In [2]:

```
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

# For time stamps
from datetime import datetime
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is de
precated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [3]:

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

# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)


#For loop for grabing yahoo finance data and setting as a dataframe
for stock in tech_list:
    # Set DataFrame as the Stock Ticker
    globals()[stock] = DataReader(stock, 'yahoo', start, end)
```

In [4]:

```
# for company, company_name in zip(company_list, tech_list):
#     company["company_name"] = company_name
```

In [5]:

```
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)
```

Out[5]:

Date	High	Low	Open	Close	Volume	Adj Close	company_name
2020-08-13	3217.520020	3155.000000	3182.989990	3161.020020	3149000.0	3161.020020	AMAZON
2020-08-14	3178.239990	3120.000000	3178.179932	3148.020020	2751700.0	3148.020020	AMAZON
2020-08-17	3194.969971	3154.179932	3173.120117	3182.409912	2691200.0	3182.409912	AMAZON
2020-08-18	3320.000000	3205.820068	3212.000000	3312.489990	5346000.0	3312.489990	AMAZON
2020-08-19	3315.899902	3256.000000	3303.010010	3260.479980	4185100.0	3260.479980	AMAZON
2020-08-20	3312.620117	3238.000000	3252.000000	3297.370117	3332500.0	3297.370117	AMAZON
2020-08-21	3314.399902	3275.389893	3295.000000	3284.719971	3575900.0	3284.719971	AMAZON
2020-08-24	3380.320068	3257.560059	3310.149902	3307.459961	4666300.0	3307.459961	AMAZON
2020-08-25	3357.399902	3267.000000	3294.989990	3346.489990	3986300.0	3346.489990	AMAZON
2020-08-26	3451.738770	3344.567383	3351.110107	3441.850098	6508743.0	3441.850098	AMAZON

In [6]:

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

Out[6]:

	High	Low	Open	Close	Volume	Adj Close
count	253.000000	253.000000	253.000000	253.000000	2.530000e+02	253.000000
mean	304.205138	296.953439	300.176861	301.065889	3.674888e+07	299.657823
std	66.012378	64.332049	65.135634	65.454433	1.772548e+07	65.984301
min	205.720001	203.320007	204.100006	204.160004	1.165440e+07	202.154251
25%	258.679993	249.399994	255.600006	257.130005	2.514150e+07	254.603882

50%	293.679993	285.220001	286.529999	289.320007	3.177790e+07	287.814392
75%	325.619995	320.779999	323.519989	324.339996	4.222380e+07	322.758057
max	515.140015	500.329987	514.789978	506.089996	1.067212e+08	506.089996

In [7]:

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 253 entries, 2019-08-27 to 2020-08-26
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   High          253 non-null    float64
 1   Low           253 non-null    float64
 2   Open          253 non-null    float64
 3   Close         253 non-null    float64
 4   Volume        253 non-null    float64
 5   Adj Close     253 non-null    float64
 6   company_name  253 non-null    object 
dtypes: float64(6), object(1)
memory usage: 15.8+ KB
```

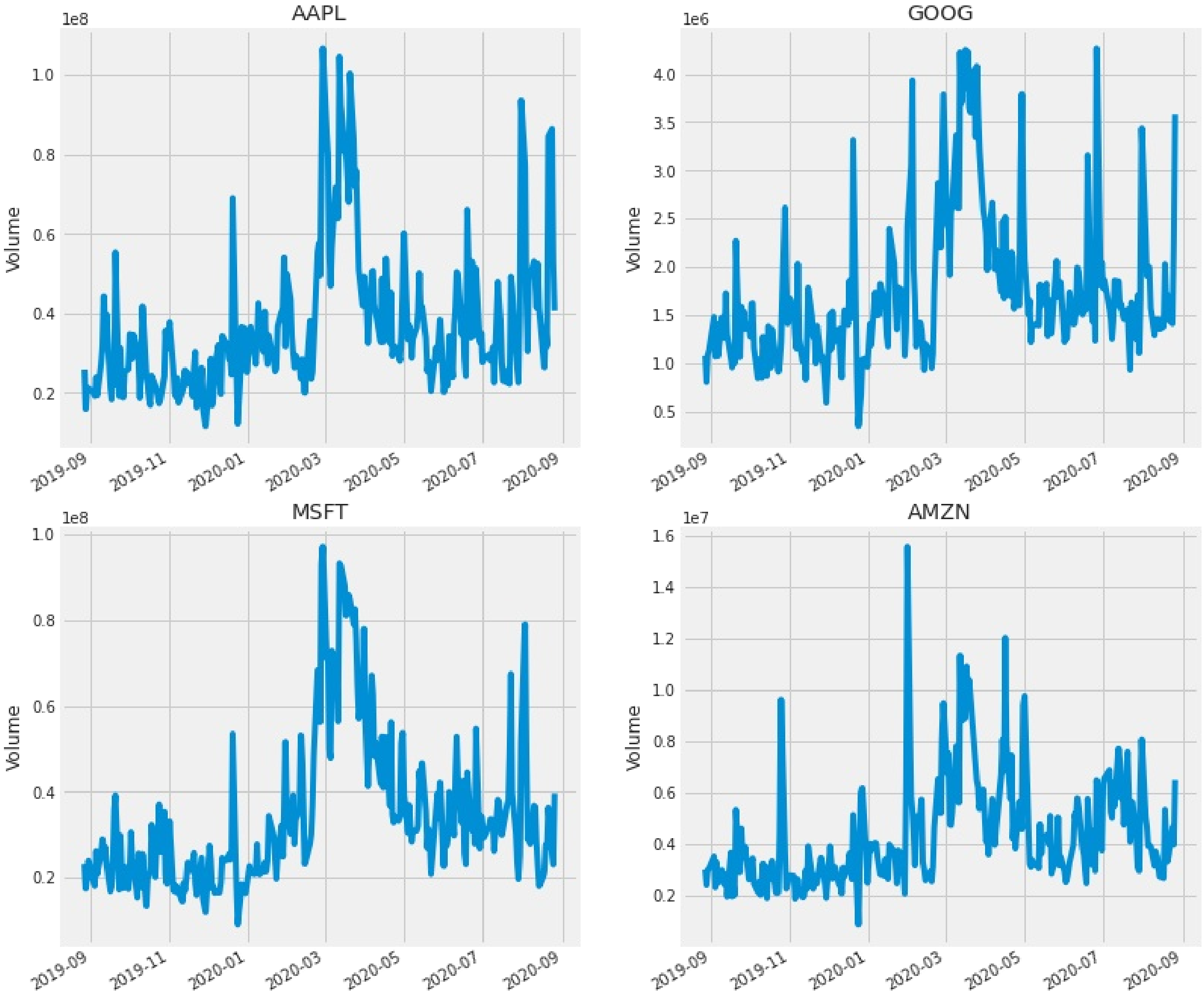
In [8]:

```
# Let's see a historical view of the closing price


plt.figure(figsize=(12, 8))
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"{tech_list[i - 1]}")
```

In [9]:

```
# Now let's plot the total volume of stock being traded each day
```

```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(12, 8))
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"{tech_list[i - 1]}")
```



In [10]:

```
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()
```

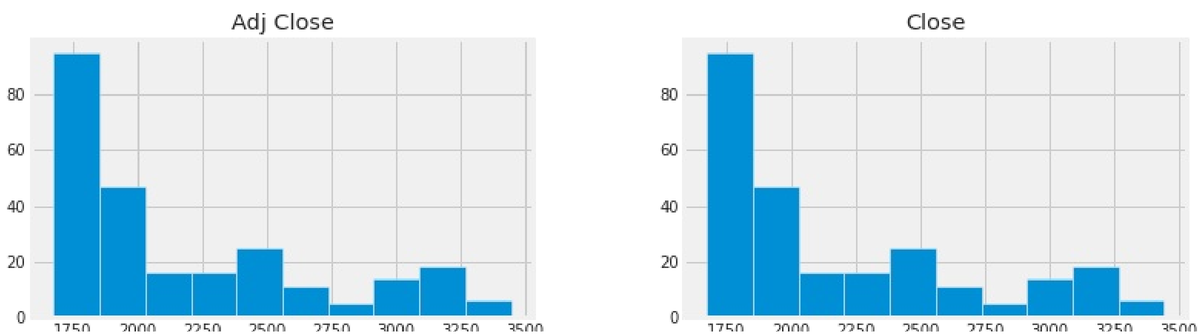
In [11]:

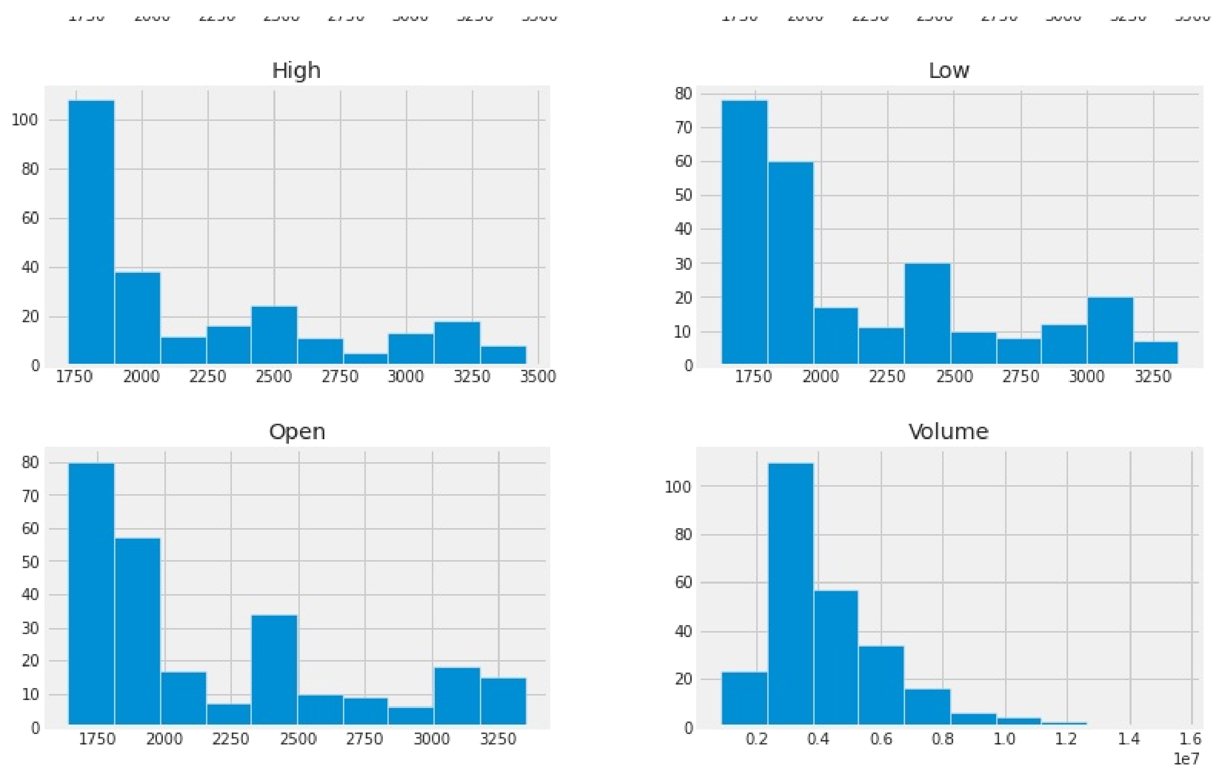
```
print(GOOG.columns)
```

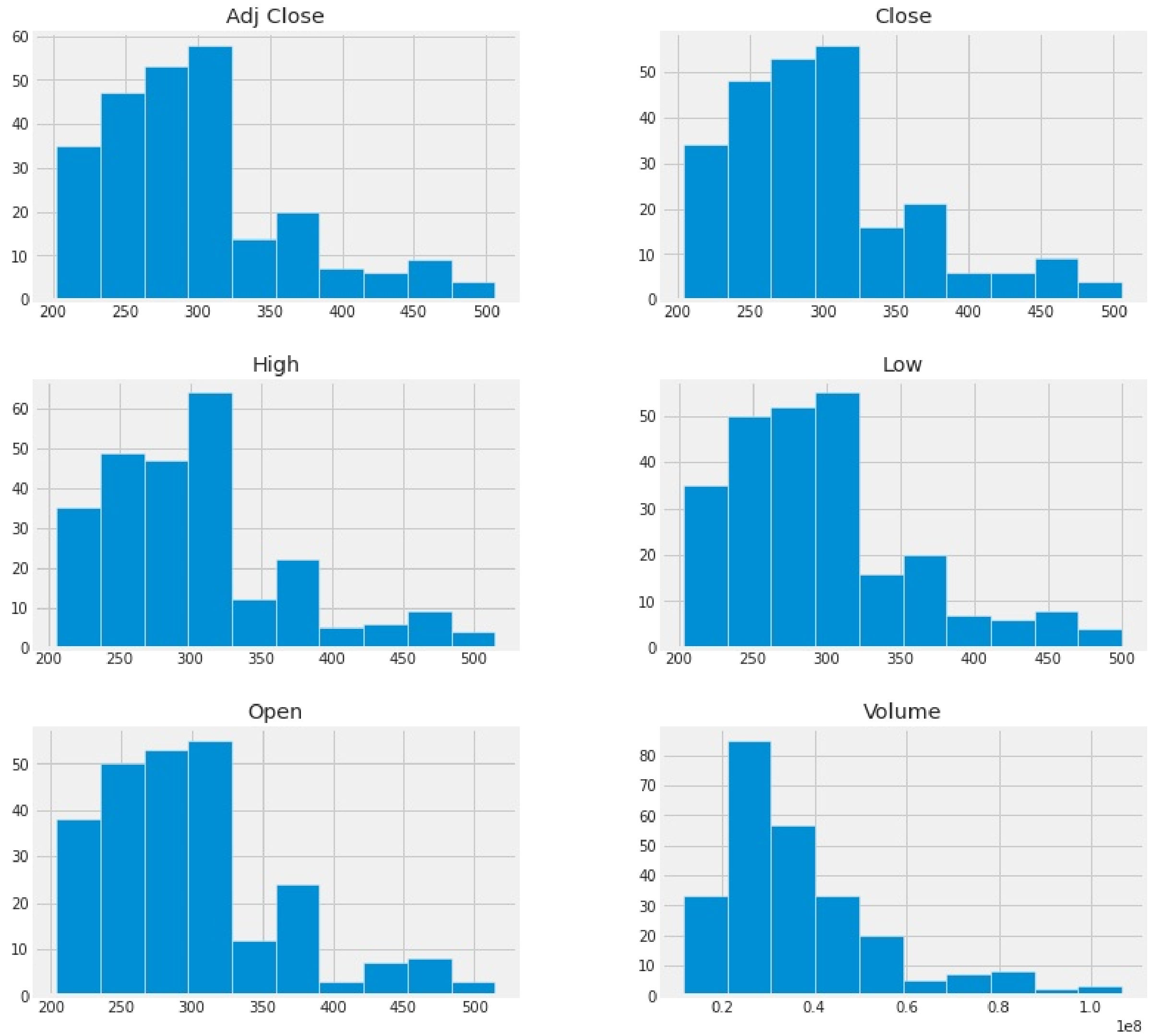
```
Index(['High', 'Low', 'Open', 'Close', 'Volume', 'Adj Close', 'company_name',
       'MA for 10 days', 'MA for 20 days', 'MA for 50 days'],
      dtype='object')
```

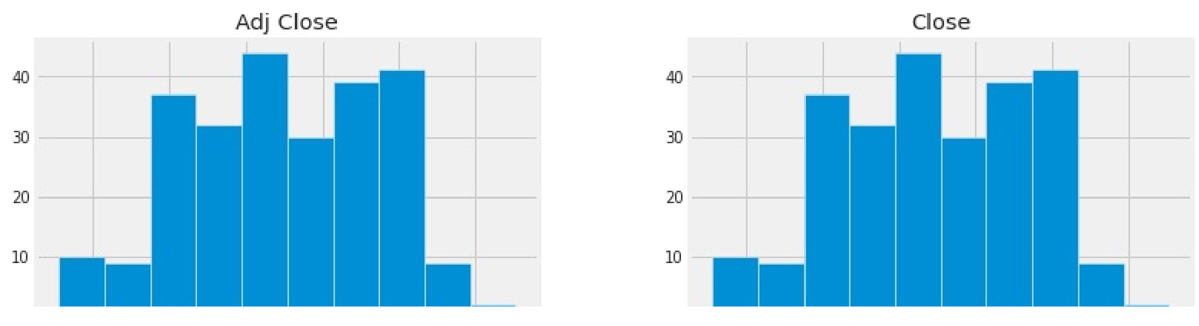
In [12]:

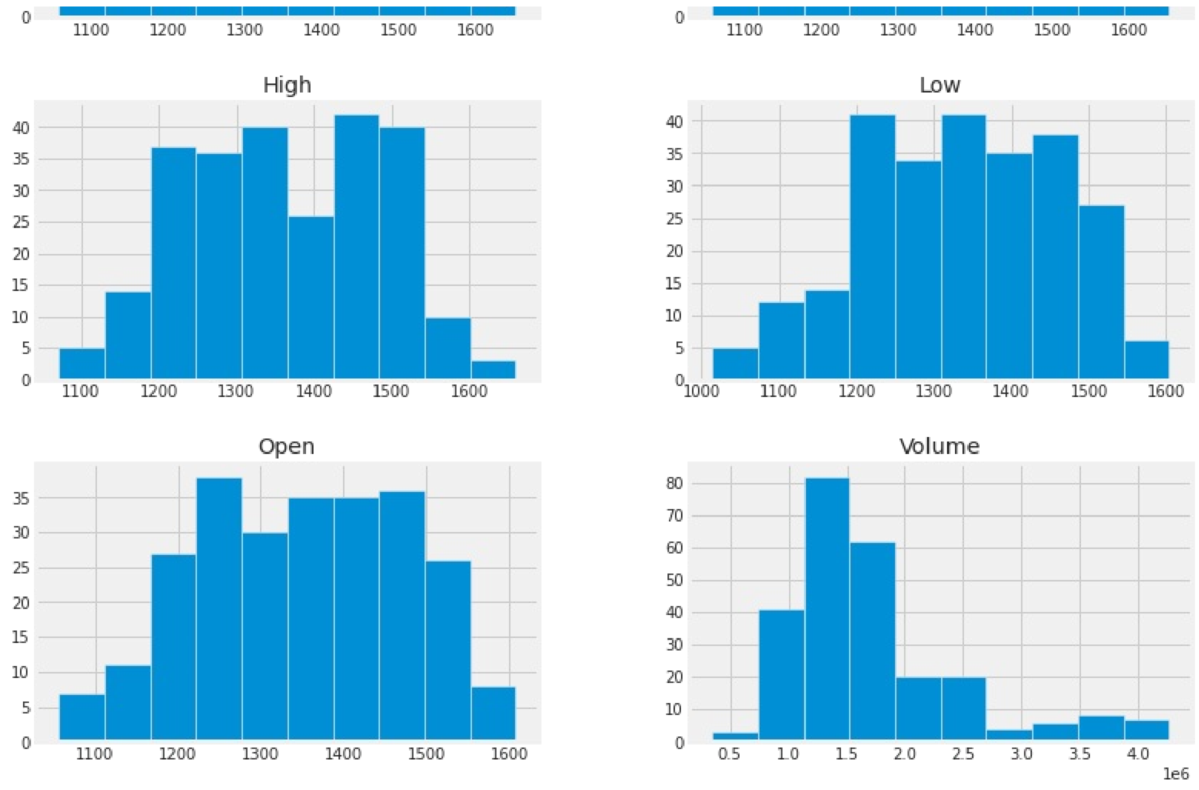
```
df.groupby("company_name").hist(figsize=(12, 12));
```

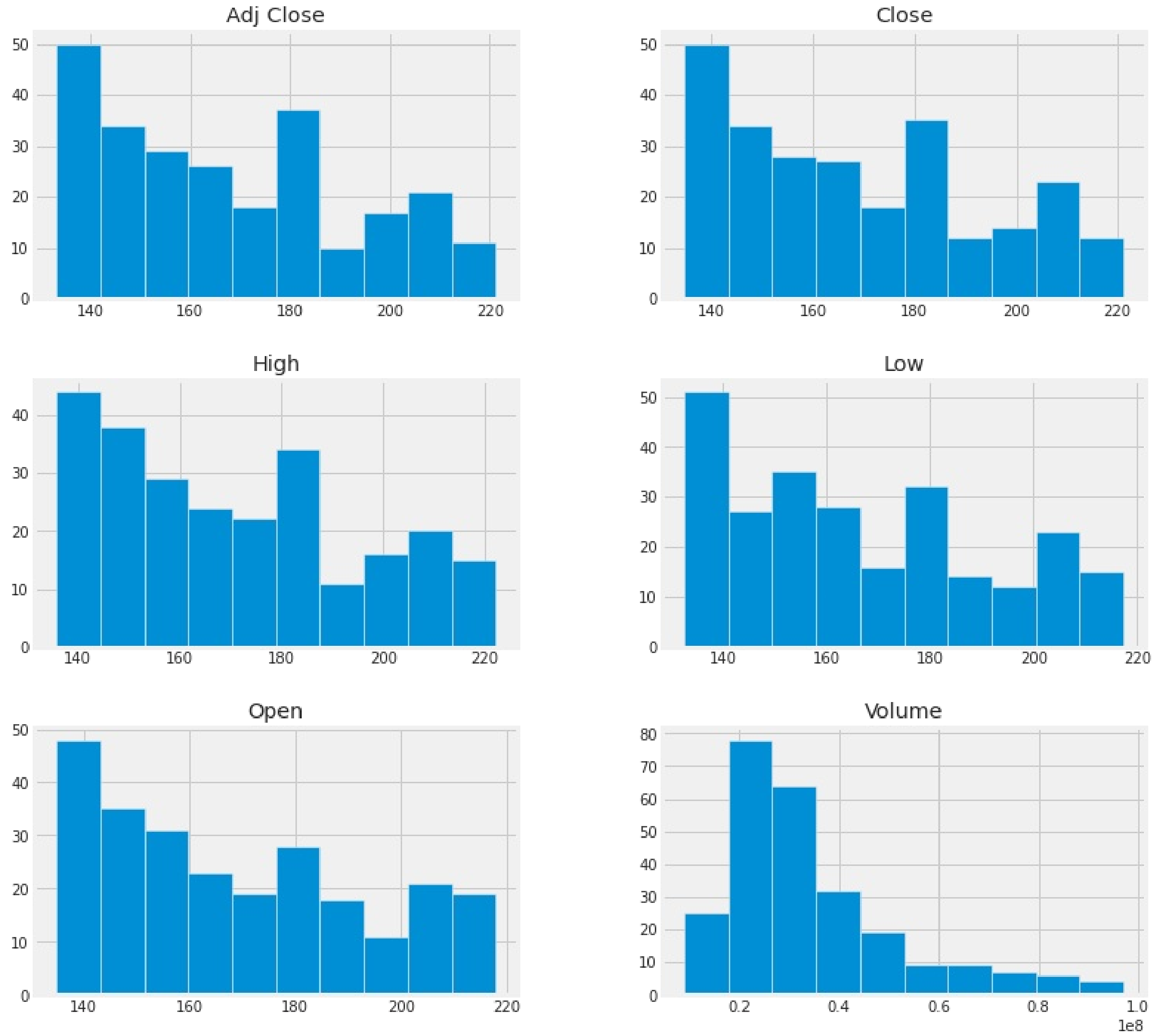



High
0
20
40
60
80
100
1750 2000 2250 2500 2750 3000 3250 3500
Low
0
10
20
30
40
50
60
70
80
1750 2000 2250 2500 2750 3000 3250
Open
0
10
20
30
40
50
60
70
80
1750 2000 2250 2500 2750 3000 3250
Volume
0
20
40
60
80
100
0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6
1e7


Adj Close
0
10
20
30
40
50
60
200 250 300 350 400 450 500
Close
0
10
20
30
40
50
200 250 300 350 400 450 500
High
0
10
20
30
40
50
60
200 250 300 350 400 450 500
Low
0
10
20
30
40
50
200 250 300 350 400 450 500
Open
0
10
20
30
40
50
200 250 300 350 400 450 500
Volume
0
10
20
30
40
50
60
70
80
0.2 0.4 0.6 0.8 1.0
1e8


Adj Close
10
20
30
40
Close
10
20
30
40

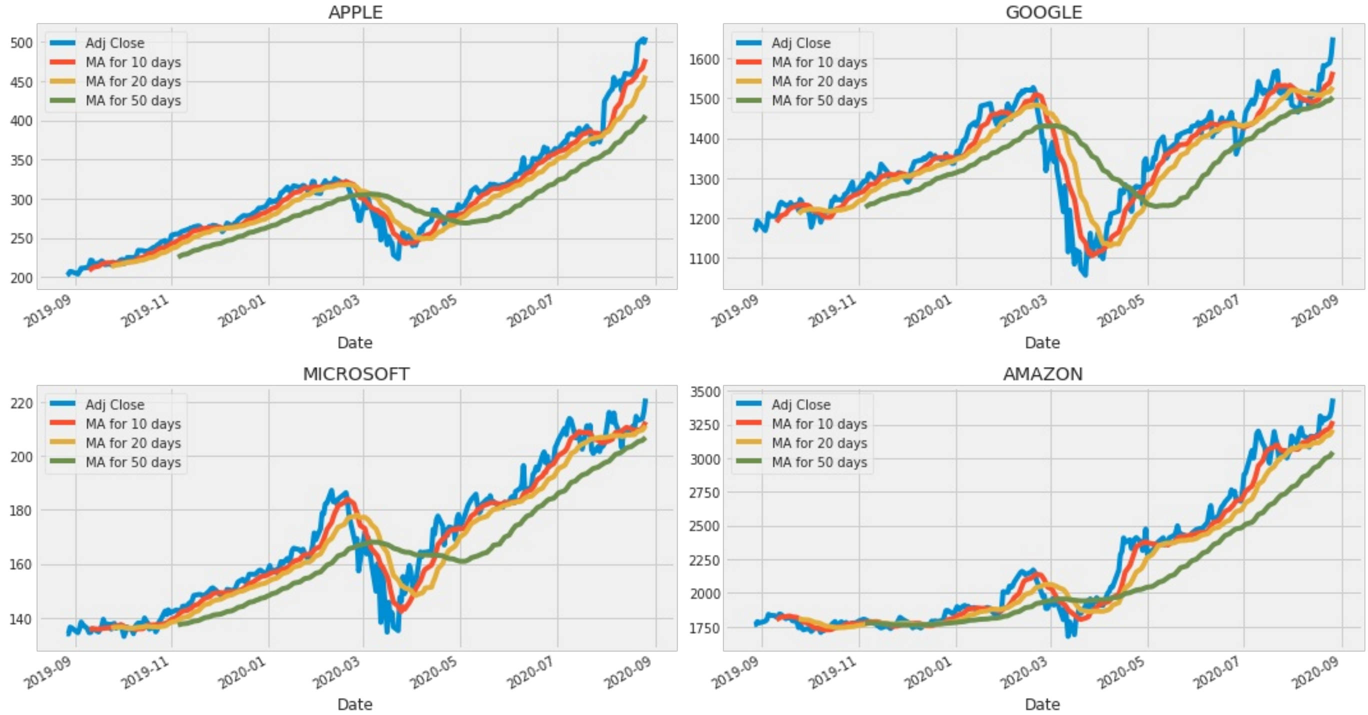




In [13]:

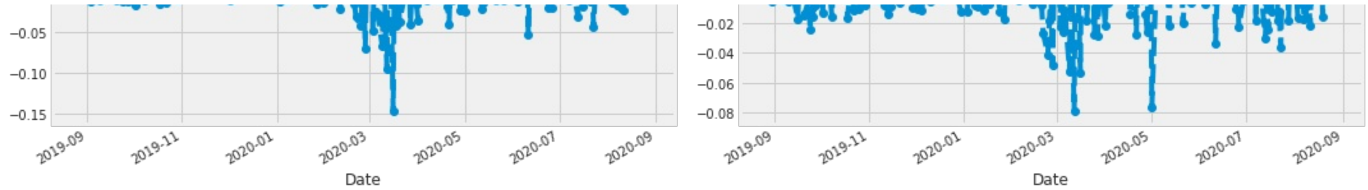
```
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(15)

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()
```



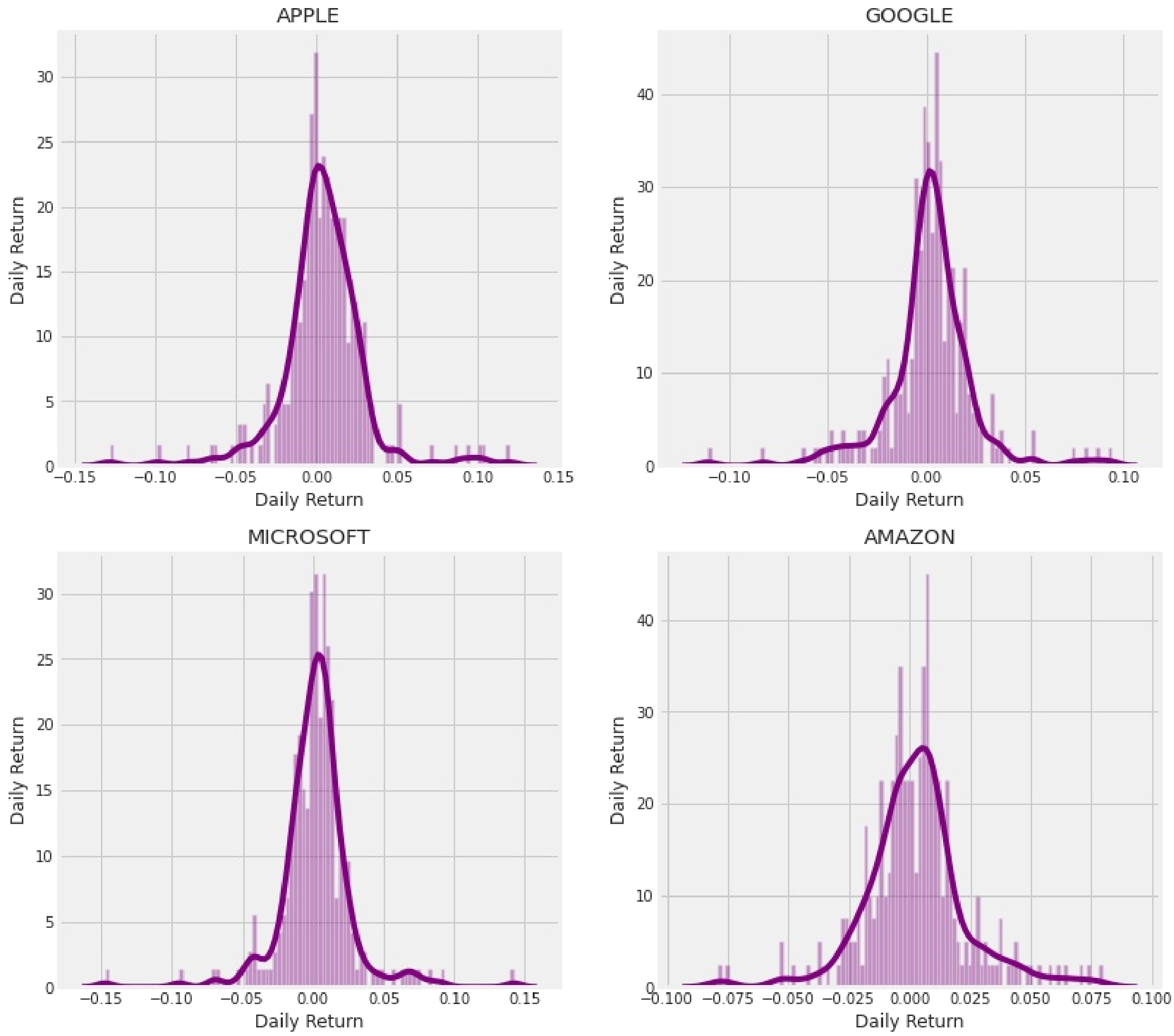
In [14]:

```
# 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(8)
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()
```



In [15]:

```
# Note the use of dropna() here, otherwise the NaN values can't be read by seaborn
plt.figure(figsize=(12, 12))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    sns.distplot(company['Daily Return'].dropna(), bins=100, color='purple')
    plt.ylabel('Daily Return')
    plt.title(f'{company_name[i - 1]}')
# Could have also done:
#AAPL['Daily Return'].hist()
```



In [16]:

```
# Grab all the closing prices for the tech stock list into one DataFrame
closing_df = DataReader(tech_list, 'yahoo', start, end)['Adj Close']

# Let's take a quick look
closing_df.head()
```

Out[16]:

Symbols	AAPL	GOOG	MSFT	AMZN
Date				
2019-08-27	202.154251	1167.839966	134.212051	1761.829956
2019-08-28	203.510803	1171.020020	134.034073	1764.250000
2019-08-29	206.956604	1192.849976	136.565262	1786.400024
2019-08-30	206.689255	1188.099976	136.308212	1776.290039
2019-09-03	203.679108	1168.390015	134.508682	1789.839966

In [17]:

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

Out[17]:

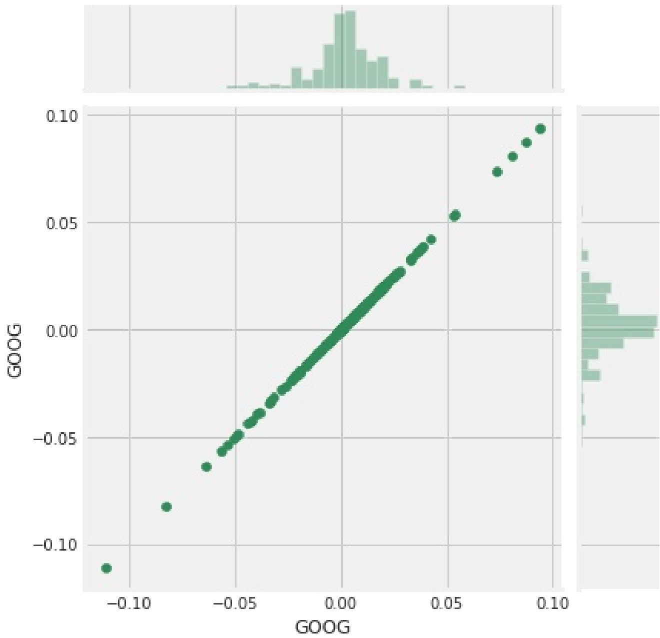
Symbols	AAPL	GOOG	MSFT	AMZN
Date				
2019-08-27	NaN	NaN	NaN	NaN
2019-08-28	0.006710	0.002723	-0.001326	0.001374
2019-08-29	0.016932	0.018642	0.018885	0.012555
2019-08-30	-0.001292	-0.003982	-0.001882	-0.005659
2019-09-03	-0.014564	-0.016589	-0.013202	0.007628

In [18]:

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

Out[18]:

```
<seaborn.axisgrid.JointGrid at 0x7f96612f06a0>
```

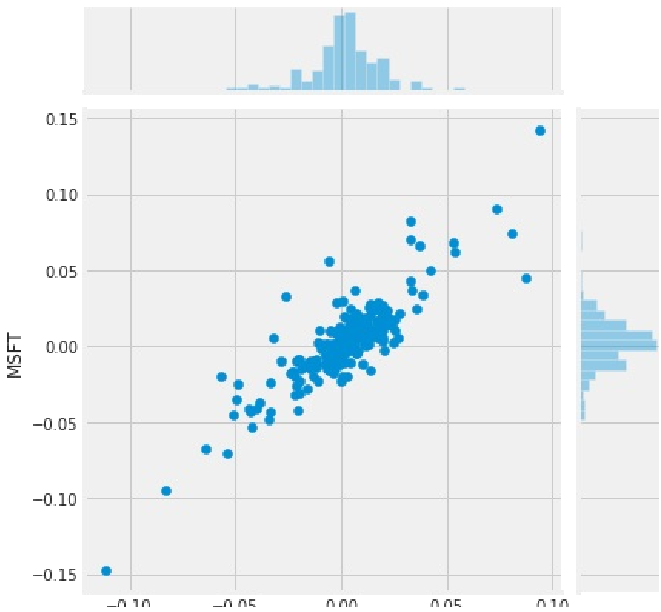


In [19]:

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

Out[19]:

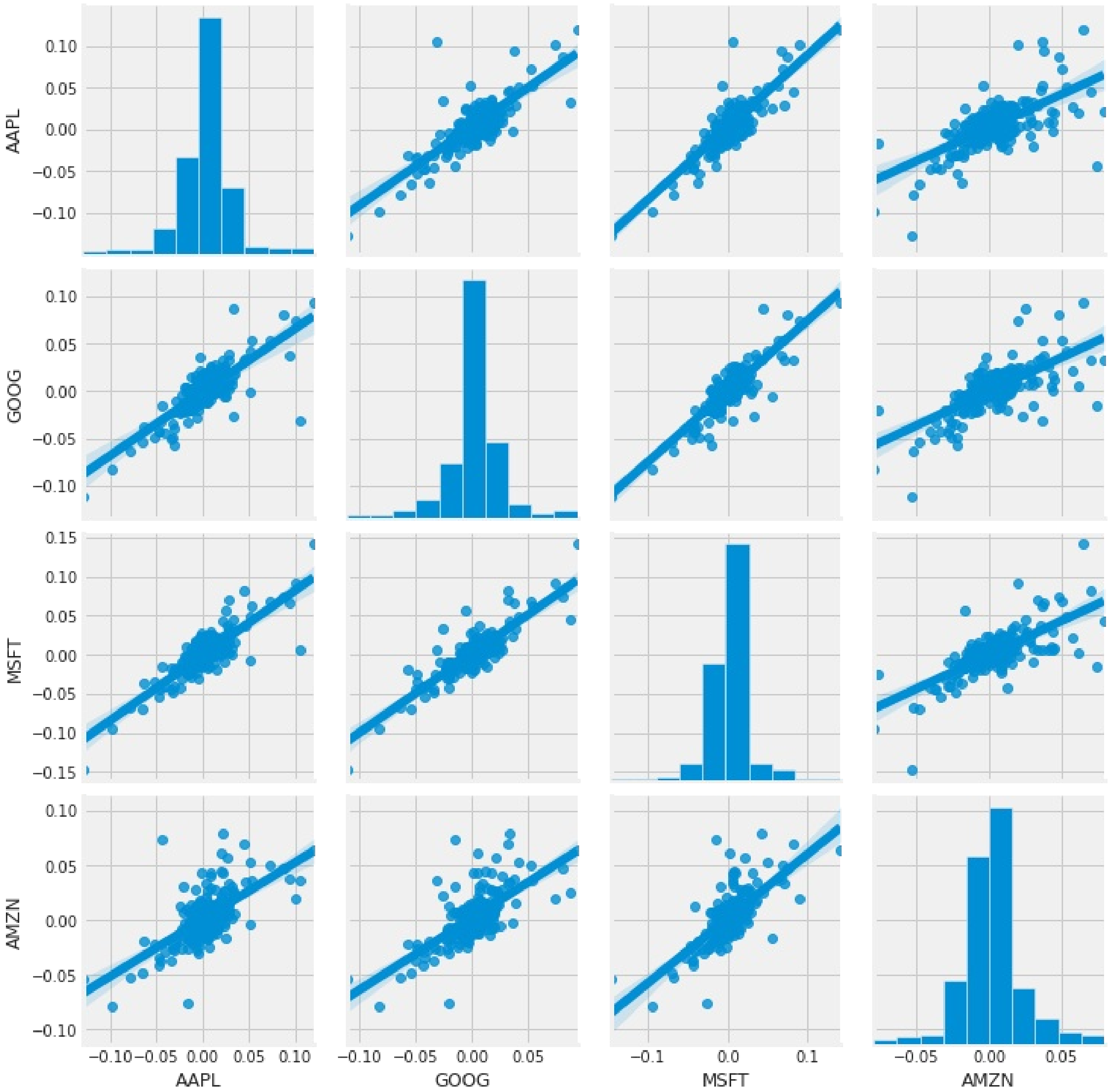
```
<seaborn.axisgrid.JointGrid at 0x7f96612f9748>
```



In [20]:

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

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

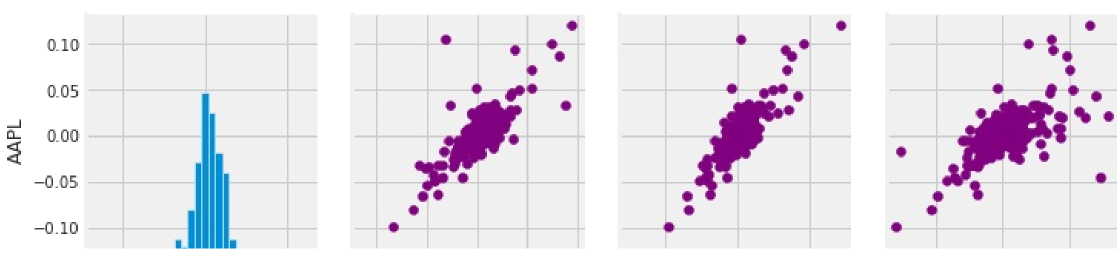
Out[20]: <seaborn.axisgrid.PairGrid at 0x7f96614ada20>

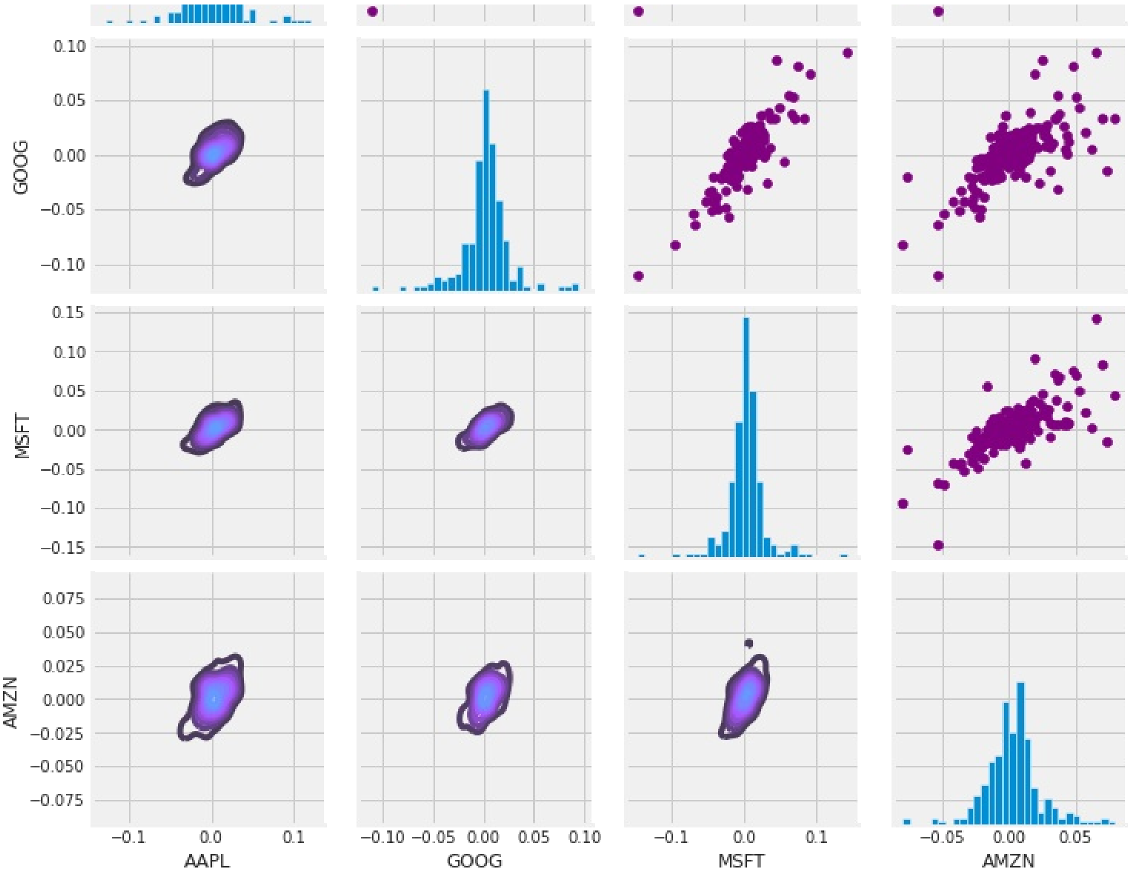


In [21]:

```
# 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)
```

Out[21]: <seaborn.axisgrid.PairGrid at 0x7f9660dd5e10>

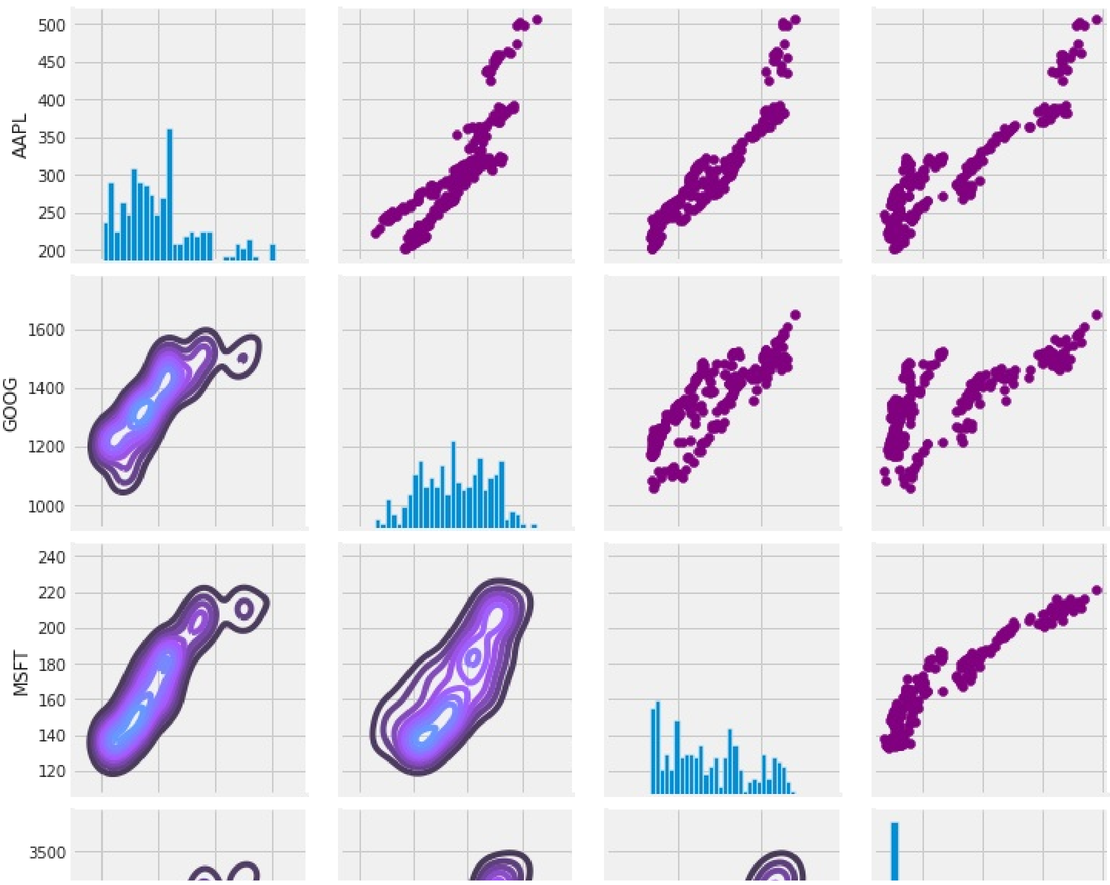


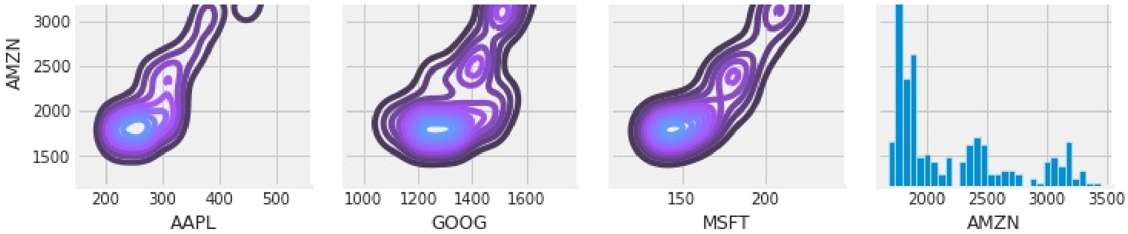


In [22]:

```
# 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)
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x7f965c929828>





In [23]:

```python
# Let's go ahead and use sebron for a quick correlation plot for the daily returns
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f965c1d22e8>

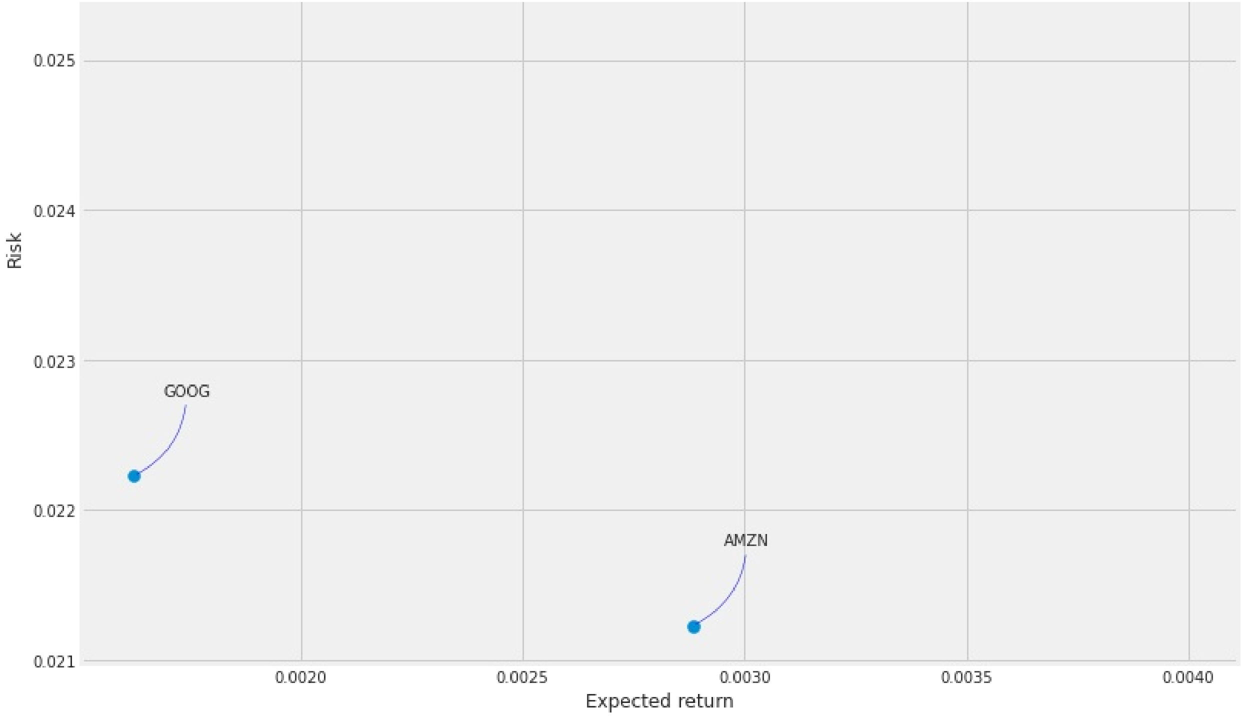
In [24]:

```python
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f965c0c7e10>

In [25]:

```python
# Let's start by defining a new DataFrame as a clenaed version of the oriignal tech_rets DataFrame
rets = tech_rets.dropna()

area = np.pi*20

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



In [26]:

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

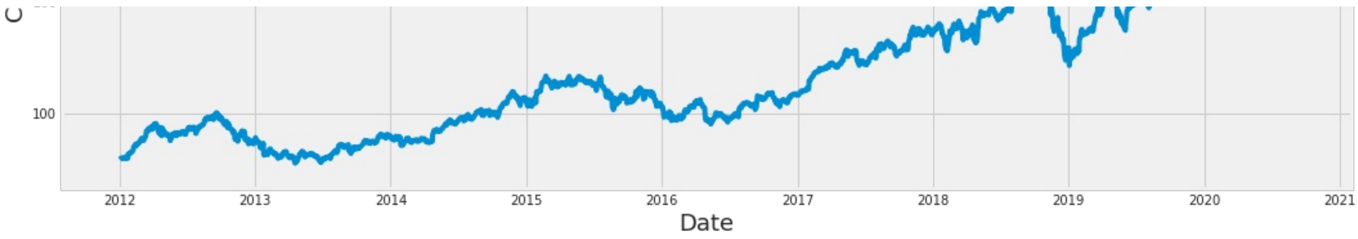
Out[26]:

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	58.928570	58.428570	58.485714	58.747143	75555200.0	50.765709
2012-01-04	59.240002	58.468571	58.571430	59.062859	65005500.0	51.038536
2012-01-05	59.792858	58.952858	59.278572	59.718571	67817400.0	51.605175
2012-01-06	60.392857	59.888573	59.967144	60.342857	79573200.0	52.144630
2012-01-09	61.107143	60.192856	60.785713	60.247143	98506100.0	52.061932
...	...	...	...	...	...	...
2020-08-20	473.570007	462.929993	463.000000	473.100006	31726800.0	473.100006
2020-08-21	499.470001	477.000000	477.049988	497.480011	84513700.0	497.480011
2020-08-24	515.140015	495.750000	514.789978	503.429993	86484400.0	503.429993
2020-08-25	500.720001	492.209991	498.790009	499.299988	52776900.0	499.299988
2020-08-26	507.970001	500.329987	504.716003	506.089996	40755567.0	506.089996

2177 rows × 6 columns

In [27]:

```
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```



In [28]:

```
#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) * .8 ))

training_data_len
```

Out[28]:

```
1742
```

In [29]:

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

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

scaled_data
```

Out[29]:

```
array([[0.00656705],
       [0.00726817],
       [0.00872434],
       ...,
       [0.99409282],
       [0.98492114],
       [1.        ]])
```

In [30]:

```
#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()
```

```
[array([0.00656705, 0.00726817, 0.00872434, 0.01011072, 0.00989816,
       0.01037721, 0.0101583 , 0.00979029, 0.00928905, 0.01084039,
       0.01223946, 0.011808  , 0.0094445 , 0.01170013, 0.00947939,
       0.01780717, 0.01716316, 0.01800387, 0.0198217 , 0.02092255,
       0.02083055, 0.0204911 , 0.02193775, 0.02329874, 0.02484058,
       0.02733099, 0.03256242, 0.03264173, 0.03555408, 0.0377304 ,
       0.03399003, 0.03543035, 0.0354018 , 0.03944038, 0.03886615,
       0.03992893, 0.04183877, 0.04290156, 0.04596301, 0.04819327,
       0.04883727, 0.04906253, 0.04524921, 0.04432917, 0.0444656 ,
       0.0480505 , 0.04905936, 0.05122616, 0.05633387, 0.06314838,
       0.06187303, 0.0618762 , 0.06680308, 0.06834491, 0.06724723,
       0.06624473, 0.06520098, 0.0686685 , 0.07104788, 0.07204404])]
[0.06958217975378928]

[array([0.00656705, 0.00726817, 0.00872434, 0.01011072, 0.00989816,
       0.01037721, 0.0101583 , 0.00979029, 0.00928905, 0.01084039,
       0.01223946, 0.011808  , 0.0094445 , 0.01170013, 0.00947939,
       0.01780717, 0.01716316, 0.01800387, 0.0198217 , 0.02092255,
       0.02083055, 0.0204911 , 0.02193775, 0.02329874, 0.02484058,
       0.02733099, 0.03256242, 0.03264173, 0.03555408, 0.0377304 ,
       0.03399003, 0.03543035, 0.0354018 , 0.03944038, 0.03886615,
       0.03992893, 0.04183877, 0.04290156, 0.04596301, 0.04819327,
       0.04883727, 0.04906253, 0.04524921, 0.04432917, 0.0444656 ,
       0.0480505 , 0.04905936, 0.05122616, 0.05633387, 0.06314838,
       0.06187303, 0.0618762 , 0.06680308, 0.06834491, 0.06724723,
       0.06624473, 0.06520098, 0.0686685 , 0.07104788, 0.07204404]), array([0.00726817, 0.00872434, 0.01011072, 0
.00989816, 0.01037721,
```

```
       0.0101583 , 0.00979029, 0.00928905, 0.01084039, 0.01223946,
       0.011808  , 0.0094445 , 0.01170013, 0.00947939, 0.01780717,
       0.01716316, 0.01800387, 0.0198217 , 0.02092255, 0.02083055,
       0.0204911 , 0.02193775, 0.02329874, 0.02484058, 0.02733099,
       0.03256242, 0.03264173, 0.03555408, 0.0377304 , 0.03399003,
       0.03543035, 0.0354018 , 0.03944038, 0.03886615, 0.03992893,
       0.04183877, 0.04290156, 0.04596301, 0.04819327, 0.04883727,
       0.04906253, 0.04524921, 0.04432917, 0.0444656 , 0.0480505 ,
       0.04905936, 0.05122616, 0.05633387, 0.06314838, 0.06187303,
       0.0618762 , 0.06680308, 0.06834491, 0.06724723, 0.06624473,
       0.06520098, 0.0686685 , 0.07104788, 0.07204404, 0.06958218])]
[0.06958217975378928, 0.06631135001976646]
```

In [31]:

```
# 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
```

In [32]:

```
from keras.models import Sequential
from keras.layers import Dense, LSTM

#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(50, 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)
```

```
1682/1682 [==============================] - 38s 22ms/step - loss: 3.1311e-04
```

Out[32]:

```
<tensorflow.python.keras.callbacks.History at 0x7f96249d1f28>
```

In [35]:

```
#Create a new array containing scaled values from index 1543 to 2002
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
```

Out[35]:

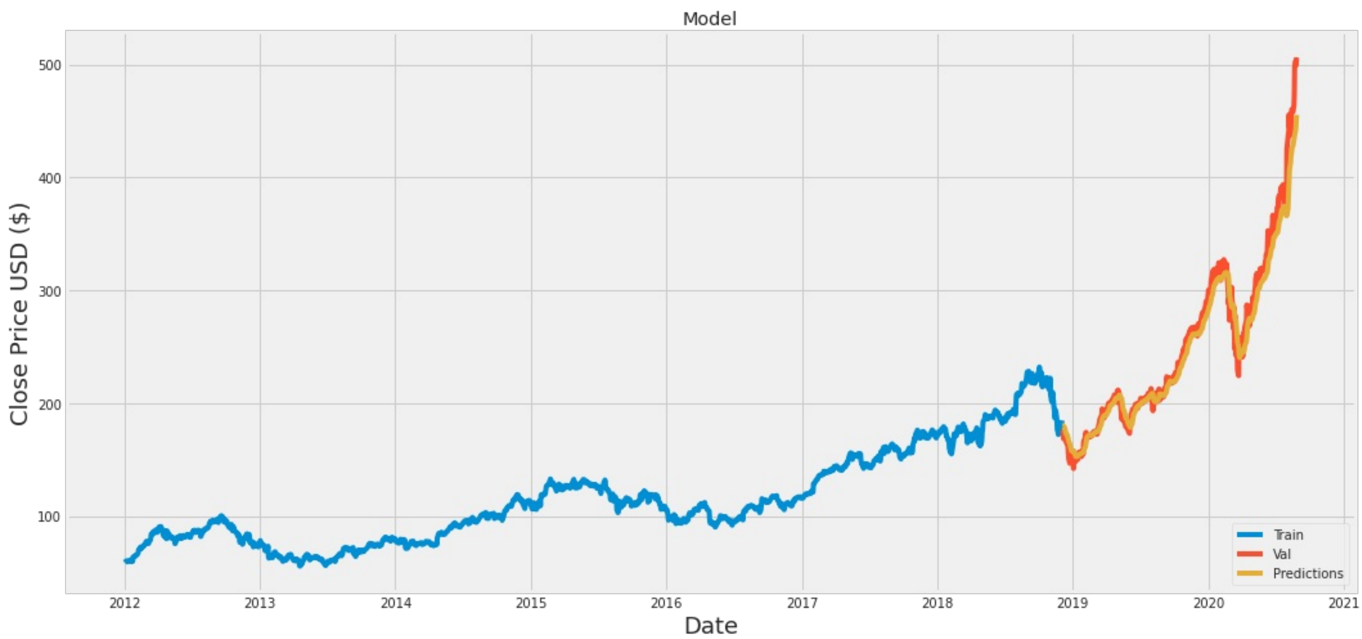
```
13.863484023436264
```

In [36]:

```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,8))
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()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: 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#retur
ning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```



In [37]:

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

Out[37]:

	Close	Predictions
Date		
2018-12-04	176.690002	179.923920
2018-12-06	174.720001	180.091034
2018-12-07	168.490005	179.702438
2018-12-10	169.600006	178.356049
2018-12-11	168.630005	176.818344
...	...	...
2020-08-20	473.100006	434.715576
2020-08-21	497.480011	437.345734
2020-08-24	503.429993	442.873230
2020-08-25	499.299988	449.547791
2020-08-26	506.089996	455.354279

435 rows × 2 columns

In [ ]: