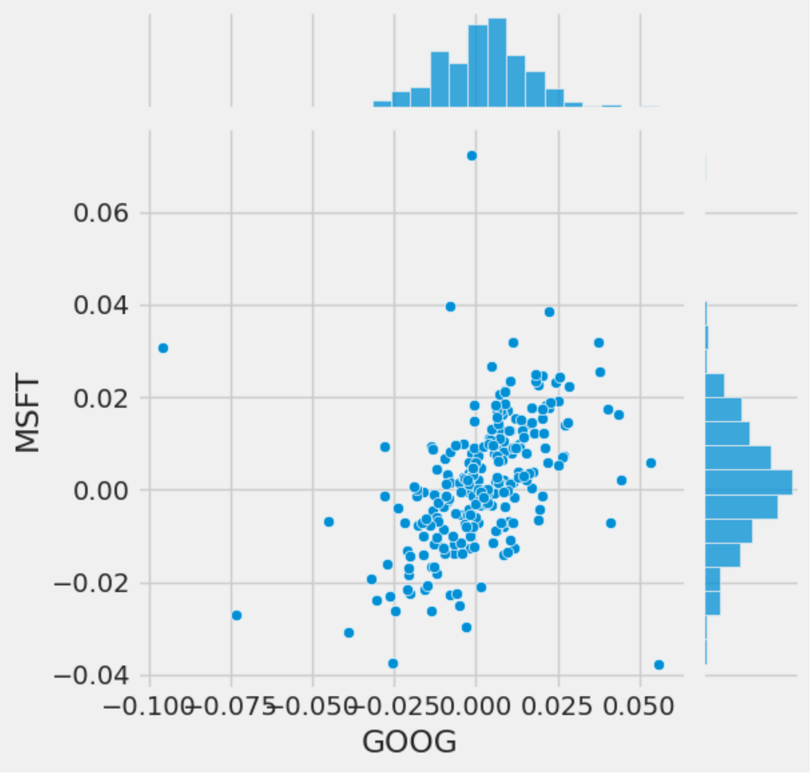
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```

```
<seaborn.axisgrid.JointGrid at 0x7b3e69758340>
```



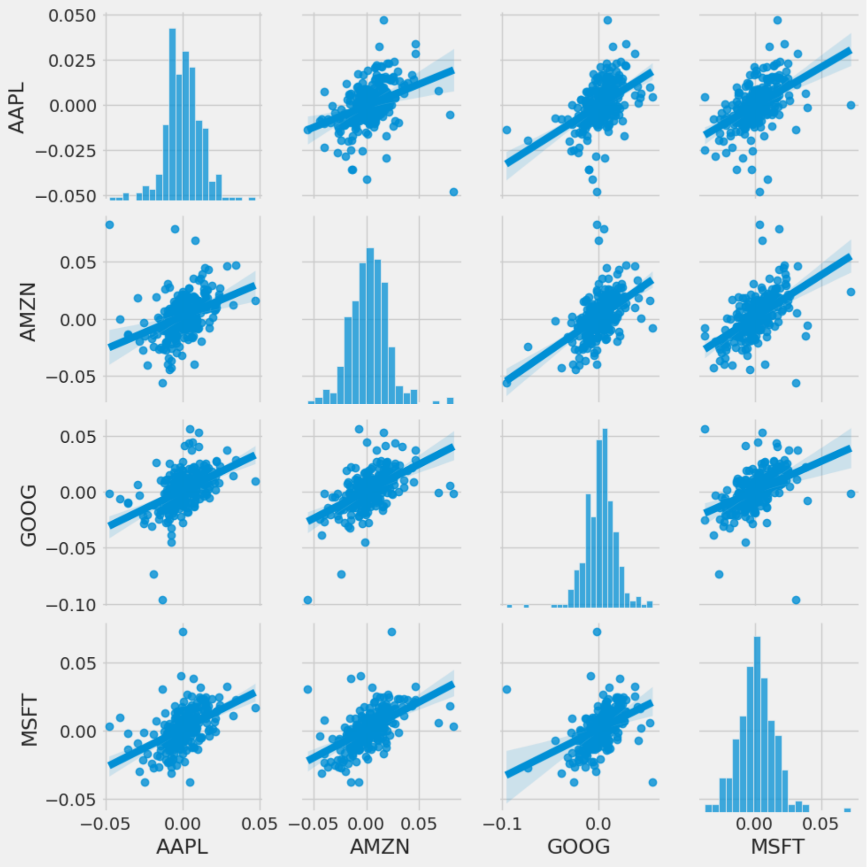
```
# We'll use joinplot to compare the daily returns of Google and Microsoft
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7b3e6975ada0>
```



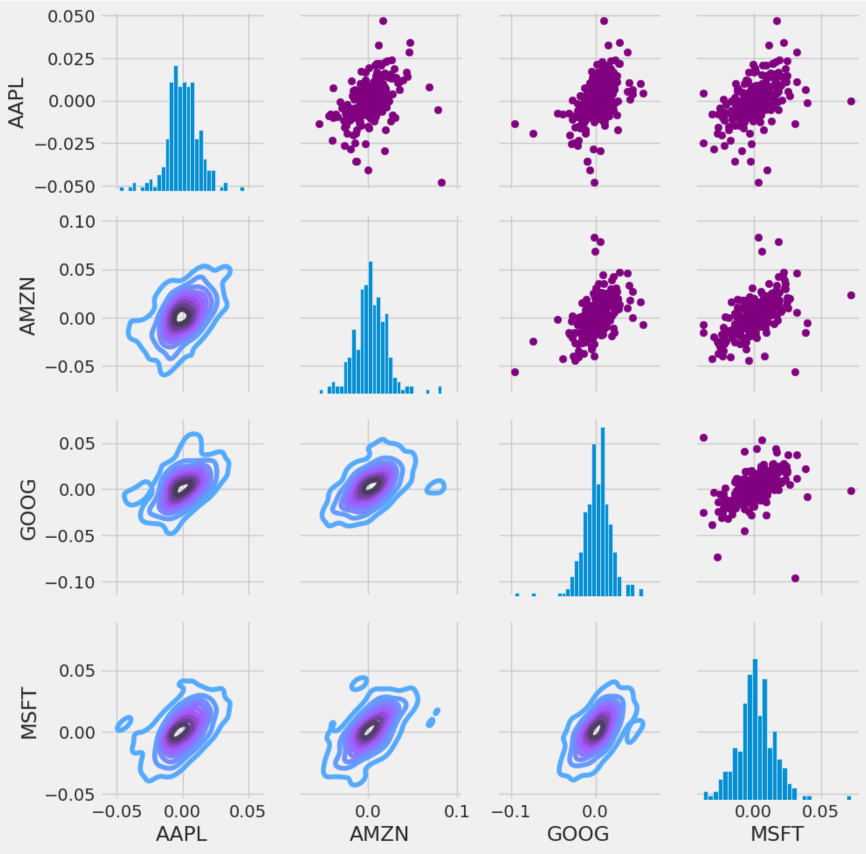
```
# We can simply call pairplot on our DataFrame for an automatic visual analysis
# of all the comparisons

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

```
<seaborn.axisgrid.PairGrid at 0x7b3e6939e7a0>
```



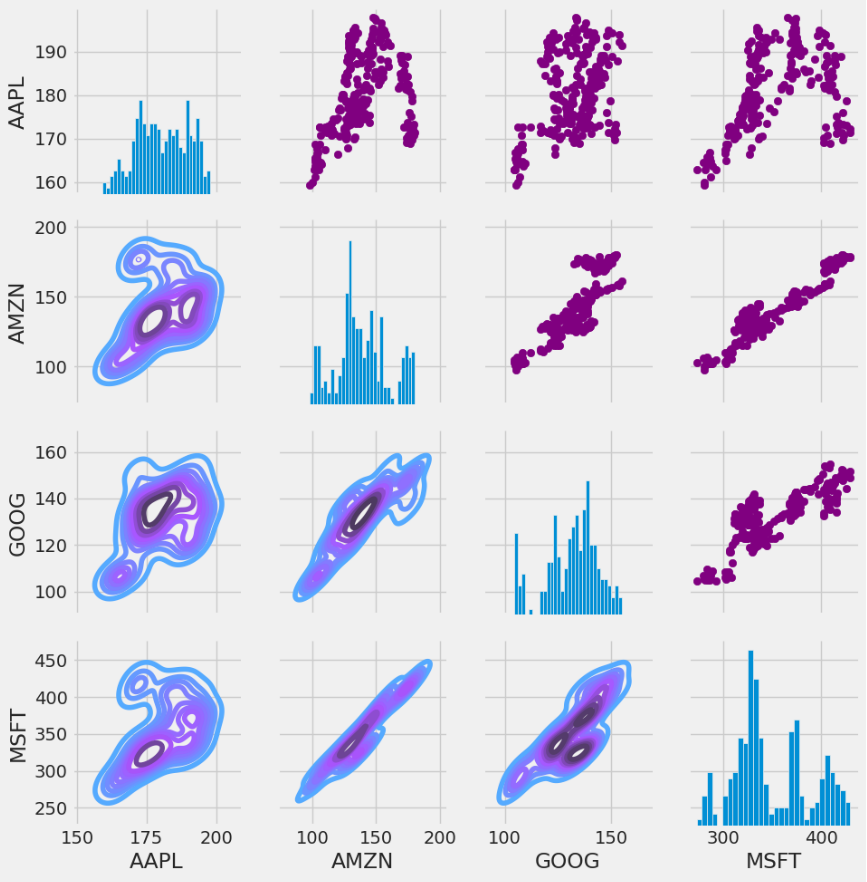
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
return_fig = sns.PairGrid(tech_rets.dropna())

# 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 0x7b3e66cd4c40>
```



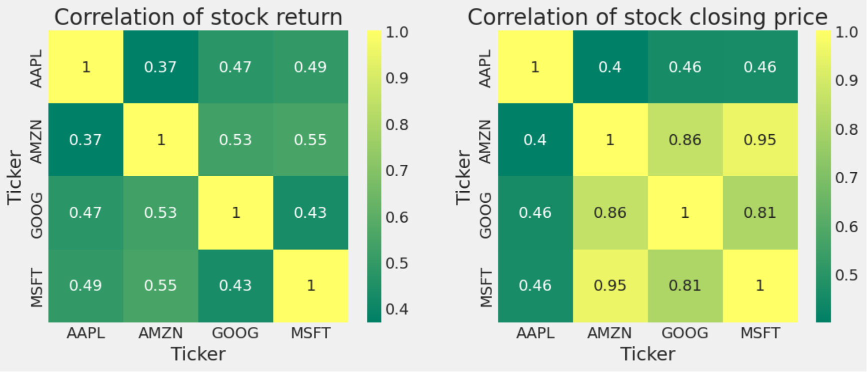
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df)

# 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 (BluePurple)
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 0x7b3e55072da0>
```

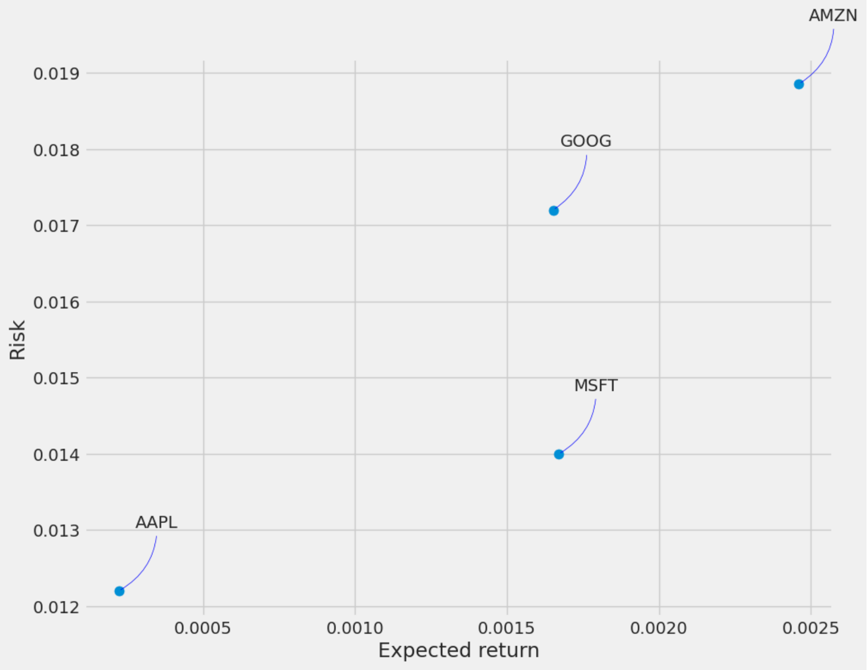


```
plt.figure(figsize=(12, 10))

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='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
```



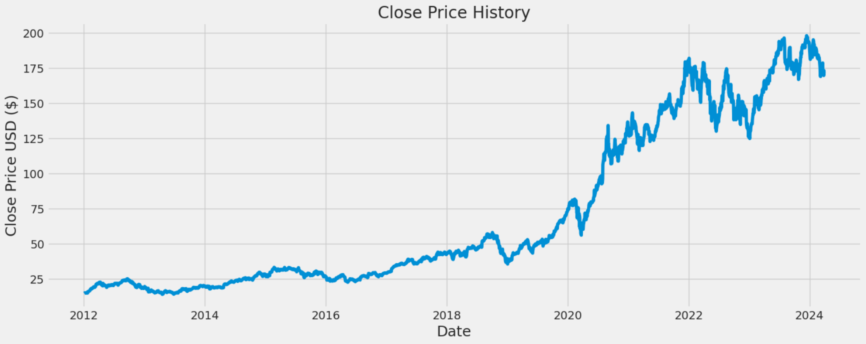
```
# Get the stock quote
df = pdr.get_data_yahoo('AAPL', start='2012-01-01', end=datetime.now())
# Show teh data
df
```

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

	Open	High	Low	Close	Adj Close	Volume
Date						
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.433827	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.500646	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.639426	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.771557	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.751297	394024400
...	...	...	...	...	...	...
2024-03-22	171.759995	173.050003	170.059998	172.279999	172.279999	71106600
2024-03-25	170.570007	171.940002	169.449997	170.850006	170.850006	54288300

Next steps: Generate code with df View recommended plots

```
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()
```



```
# Create a new dataframe with only the 'Close column
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
```

```
     2926
```

```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

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

scaled_data
```

```
     array([[0.00401431],
            [0.00444289],
            [0.00533302],
            ...,
            [0.8457884 ],
            [0.8653363 ],
            [0.85539942]])
```

```
# Create the training data set
# Create the scaled training data set
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)
        print()

# 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
```

```
     [array([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
            0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
            0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
            0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
            0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
            0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
            0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
            0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
            0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
            0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
            0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
            0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914])]
     [0.042534249860459186]

     [array([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
            0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
            0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
            0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
            0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
            0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
            0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
            0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
            0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
            0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
            0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
            0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914]), array([0.00444289, 0.00533302, 0.00618049, 0.00605056, 0.00634
            0.00620958, 0.00598462, 0.00567821, 0.00662652, 0.00748175,
            0.007218  , 0.00577323, 0.00715207, 0.00579457, 0.01088518,
            0.01049151, 0.01100542, 0.01211663, 0.01278955, 0.01273332,
            0.01252582, 0.01341013, 0.01424207, 0.01518457, 0.01670691,
            0.01990478, 0.01995326, 0.02173353, 0.02306387, 0.02077746,
            0.02165789, 0.02164044, 0.02410915, 0.02375813, 0.02440779,
            0.02557523, 0.0262249 , 0.02809631, 0.02945961, 0.02985329,
            0.02999098, 0.02765997, 0.02709757, 0.02718096, 0.02937236,
            0.02998905, 0.03131358, 0.03443581, 0.03860139, 0.0378218 ,
            0.03782373, 0.04083544, 0.04177794, 0.04110694, 0.04049413,
            0.03985611, 0.04197573, 0.0434302 , 0.04403914, 0.04253425])]
     [0.042534249860459186, 0.04053485447430975]
```