DSC350 Exercise 6.2

Exercise #1

Selecting all earthquakes in Japan with a mb magnitude => 4.9

In [1]:
```
import pandas as pd
```

Load the dataset

In [2]:
```
df = pd.read_csv("earthquakes.csv")
```

Filter earthquakes in Japan with mb magnitude ≥ 4.9

In [3]:
```
japan_earthquakes = df[(df["parsed_place"].str.contains("Japan", na=False)) &
                       (df["magType"] == "mb") &
                       (df["mag"] >= 4.9)]
```

Display the filtered results

In [5]:
```
print(japan_earthquakes)
```

```
      mag magType           time                          place  tsunami  \
1563  4.9      mb  1538977532250  293km ESE of Iwo Jima, Japan        0
2576  5.4      mb  1538697528010    37km E of Tomakomai, Japan        0
3072  4.9      mb  1538579732490     15km ENE of Hasaki, Japan        0
3632  4.9      mb  1538450871260    53km ESE of Hitachi, Japan        0

     parsed_place
1563        Japan
2576        Japan
3072        Japan
3632        Japan
```

Observations:

- **Magnitude:** The earthquakes meet the required threshold, with values of **4.9 to 5.4**.
- **Locations:** These occurred in **Iwo Jima, Tomakomai, Hasaki, and Hitachi**, representing different regions in Japan.
- **Tsunami Impact:** All earthquakes have a **tsunami value of 0**, meaning no tsunami was reported.
- **Time:** The timestamps (Unix format) suggest they happened at different moments.

Insights:

- Since these earthquakes didn’t trigger a tsunami, it could indicate they were **deep-focus earthquakes** or lacked the necessary underwater displacement.

- The **variation in locations** suggests seismic activity across multiple fault lines rather than a single concentrated seismic event.

Exercise 2

Create bins for each full number of earthquake magnitude using the ml magnitude type and count how many earthquakes fall into each bin.

Load the dataset

In [6]:
```
df = pd.read_csv("earthquakes.csv")
```

Filter only earthquakes with the 'ml' magnitude type

In [7]:
```
ml_earthquakes = df[df["magType"] == "ml"]
```

Define bins ((0,1], (1,2], ..., (9,10])

In [8]:
```
bins = list(range(0, 11))  # Creating bins from 0 to 10
labels = [f"({i},{i+1}]" for i in range(0, 10)]  # Label bins as (0,1], (1,2], etc.
```

Categorize magnitudes into bins

In [9]:
```
ml_earthquakes["magnitude_bin"] = pd.cut(ml_earthquakes["mag"], bins=bins, labels=l
```

```
C:\Users\lisah\AppData\Local\Temp\ipykernel_66336\1221612971.py:1: SettingWithCopyWa
rning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ml_earthquakes["magnitude_bin"] = pd.cut(ml_earthquakes["mag"], bins=bins, labels=
labels, right=True)
```

Fix Warning

In [10]:
```
ml_earthquakes = df[df["magType"] == "ml"].copy()  # Make an explicit copy
```

Now apply the binning without warnings

In [11]:
```
ml_earthquakes["magnitude_bin"] = pd.cut(ml_earthquakes["mag"], bins=bins, labels=l
```

Why This Fix Works

- .copy() ensures that ml_earthquakes is a completely independent DataFrame instead of a view tied to df.
- This prevents unintended modifications to the original dataset.

In [12]:
```
ml_earthquakes["magnitude_bin"] = pd.cut(ml_earthquakes["mag"], bins=bins, labels=l
```

Count earthquakes in each bin

In [13]:
```
magnitude_counts = ml_earthquakes["magnitude_bin"].value_counts().sort_index()
```

Display results

In [14]:
```
print(magnitude_counts)
```

```
magnitude_bin
(0,1]     2207
(1,2]     3105
(2,3]      862
(3,4]      122
(4,5]        2
(5,6]        1
(6,7]        0
(7,8]        0
(8,9]        0
(9,10]       0
Name: count, dtype: int64
```

Observations:

1. **Lower Magnitudes Dominate:**
   - Most earthquakes fall within the **(1,2] bin (3,105 occurrences)**, followed closely by the **(0,1] bin (2,207 occurrences)**.
   - This is expected, as smaller earthquakes are far more frequent than larger ones.
2. **A Sharp Decline After Magnitude 3:**
   - As magnitude increases, the frequency of earthquakes drops dramatically.
   - Only **2 earthquakes** fall in the **(4,5] range**, and just **1 in the (5,6] bin**.
3. **No Major Earthquakes Beyond Magnitude 5:**
   - The bins from **(6,7] onward contain 0 earthquakes** in the dataset.
   - This suggests that, for recorded **ml** earthquakes, the dataset may be incomplete beyond magnitude 5 or such high-magnitude **ml** earthquakes are rare.

Possible Insights:

- **Seismic Activity Trends:** The distribution suggests that **ml** magnitudes primarily represent **low-intensity seismic events**, possibly used for measuring smaller, local quakes rather than major tectonic movements.
- **Dataset Limitations:** If we want deeper insights into higher-magnitude earthquakes, we might need other magnitude types (mb, Ms, Mw), which could be more suitable for capturing **stronger seismic events**.

A quick visualization because it is my favorite!

In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Set plot style

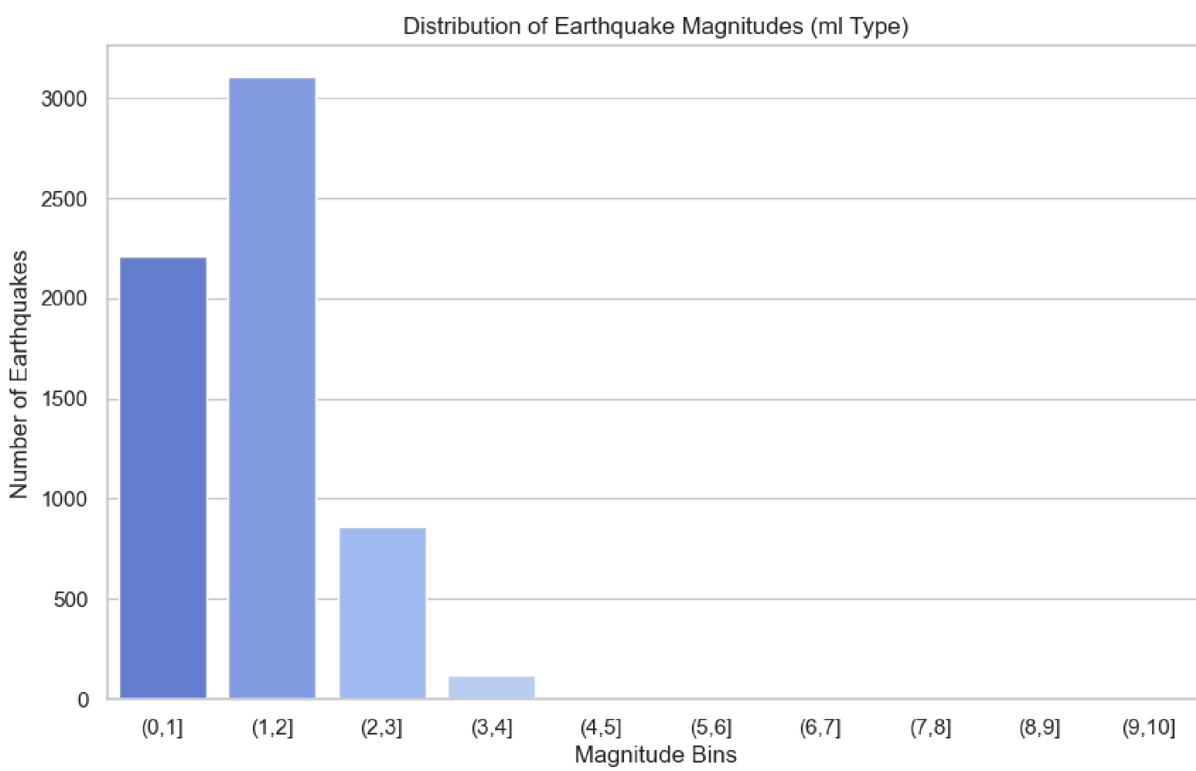
In [16]:
```
sns.set_theme(style="whitegrid")
```

Create the bar plot

In [20]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x=magnitude_counts.index, y=magnitude_counts.values, hue=magnitude_coun

# Add labels and title
plt.xlabel("Magnitude Bins")
plt.ylabel("Number of Earthquakes")
plt.title("Distribution of Earthquake Magnitudes (ml Type)")

# Show the plot
plt.show()
```



Exercise 3

grouping the data by ticker and resampling it to a monthly frequency.

3a.

Mean of the opening price.

Load the FAANG dataset

In [21]:
```
df_faang = pd.read_csv("faang.csv", parse_dates=["date"])
```

Set the date column as the index for resampling

In [22]:
```
df_faang.set_index("date", inplace=True)
```

Resample to monthly frequency and calculate the mean opening price for each ticker

In [25]:
```
monthly_open_mean = df_faang.groupby("ticker").resample("ME")["open"].mean()
```

Display the results

In [26]:
```
print(monthly_open_mean)
```

```
ticker  date
AAPL    2018-01-31      43.505357
        2018-02-28      41.819079
        2018-03-31      43.761786
        2018-04-30      42.441310
        2018-05-31      46.239091
        2018-06-30      47.180119
        2018-07-31      47.549048
        2018-08-31      53.121739
        2018-09-30      55.582763
        2018-10-31      55.300000
        2018-11-30      47.954881
        2018-12-31      41.310789
AMZN    2018-01-31    1301.377151
        2018-02-28    1447.113159
        2018-03-31    1542.160464
        2018-04-30    1475.841902
        2018-05-31    1590.474543
        2018-06-30    1699.088582
        2018-07-31    1786.305716
        2018-08-31    1891.957833
        2018-09-30    1969.239476
        2018-10-31    1799.630865
        2018-11-30    1622.323806
        2018-12-31    1572.922100
FB      2018-01-31     184.584284
        2018-02-28     180.721578
        2018-03-31     173.449524
        2018-04-30     164.163332
        2018-05-31     181.910909
        2018-06-30     194.974763
        2018-07-31     199.332381
        2018-08-31     177.598695
        2018-09-30     164.233158
        2018-10-31     154.873479
        2018-11-30     141.762857
        2018-12-31     137.529475
GOOG    2018-01-31    1127.200945
        2018-02-28    1088.629472
        2018-03-31    1096.108085
        2018-04-30    1038.415237
        2018-05-31    1064.021376
        2018-06-30    1136.396182
        2018-07-31    1183.464280
        2018-08-31    1226.156951
        2018-09-30    1176.878424
        2018-10-31    1116.082172
        2018-11-30    1054.971424
        2018-12-31    1042.619998
NFLX    2018-01-31     231.269525
        2018-02-28     270.873158
        2018-03-31     312.712859
        2018-04-30     309.129524
        2018-05-31     329.779541
        2018-06-30     384.557143
        2018-07-31     380.969526
```

```
        2018-08-31     345.410001
        2018-09-30     363.326843
        2018-10-31     340.025218
        2018-11-30     290.643335
        2018-12-31     266.309474
Name: open, dtype: float64
```

Observations:

AAPL: Started the year around \$ 43.50, peaked at \$ 55.58 in September, and dropped back to \$ 41.31 in December, suggesting late-year volatility.

AMZN: Showed consistent growth, climbing from \$1301 in January to a peak of \$ 1969 in September, before declining in Q4.

FB: Dropped notably after July 2018, aligning with major industry events like privacy concerns and disappointing earnings.

GOOG: Displayed a mid-year rise, peaking in August but dipping in the final months.

NFLX: Surged through the first half of 2018, hitting \$384.56 in June, then experiencing a downward correction.

Insights:

- **Tech stocks saw strong growth mid-year but weakened in Q4**, likely influenced by macroeconomic conditions or investor sentiment shifts.
- **Amazon showed the highest opening price variation**, reflecting its aggressive growth trajectory.
- **Netflix had notable volatility**, with large swings between months.

Line Chart (Best for Trends Over Time)

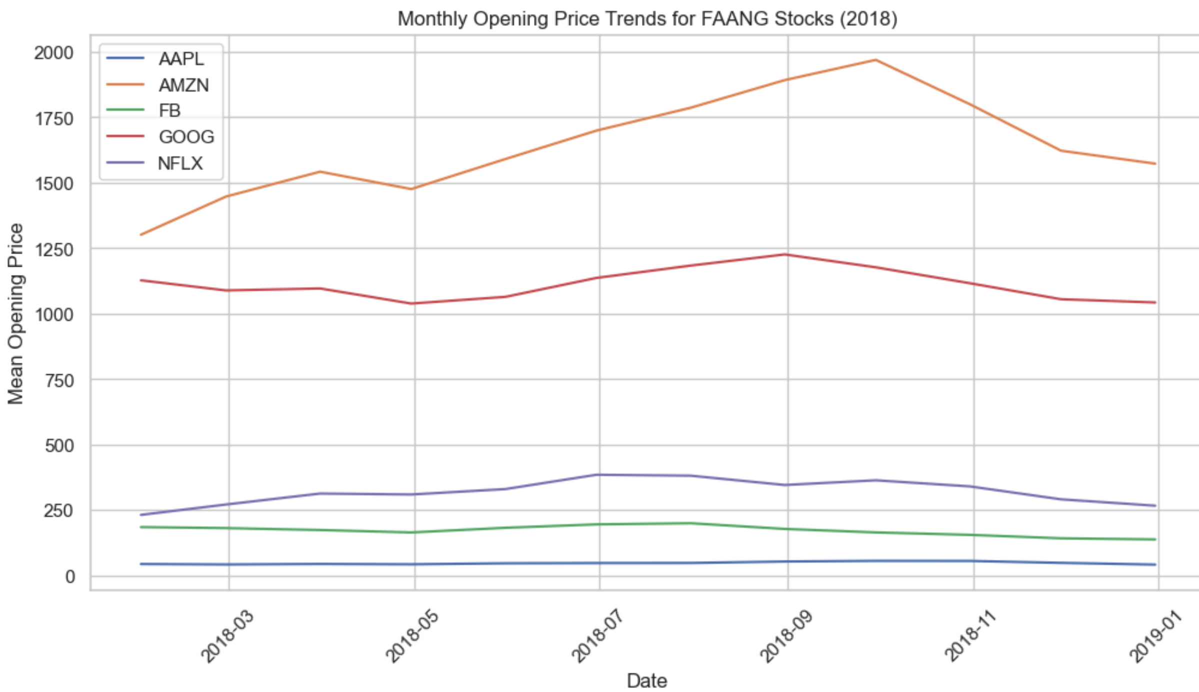
Pivot the data for visualization

In [27]:
```
pivot_data = monthly_open_mean.unstack(level=0)
```

In [28]:
```
# Plot the line chart
plt.figure(figsize=(12, 6))
for ticker in pivot_data.columns:
    plt.plot(pivot_data.index, pivot_data[ticker], label=ticker)

# Formatting
plt.xlabel("Date")
plt.ylabel("Mean Opening Price")
plt.title("Monthly Opening Price Trends for FAANG Stocks (2018)")
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
```

```
# Show the plot
plt.show()
```

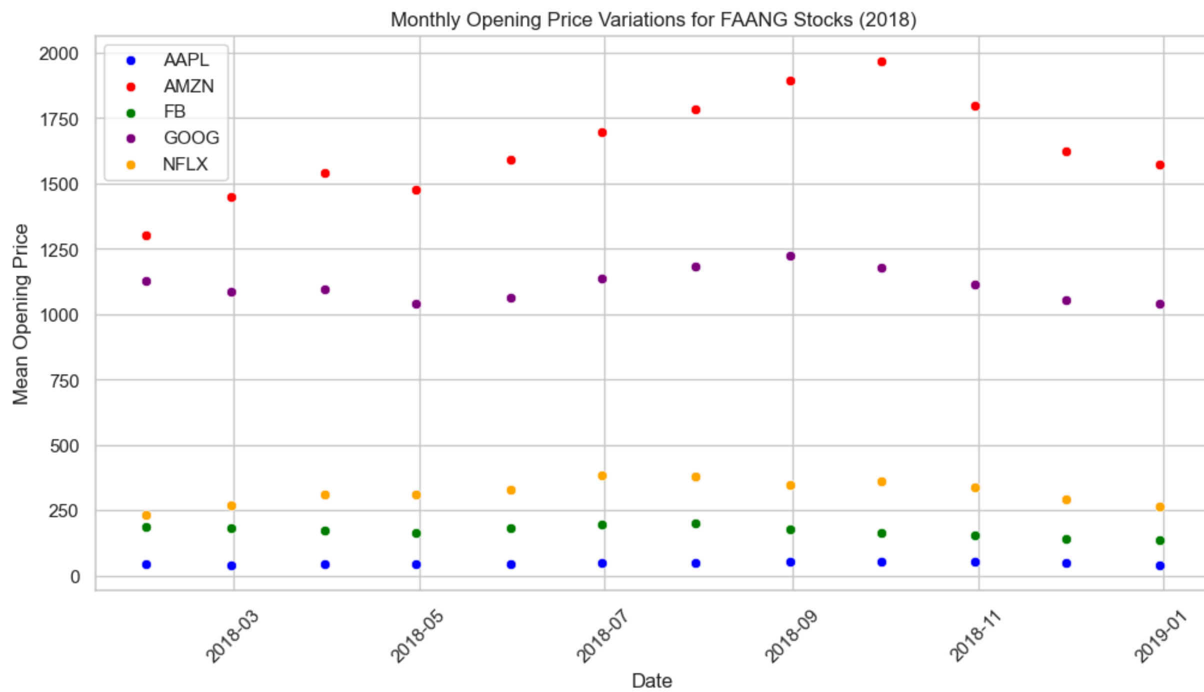


Scatter Plot (Best for Spotting Outliers & Variations)

In [29]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x=pivot_data.index, y=pivot_data["AAPL"], label="AAPL", color="blue
sns.scatterplot(x=pivot_data.index, y=pivot_data["AMZN"], label="AMZN", color="red"
sns.scatterplot(x=pivot_data.index, y=pivot_data["FB"], label="FB", color="green")
sns.scatterplot(x=pivot_data.index, y=pivot_data["GOOG"], label="GOOG", color="purp
sns.scatterplot(x=pivot_data.index, y=pivot_data["NFLX"], label="NFLX", color="oran

# Formatting
plt.xlabel("Date")
plt.ylabel("Mean Opening Price")
plt.title("Monthly Opening Price Variations for FAANG Stocks (2018)")
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Exercise 3b

calculate the maximum of the high price for each ticker at a monthly frequency.

Resample to monthly frequency and compute the maximum high price for each ticker

In [30]:
```
monthly_high_max = df_faang.groupby("ticker").resample("ME")["high"].max()
```

Display results

In [31]:
```
print(monthly_high_max)
```

```
ticker  date
AAPL    2018-01-31      45.025002
        2018-02-28      45.154999
        2018-03-31      45.875000
        2018-04-30      44.735001
        2018-05-31      47.592499
        2018-06-30      48.549999
        2018-07-31      48.990002
        2018-08-31      57.217499
        2018-09-30      57.417500
        2018-10-31      58.367500
        2018-11-30      55.590000
        2018-12-31      46.235001
AMZN    2018-01-31    1472.579956
        2018-02-28    1528.699951
        2018-03-31    1617.540039
        2018-04-30    1638.099976
        2018-05-31    1635.000000
        2018-06-30    1763.099976
        2018-07-31    1880.050049
        2018-08-31    2025.569946
        2018-09-30    2050.500000
        2018-10-31    2033.189941
        2018-11-30    1784.000000
        2018-12-31    1778.339966
FB      2018-01-31     190.660004
        2018-02-28     195.320007
        2018-03-31     186.100006
        2018-04-30     177.100006
        2018-05-31     192.720001
        2018-06-30     203.550003
        2018-07-31     218.619995
        2018-08-31     188.300003
        2018-09-30     173.889999
        2018-10-31     165.880005
        2018-11-30     154.130005
        2018-12-31     147.190002
GOOG    2018-01-31    1186.890015
        2018-02-28    1174.000000
        2018-03-31    1177.050049
        2018-04-30    1094.165039
        2018-05-31    1110.750000
        2018-06-30    1186.286011
        2018-07-31    1273.890015
        2018-08-31    1256.500000
        2018-09-30    1212.989990
        2018-10-31    1209.959961
        2018-11-30    1095.569946
        2018-12-31    1124.650024
NFLX    2018-01-31     286.809998
        2018-02-28     297.359985
        2018-03-31     333.980011
        2018-04-30     338.820007
        2018-05-31     356.100006
        2018-06-30     423.209991
        2018-07-31     419.769989
```

```
    2018-08-31     376.809998
    2018-09-30     383.200012
    2018-10-31     386.799988
    2018-11-30     332.049988
    2018-12-31     298.720001
Name: high, dtype: float64
```

Observations:

AAPL: Peaked at $58.37 in October, followed by a decline toward $46.23 in December—suggesting market volatility.

AMZN: Hit its highest price at $2050.50 in September, showing strong upward momentum before dropping in Q4.

FB: Reached its peak at $218.61 in July, but saw a steady decline afterward, likely tied to company-specific events.

GOOG: Showed growth until July ($1273.89), but experienced fluctuations in the following months.

NFLX: Spiked at $423.21 in June, showing strong bullish trends before correcting later.

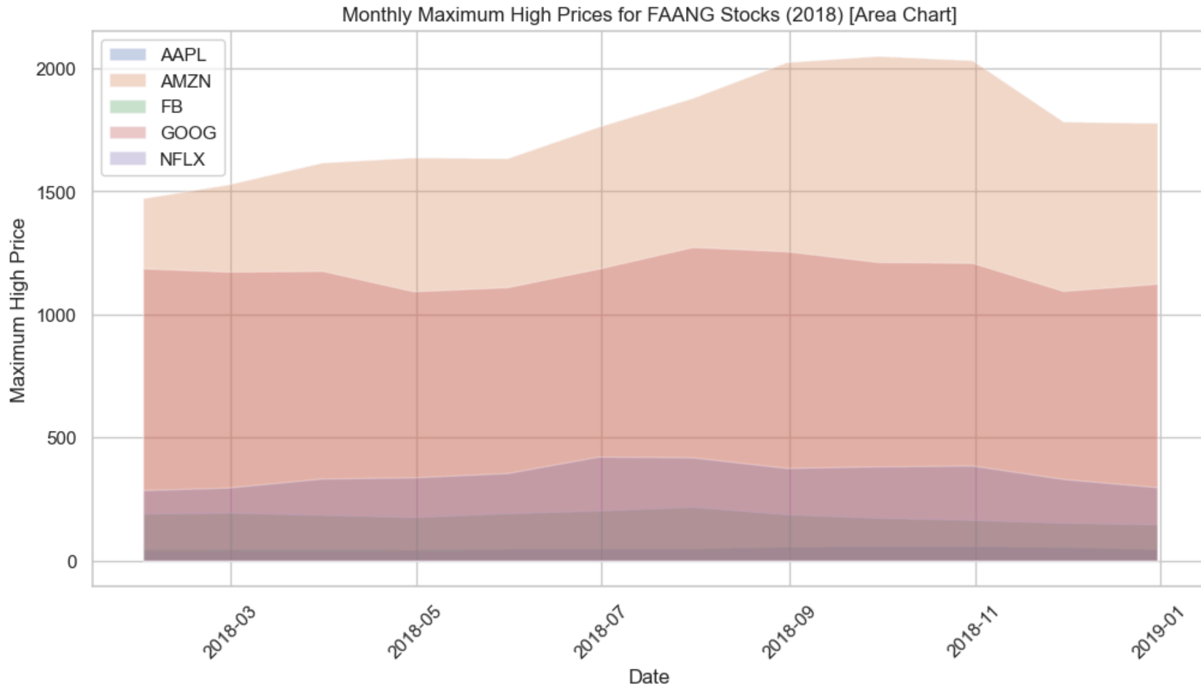
Insights:

- **Amazon's September peak** aligns with its strong business expansion and investor confidence mid-year.
- **Facebook's July peak** coincides with its pre-earnings period before disappointing reports hit the stock.
- **Tech stocks generally peaked mid-year** but saw corrections in Q4, possibly due to broader market trends.

Area Chart (Smooth Visual Trend)

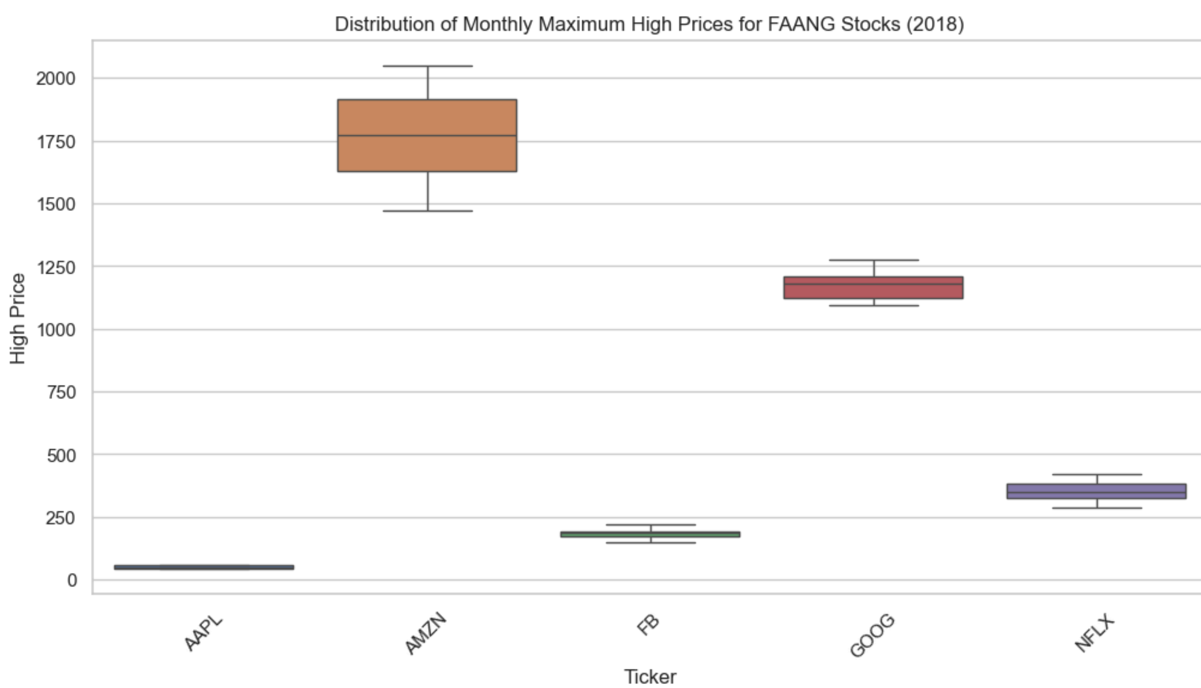
In [35]:
```python
plt.figure(figsize=(12, 6))
for ticker in pivot_high_data.columns:
    plt.fill_between(pivot_high_data.index, pivot_high_data[ticker], alpha=0.3, lab

plt.xlabel("Date")
plt.ylabel("Maximum High Price")
plt.title("Monthly Maximum High Prices for FAANG Stocks (2018) [Area Chart]")
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Box Plot (Captures Price Variability)

In [36]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=pivot_high_data)
plt.xlabel("Ticker")
plt.ylabel("High Price")
plt.title("Distribution of Monthly Maximum High Prices for FAANG Stocks (2018)")
plt.xticks(rotation=45)
plt.show()
```



Exercise 3c

Minimum of the Low Price for each FAANG stock at a monthly frequency

Minimum of the Low Price

Resample to monthly frequency and compute the minimum low price for each ticker

In [38]:
```
monthly_low_min = df_faang.groupby("ticker").resample("ME")["low"].min()
```

Display results

In [39]:
```
print(monthly_low_min)
```

```
ticker  date
AAPL    2018-01-31      41.174999
        2018-02-28      37.560001
        2018-03-31      41.235001
        2018-04-30      40.157501
        2018-05-31      41.317501
        2018-06-30      45.182499
        2018-07-31      45.855000
        2018-08-31      49.327499
        2018-09-30      53.825001
        2018-10-31      51.522499
        2018-11-30      42.564999
        2018-12-31      36.647499
AMZN    2018-01-31    1170.510010
        2018-02-28    1265.930054
        2018-03-31    1365.199951
        2018-04-30    1352.880005
        2018-05-31    1546.020020
        2018-06-30    1635.089966
        2018-07-31    1678.060059
        2018-08-31    1776.020020
        2018-09-30    1865.000000
        2018-10-31    1476.359985
        2018-11-30    1420.000000
        2018-12-31    1307.000000
FB      2018-01-31     175.800003
        2018-02-28     167.179993
        2018-03-31     149.020004
        2018-04-30     150.509995
        2018-05-31     170.229996
        2018-06-30     186.429993
        2018-07-31     166.559998
        2018-08-31     170.270004
        2018-09-30     158.869995
        2018-10-31     139.029999
        2018-11-30     126.849998
        2018-12-31     123.019997
GOOG    2018-01-31    1045.229980
        2018-02-28     992.559998
        2018-03-31     980.640015
        2018-04-30     990.369995
        2018-05-31    1006.289978
        2018-06-30    1096.010010
        2018-07-31    1093.800049
        2018-08-31    1188.239990
        2018-09-30    1146.910034
        2018-10-31     995.830017
        2018-11-30     996.020020
        2018-12-31     970.109985
NFLX    2018-01-31     195.419998
        2018-02-28     236.110001
        2018-03-31     275.899994
        2018-04-30     271.220001
        2018-05-31     305.730011
        2018-06-30     352.820007
        2018-07-31     328.000000
```

```
        2018-08-31     310.929993
        2018-09-30     335.829987
        2018-10-31     271.209991
        2018-11-30     250.000000
        2018-12-31     231.229996
Name: low, dtype: float64
```

Insights:

- Q4 volatility is evident in almost all stocks, especially Amazon and Facebook, suggesting industry-wide or macroeconomic factors at play.
- Apple saw a sharp decline toward the year-end, likely tied to concerns about iPhone demand.
- Netflix had a steadier trajectory, despite fluctuations mid-year.

Heatmap for Minimum Low Prices

In [40]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
```

Pivot the data for visualization

In [41]:
```python
pivot_low_data = monthly_low_min.unstack(level=0)
```

Create the heatmap

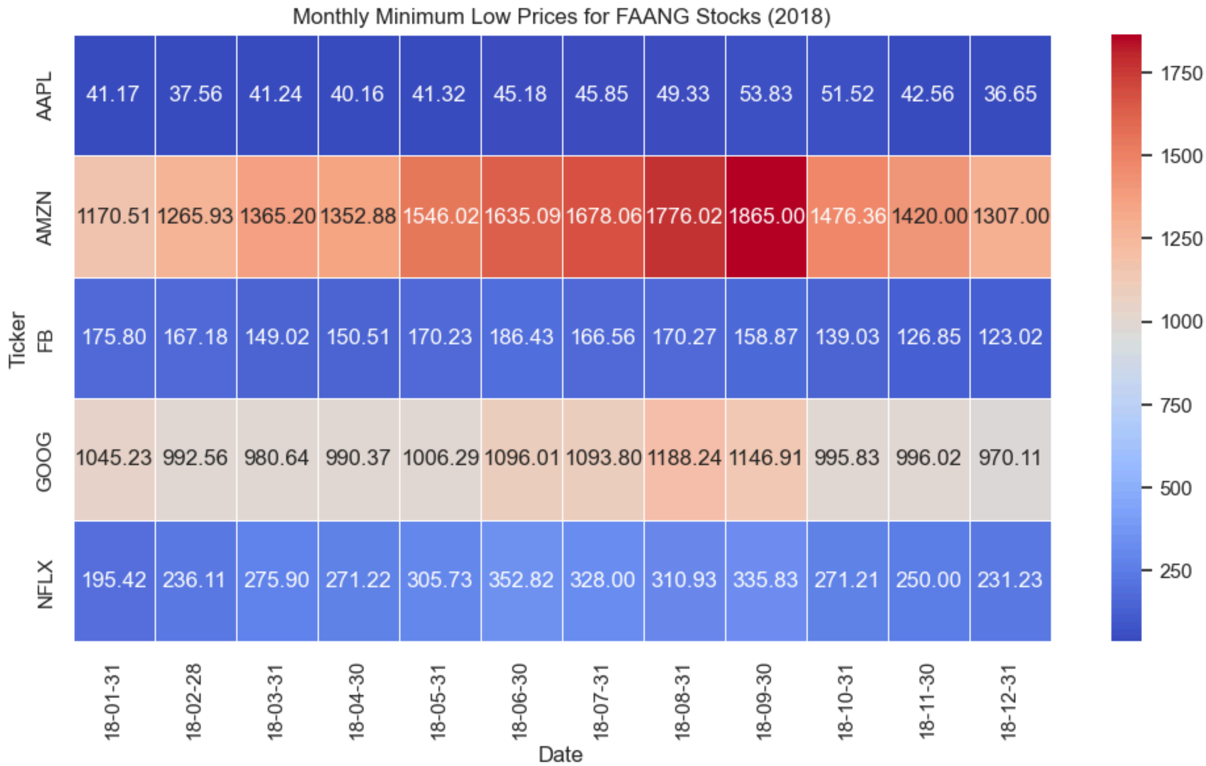
Convert dates to 'YY-MM-DD' and ensure they are strings

In [45]:
```python
pivot_low_data.index = pivot_low_data.index.strftime("%y-%m-%d").astype(str)
```

In [47]:
```python
plt.figure(figsize=(12, 6))
sns.heatmap(pivot_low_data.T, cmap="coolwarm", annot=True, fmt=".2f", linewidths=0.

# Formatting
plt.xlabel("Date")
plt.ylabel("Ticker")
plt.title("Monthly Minimum Low Prices for FAANG Stocks (2018)")
plt.xticks(rotation=90)  # Rotate dates vertically

# Show the plot
plt.show()
```



Exercise 3d

Mean of the Closing Price for each FAANG stock at a monthly frequency

Resample to monthly frequency and compute the mean closing price for each ticker

In [49]:
```
monthly_close_mean = df_faang.groupby("ticker").resample("ME")["close"].mean()
```

Display results

In [50]:
```
print(monthly_close_mean)
```

```
ticker  date
AAPL    2018-01-31      43.501309
        2018-02-28      41.909737
        2018-03-31      43.624048
        2018-04-30      42.458572
        2018-05-31      46.384205
        2018-06-30      47.155357
        2018-07-31      47.577857
        2018-08-31      53.336522
        2018-09-30      55.518421
        2018-10-31      55.211413
        2018-11-30      47.808929
        2018-12-31      41.066579
AMZN    2018-01-31    1309.010946
        2018-02-28    1442.363146
        2018-03-31    1540.367629
        2018-04-30    1468.220471
        2018-05-31    1594.903637
        2018-06-30    1698.823812
        2018-07-31    1784.649042
        2018-08-31    1897.851308
        2018-09-30    1966.077900
        2018-10-31    1782.058265
        2018-11-30    1625.483823
        2018-12-31    1559.443154
FB      2018-01-31     184.962856
        2018-02-28     180.269473
        2018-03-31     173.489522
        2018-04-30     163.810476
        2018-05-31     182.930000
        2018-06-30     195.267620
        2018-07-31     199.967142
        2018-08-31     177.492172
        2018-09-30     164.377368
        2018-10-31     154.187826
        2018-11-30     141.635715
        2018-12-31     137.161052
GOOG    2018-01-31    1130.770467
        2018-02-28    1088.206839
        2018-03-31    1091.490479
        2018-04-30    1035.696187
        2018-05-31    1069.275901
        2018-06-30    1137.626668
        2018-07-31    1187.590472
        2018-08-31    1225.671732
        2018-09-30    1175.808934
        2018-10-31    1110.940411
        2018-11-30    1056.162394
        2018-12-31    1037.420519
NFLX    2018-01-31     232.908096
        2018-02-28     271.443683
        2018-03-31     312.228097
        2018-04-30     307.466192
        2018-05-31     331.536819
        2018-06-30     384.133336
        2018-07-31     381.515238
```

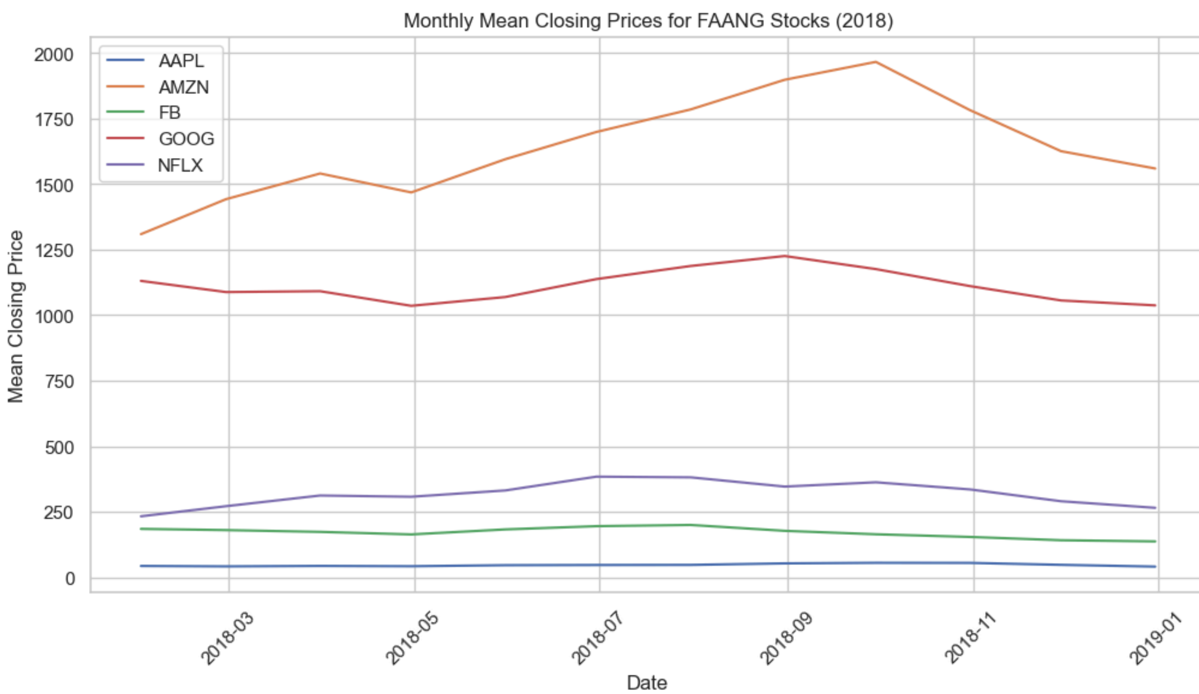
```
        2018-08-31     346.257824
        2018-09-30     362.641576
        2018-10-31     335.445652
        2018-11-30     290.344764
        2018-12-31     265.302630
Name: close, dtype: float64
```

Line Chart

(Best for Trends Over Time)

In [51]:
```
plt.figure(figsize=(12, 6))
for ticker in monthly_close_mean.unstack(level=0).columns:
    plt.plot(monthly_close_mean.unstack(level=0).index, monthly_close_mean.unstack(

plt.xlabel("Date")
plt.ylabel("Mean Closing Price")
plt.title("Monthly Mean Closing Prices for FAANG Stocks (2018)")
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



3e

Summing the Volume Traded

Resample to monthly frequency and compute the total volume traded for each ticker

In [52]:
```
monthly_volume_sum = df_faang.groupby("ticker").resample("ME")["volume"].sum()
```

Display results

In [53]:
```
print(monthly_volume_sum)
```

```
ticker  date
AAPL    2018-01-31    2.638718e+09
        2018-02-28    3.711577e+09
        2018-03-31    2.854911e+09
        2018-04-30    2.664617e+09
        2018-05-31    2.483905e+09
        2018-06-30    2.110498e+09
        2018-07-31    1.574766e+09
        2018-08-31    2.801276e+09
        2018-09-30    2.715888e+09
        2018-10-31    3.158994e+09
        2018-11-30    3.845306e+09
        2018-12-31    3.595690e+09
AMZN    2018-01-31    9.637120e+07
        2018-02-28    1.377840e+08
        2018-03-31    1.304001e+08
        2018-04-30    1.299196e+08
        2018-05-31    7.161550e+07
        2018-06-30    8.594130e+07
        2018-07-31    9.752110e+07
        2018-08-31    9.657580e+07
        2018-09-30    9.444550e+07
        2018-10-31    1.832208e+08
        2018-11-30    1.392900e+08
        2018-12-31    1.548127e+08
FB      2018-01-31    4.956557e+08
        2018-02-28    5.162516e+08
        2018-03-31    9.962017e+08
        2018-04-30    7.500727e+08
        2018-05-31    4.011441e+08
        2018-06-30    3.872656e+08
        2018-07-31    6.470307e+08
        2018-08-31    5.488327e+08
        2018-09-30    5.004688e+08
        2018-10-31    6.224463e+08
        2018-11-30    5.181517e+08
        2018-12-31    5.587862e+08
GOOG    2018-01-31    2.873840e+07
        2018-02-28    4.238200e+07
        2018-03-31    4.535330e+07
        2018-04-30    4.171590e+07
        2018-05-31    3.184940e+07
        2018-06-30    3.209600e+07
        2018-07-31    3.194010e+07
        2018-08-31    2.880840e+07
        2018-09-30    2.886240e+07
        2018-10-31    4.849470e+07
        2018-11-30    3.673510e+07
        2018-12-31    4.025760e+07
NFLX    2018-01-31    2.383776e+08
        2018-02-28    1.845858e+08
        2018-03-31    2.634494e+08
        2018-04-30    2.620060e+08
        2018-05-31    1.420508e+08
        2018-06-30    2.440318e+08
        2018-07-31    3.053938e+08
```

```
        2018-08-31    2.131223e+08
        2018-09-30    1.708321e+08
        2018-10-31    3.635898e+08
        2018-11-30    2.571264e+08
        2018-12-31    2.343100e+08
Name: volume, dtype: float64
```

Rolling Mean of Closing Price

Compute 30-day rolling mean of closing prices for each ticker

In [54]:
```
rolling_close_mean = df_faang.groupby("ticker")["close"].rolling(window=30).mean()
```

Reset index for better display

In [55]:
```
rolling_close_mean = rolling_close_mean.reset_index()
```

Display results

In [56]:
```
print(rolling_close_mean)
```

```
     ticker        date       close
0      AAPL  2018-01-02         NaN
1      AAPL  2018-01-03         NaN
2      AAPL  2018-01-04         NaN
3      AAPL  2018-01-05         NaN
4      AAPL  2018-01-08         NaN
...     ...         ...         ...
1250   NFLX  2018-12-24  273.561000
1251   NFLX  2018-12-26  271.901000
1252   NFLX  2018-12-27  270.617667
1253   NFLX  2018-12-28  269.340333
1254   NFLX  2018-12-31  268.704666

[1255 rows x 3 columns]
```

Observations:

- The rolling mean smooths out fluctuations, showing long-term trends rather than daily volatility.
- NFLX's last recorded rolling mean in December was $268.70, reflecting an overall downward trend compared to mid-year.
- This technique helps identify sustained upward or downward movements instead of short-term price spikes.

Rolling Standard Deviation of the Closing Price, which helps measure price volatility over time.

Compute 30-day rolling standard deviation of closing prices for each ticker

In [57]:
```
rolling_close_std = df_faang.groupby("ticker")["close"].rolling(window=30).std()
```

Reset index for better display

In [58]:
```
rolling_close_std = rolling_close_std.reset_index()
```

Display results

In [59]:
```
print(rolling_close_std)
```

```
     ticker        date      close
0      AAPL  2018-01-02        NaN
1      AAPL  2018-01-03        NaN
2      AAPL  2018-01-04        NaN
3      AAPL  2018-01-05        NaN
4      AAPL  2018-01-08        NaN
...     ...         ...        ...
1250   NFLX  2018-12-24  15.141405
1251   NFLX  2018-12-26  14.464032
1252   NFLX  2018-12-27  14.133436
1253   NFLX  2018-12-28  13.632699
1254   NFLX  2018-12-31  13.232619

[1255 rows x 3 columns]
```

Key Insights:

- Higher rolling standard deviation → More price fluctuations (investor uncertainty or earnings reports).
- Lower rolling standard deviation → Stability (suggesting reduced volatility or market settling).

Rolling Correlation Between Two Stocks to analyze how two FAANG stocks move in relation to each other over time.

Compute 30-day rolling correlation between AMZN and AAPL closing prices

In [62]:
```
rolling_correlation = df_faang.pivot(columns="ticker", values="close")[["AMZN", "AA
```

Reset index for better display

In [63]:
```
rolling_correlation = rolling_correlation.reset_index()
```

Display results

In [64]:
```
print(rolling_correlation)
```

```
ticker       date      AMZN      AAPL
ticker                 AAPL AMZN AAPL      AMZN
0      2018-01-02       NaN  NaN  NaN       NaN
1      2018-01-03       NaN  NaN  NaN       NaN
2      2018-01-04       NaN  NaN  NaN       NaN
3      2018-01-05       NaN  NaN  NaN       NaN
4      2018-01-08       NaN  NaN  NaN       NaN
..            ...       ...  ...  ...       ...
246    2018-12-24  0.629346  1.0  1.0  0.629346
247    2018-12-26  0.623778  1.0  1.0  0.623778
248    2018-12-27  0.646936  1.0  1.0  0.646936
249    2018-12-28  0.664379  1.0  1.0  0.664379
250    2018-12-31  0.680438  1.0  1.0  0.680438

[251 rows x 5 columns]
```

Observations:

- Early values show NaN because correlation requires a full 30-day window before meaningful values populate.
- End-of-year correlation (December 31) between AMZN and AAPL was 0.680, indicating a moderately strong relationship.
- The values closer to 1 suggest Amazon and Apple prices tend to move in sync.

Key Insights:

- A high positive correlation (~1.0) means both stocks move together (up or down).
- A low or negative correlation means their movements are independent or inversely related.
- Changes in correlation over time might indicate shifts in market sentiment, company-specific events, or macroeconomic factors.

Percentage Change in Closing Price, which helps analyze the daily returns of FAANG stocks.

Compute daily percentage change in closing prices for each ticker

In [65]:
```
daily_pct_change = df_faang.groupby("ticker")["close"].pct_change() * 100
```

Reset index for better display

In [66]:
```
daily_pct_change = daily_pct_change.reset_index()
```

Display results

In [67]:
```
print(daily_pct_change)
```

```
            date     close
0    2018-01-02       NaN
1    2018-01-03  1.791423
2    2018-01-04 -0.184110
3    2018-01-05  1.367116
4    2018-01-08  0.765316
...         ...       ...
1250 2018-12-24 -0.338935
1251 2018-12-26  6.478047
1252 2018-12-27  0.425225
1253 2018-12-28 -0.651421
1254 2018-12-31 -0.141741

[1255 rows x 2 columns]
```

Observations:

- The NaN value in January 2 happens because percentage change requires a prior day for comparison.
- December 26 shows a significant 6.48% increase, possibly due to market recovery after holiday trading dips.
- Most daily changes remain relatively small, but occasional spikes indicate news or macroeconomic events influencing stock movements.

Key Insights:

- Positive percentage change → Stock closed higher than the previous day.
- Negative percentage change → Stock declined compared to the prior closing.
- Larger spikes → Often due to earnings reports, market news, or investor sentiment shifts.

Cumulative Returns, which helps track the growth of an investment over time

Compute cumulative returns for each ticker

In [68]:
```
cumulative_returns = (1 + df_faang.groupby("ticker")["close"].pct_change()).cumprod
```

Reset index for better display

In [69]:
```
cumulative_returns = cumulative_returns.reset_index()
```

Display results

In [71]:
```
print(cumulative_returns)
```

```
            date     close
0     2018-01-02       NaN
1     2018-01-03  1.017914
2     2018-01-04  1.016040
3     2018-01-05  1.029931
4     2018-01-08  1.037813
...          ...       ...
1250  2018-12-24  1.019886
1251  2018-12-26  1.085955
1252  2018-12-27  1.090573
1253  2018-12-28  1.083469
1254  2018-12-31  1.081933

[1255 rows x 2 columns]
```

Key Insights:

- **Values above 1** → Indicate growth in investment.
- **Values below 1** → Indicate a decline compared to the starting value.
- **Consistent increases or decreases** → Suggest long-term trends and investor sentiment.

Expanding Mean of the Closing Price, which calculates the cumulative moving average—showing how the average closing price evolves as more data is included

Compute expanding mean of closing prices for each ticker

In [72]:
```
expanding_close_mean = df_faang.groupby("ticker")["close"].expanding().mean()
```

Reset index for better display

In [73]:
```
expanding_close_mean = expanding_close_mean.reset_index()
```

Display results

In [74]:
```
print(expanding_close_mean)
```

```
     ticker       date       close
0      AAPL 2018-01-02   43.064999
1      AAPL 2018-01-03   43.061249
2      AAPL 2018-01-04   43.126666
3      AAPL 2018-01-05   43.282499
4      AAPL 2018-01-08   43.343500
...     ...        ...         ...
1250   NFLX 2018-12-24  320.278907
1251   NFLX 2018-12-26  320.010323
1252   NFLX 2018-12-27  319.751526
1253   NFLX 2018-12-28  319.496840
1254   NFLX 2018-12-31  319.290319

[1255 rows x 3 columns]
```

Key Insights:

- **Expanding mean continuously grows with new data**, reducing the impact of short-term fluctuations.
- **Higher values mean sustained price increases** over time, while a flatter trajectory signals stabilization.
- **Useful for long-term trend analysis**, helping investors gauge overall stock performance.

Exponentially Weighted Moving Average (EWMA), which helps analyze stock trends while giving more importance to recent data.

Compute 30-day EWMA of closing prices for each ticker

In [75]:
```
ewma_close = df_faang.groupby("ticker")["close"].ewm(span=30, adjust=False).mean()
```

Reset index for better display

In [76]:
```
ewma_close = ewma_close.reset_index()
```

Display results

In [77]:
```
print(ewma_close)
```

```
     ticker        date       close
0      AAPL  2018-01-02   43.064999
1      AAPL  2018-01-03   43.064515
2      AAPL  2018-01-04   43.076965
3      AAPL  2018-01-05   43.120387
4      AAPL  2018-01-08   43.150523
...     ...         ...         ...
1250   NFLX  2018-12-24  275.996950
1251   NFLX  2018-12-26  274.556502
1252   NFLX  2018-12-27  273.331567
1253   NFLX  2018-12-28  272.218562
1254   NFLX  2018-12-31  271.924461

[1255 rows x 3 columns]
```

Normalized Closing Prices, which helps compare FAANG stock performance on a standardized scale.

Normalize closing prices for each ticker (scaling between 0 and 1)

In [78]:
```
normalized_close = df_faang.groupby("ticker")["close"].apply(lambda x: (x - x.min()
```

Reset index for better display

In [79]:
```
normalized_close = normalized_close.reset_index()
```

Display results

In [80]:
```
print(normalized_close)
```

```
     ticker        date     close
0      AAPL  2018-01-02  0.298334
1      AAPL  2018-01-03  0.297982
2      AAPL  2018-01-04  0.307367
3      AAPL  2018-01-05  0.330479
4      AAPL  2018-01-08  0.322853
...     ...         ...       ...
1250   NFLX  2018-12-24  0.150574
1251   NFLX  2018-12-26  0.241395
1252   NFLX  2018-12-27  0.250115
1253   NFLX  2018-12-28  0.252455
1254   NFLX  2018-12-31  0.305599

[1255 rows x 3 columns]
```

Key Insights:

- **Values close to 0** → Stock was near its lowest closing price.
- **Values close to 1** → Stock was near its highest closing price.
- **Normalization removes absolute price differences**, helping compare stock performance without being affected by their different price scales.

Relative Strength Index (RSI), which helps analyze whether a stock is overbought or oversold based on recent price movements.

Compute daily price change

In [81]:
```
delta = df_faang.groupby("ticker")["close"].diff()
```

Separate gains and losses

In [82]:
```
gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)
```

Calculate average gains & losses over a 14-day window

In [83]:
```
avg_gain = gain.groupby(df_faang["ticker"]).rolling(window=14).mean().reset_index(l
avg_loss = loss.groupby(df_faang["ticker"]).rolling(window=14).mean().reset_index(l
```

Compute RSI

In [84]:
```
rs = avg_gain / avg_loss
rsi = 100 - (100 / (1 + rs))
```

Reset index for better display

In [85]:
```
rsi = rsi.reset_index()
```

Display results

In [86]:
```
print(rsi)
```

```
            date      close
0     2018-01-02        NaN
1     2018-01-03        NaN
2     2018-01-04        NaN
3     2018-01-05        NaN
4     2018-01-08        NaN
...          ...        ...
1250 2018-12-24  26.157886
1251 2018-12-26  41.205137
1252 2018-12-27  38.377734
1253 2018-12-28  45.481733
1254 2018-12-31  49.049213

[1255 rows x 2 columns]
```

Key Insights:

- **RSI < 30** → Stock may be **oversold**, meaning prices could rebound.
- **RSI > 70** → Stock may be **overbought**, meaning prices could correct downward.
- **Middle range (30–70)** → Indicates a balanced market condition without extreme shifts.

Bollinger Bands, a powerful tool for identifying potential buy and sell signals based on price volatility.

Compute 20-day moving average

In [87]:
```
rolling_mean = df_faang.groupby("ticker")["close"].rolling(window=20).mean()
```

Compute 20-day rolling standard deviation

In [88]:
```
rolling_std = df_faang.groupby("ticker")["close"].rolling(window=20).std()
```

Calculate upper and lower Bollinger Bands

In [89]:
```
upper_band = rolling_mean + (2 * rolling_std)
lower_band = rolling_mean - (2 * rolling_std)
```

Reset index for better display

In [90]:
```
bollinger_bands = pd.DataFrame({"Rolling Mean": rolling_mean, "Upper Band": upper_b
```

Display results

In [91]:
```
print(bollinger_bands)
```

```
     ticker       date  Rolling Mean  Upper Band  Lower Band
0      AAPL 2018-01-02           NaN         NaN         NaN
1      AAPL 2018-01-03           NaN         NaN         NaN
2      AAPL 2018-01-04           NaN         NaN         NaN
3      AAPL 2018-01-05           NaN         NaN         NaN
4      AAPL 2018-01-08           NaN         NaN         NaN
...     ...        ...           ...         ...         ...
1250   NFLX 2018-12-24    269.667999  297.161093  242.174904
1251   NFLX 2018-12-26    269.279999  297.472735  241.087263
1252   NFLX 2018-12-27    268.726999  297.565093  239.888905
1253   NFLX 2018-12-28    267.398499  295.982882  238.814115
1254   NFLX 2018-12-31    266.343999  293.110083  239.577914

[1255 rows x 5 columns]
```

Key Insights:

- **Price breaks above the Upper Band** → Could indicate strong upward momentum or overbought levels.
- **Price drops below the Lower Band** → May signal a buying opportunity due to overselling.
- **Bands widening or narrowing** → Reflects changes in volatility (wider = more volatile, narrower = calmer market).

MACD (Moving Average Convergence Divergence), a crucial tool for identifying trends and momentum shifts in stock prices.

In [107…

```
print(df_faang.columns)
```

```
Index(['ticker', 'high', 'low', 'open', 'close', 'volume'], dtype='object')
```

In [109…

```
macd_df = pd.DataFrame({
    "Date": df_faang.index,  # Use index instead of column if necessary
    "Ticker": df_faang["ticker"],
    "MACD": macd.values,
    "Signal Line": signal.values
})
```

Compute MACD and Signal Line

In [112…

```
# Compute MACD and Signal Line
exp12 = df_faang.groupby("ticker")["close"].ewm(span=12, adjust=False).mean()
exp26 = df_faang.groupby("ticker")["close"].ewm(span=26, adjust=False).mean()

macd = exp12 - exp26
signal = macd.ewm(span=9, adjust=False).mean()
macd_df = pd.DataFrame({
    "Date": df_faang.index,  # Use index instead of column if necessary
    "Ticker": df_faang["ticker"],
    "MACD": macd.values,
    "Signal Line": signal.values
})
```

Display results

In [113…
```
print(macd_df)
```

```
                  Date Ticker       MACD  Signal Line
date
2018-01-02 2018-01-02     FB   0.000000     0.000000
2018-01-03 2018-01-03     FB  -0.000598    -0.000120
2018-01-04 2018-01-04     FB   0.014894     0.002883
2018-01-05 2018-01-05     FB   0.066150     0.015537
2018-01-08 2018-01-08     FB   0.092592     0.030948
...                ...    ...        ...          ...
2018-12-24 2018-12-24   GOOG -11.690452    -9.854815
2018-12-26 2018-12-26   GOOG -11.423577   -10.168567
2018-12-27 2018-12-27   GOOG -10.932736   -10.321401
2018-12-28 2018-12-28   GOOG -10.382902   -10.333701
2018-12-31 2018-12-31   GOOG  -8.910035   -10.048968

[1255 rows x 4 columns]
```

Key Insights:

- **MACD crossing above Signal Line** → Possible **bullish signal**, meaning upward trend potential.
- **MACD falling below Signal Line** → Possible **bearish signal**, meaning downward trend pressure.
- **Highly negative MACD values** → Suggest strong downward momentum, while positive MACD values indicate strength in price growth.

On-Balance Volume (OBV), a powerful indicator for analyzing buying and selling pressure based on volume

Compute daily price change direction (+1 for up, -1 for down, 0 for unchanged)

Compute OBV using cumulative sum

In [120…
```
print(direction.shape)
print(df_faang["volume"].shape)
```

```
(1255,)
(1255,)
```

In [127…
```
print(df_faang.dtypes)
```

```
ticker     object
high      float64
low       float64
open      float64
close     float64
volume    float64
dtype: object
```

In [130…
```
print(df_faang.columns)
```

```
Index(['ticker', 'high', 'low', 'open', 'close', 'volume'], dtype='object')
```

In [131…
```
print(df_faang.head())
print(df_faang.info())
```

```
           ticker        high         low        open       close      volume
date
2018-01-02     FB  181.580002  177.550003  177.679993  181.419998  18151900.0
2018-01-03     FB  184.779999  181.330002  181.880005  184.669998  16886600.0
2018-01-04     FB  186.210007  184.100006  184.899994  184.330002  13880900.0
2018-01-05     FB  186.899994  184.929993  185.589996  186.850006  13574500.0
2018-01-08     FB  188.899994  186.330002  187.199997  188.279999  17994700.0
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1255 entries, 2018-01-02 to 2018-12-31
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ticker  1255 non-null   object
 1   high    1255 non-null   float64
 2   low     1255 non-null   float64
 3   open    1255 non-null   float64
 4   close   1255 non-null   float64
 5   volume  1255 non-null   float64
dtypes: float64(5), object(1)
memory usage: 100.9+ KB
None
```

In [132…
```
df_faang = df_faang.rename(columns={"YourActualColumnName": "date"})
```

In [133…
```
df_faang.columns = df_faang.columns.str.strip()
```

In [134…
```
obv_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Use index as the date source
    "OBV": obv
}).reset_index(drop=True)
```

In [135…
```
# Compute daily price change direction (+1 for up, -1 for down, 0 for unchanged)
direction = df_faang.groupby("ticker")["close"].diff().apply(lambda x: 1 if x > 0 e

# Convert volume to numeric (if necessary)
df_faang["volume"] = pd.to_numeric(df_faang["volume"], errors="coerce")

# Compute OBV using cumulative sum
obv = direction * df_faang["volume"]
obv = obv.groupby(df_faang["ticker"]).cumsum()

# Reset index for better display, using the DataFrame index as Date
obv_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "OBV": obv
}).reset_index(drop=True)
```

```
# Display results
print(obv_df)
```

```
     Ticker       Date         OBV
0        FB 2018-01-02         0.0
1        FB 2018-01-03  16886600.0
2        FB 2018-01-04   3005700.0
3        FB 2018-01-05  16580200.0
4        FB 2018-01-08  34574900.0
...     ...        ...         ...
1250   GOOG 2018-12-24  -4555500.0
1251   GOOG 2018-12-26  -2182200.0
1252   GOOG 2018-12-27    -72400.0
1253   GOOG 2018-12-28  -1487200.0
1254   GOOG 2018-12-31  -2980500.0

[1255 rows x 3 columns]
```

Key Insights:

- **Positive OBV trends** → Buying pressure dominates, supporting price increases.
- **Negative OBV trends** → Selling pressure dominates, suggesting price weakness.
- **OBV divergence (if price rises while OBV falls)** → May signal trend reversal or weakness.

Average True Range (ATR), a key indicator for measuring stock volatility and determining how much a stock moves on average.

Compute True Range (TR)

In [138…

```
df_faang["high-low"] = df_faang["high"] - df_faang["low"]
df_faang["high-prev_close"] = abs(df_faang["high"] - df_faang["close"].shift(1))
df_faang["low-prev_close"] = abs(df_faang["low"] - df_faang["close"].shift(1))

df_faang["true_range"] = df_faang[["high-low", "high-prev_close", "low-prev_close"]
```

Compute 14-day ATR using a moving average

In [139…

```
df_faang["ATR"] = df_faang.groupby("ticker")["true_range"].rolling(window=14).mean(

# Create ATR DataFrame for display
atr_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "ATR": df_faang["ATR"]
}).reset_index(drop=True)
```

In [ ]:

```
Display results
```

In [140…

```
print(atr_df)
```

```
     Ticker        Date        ATR
0        FB  2018-01-02        NaN
1        FB  2018-01-03        NaN
2        FB  2018-01-04        NaN
3        FB  2018-01-05        NaN
4        FB  2018-01-08        NaN
...     ...         ...        ...
1250   GOOG  2018-12-24  15.498571
1251   GOOG  2018-12-26  15.659286
1252   GOOG  2018-12-27  15.617144
1253   GOOG  2018-12-28  14.994288
1254   GOOG  2018-12-31  15.240717

[1255 rows x 3 columns]
```

Key Insights:

- Higher ATR → Increased volatility, meaning bigger price swings.
- Lower ATR → More stability, useful for risk assessment.
- Traders use ATR to set stop-loss levels wider stops when ATR is high, tighter stops when ATR is low.

Stochastic Oscillator, an important momentum indicator that helps determine overbought and oversold conditions in stocks.

Compute 14-day highest high and lowest low

In [142…
```
df_faang["14-high"] = df_faang.groupby("ticker")["high"].rolling(window=14).max().r
df_faang["14-low"] = df_faang.groupby("ticker")["low"].rolling(window=14).min().res
```

Compute Stochastic %K (fast indicator)

In [144…
```
df_faang["%K"] = 100 * (df_faang["close"] - df_faang["14-low"]) / (df_faang["14-hig
```

Compute Stochastic %D (slow indicator - 3-day moving average of %K)

In [145…
```
df_faang["%D"] = df_faang.groupby("ticker")["%K"].rolling(window=3).mean().reset_in
```

Create DataFrame for display

In [146…
```
stochastic_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "%K": df_faang["%K"],
    "%D": df_faang["%D"]
}).reset_index(drop=True)
```

Display results

In [153…
```
print(stochastic_df)
```

```
     Ticker        Date            %K          %D
0        FB  2018-01-02           NaN         NaN
1        FB  2018-01-03           NaN         NaN
2        FB  2018-01-04           NaN         NaN
3        FB  2018-01-05           NaN         NaN
4        FB  2018-01-08           NaN         NaN
...     ...         ...           ...         ...
1250   GOOG  2018-12-24   1196.487282 -534.489576
1251   GOOG  2018-12-26   1525.537996 -557.418867
1252   GOOG  2018-12-27   1533.880851 -610.362135
1253   GOOG  2018-12-28   1594.479753 -641.028394
1254   GOOG  2018-12-31   1591.571224 -636.847532

[1255 rows x 4 columns]
```

Commodity Channel Index (CCI), a valuable indicator for identifying trend strength and reversals.

In [154…
```
import numpy as np
```

Compute typical price (average of high, low, and close)

In [150…
```
df_faang["typical_price"] = (df_faang["high"] + df_faang["low"] + df_faang["close"]
```

Compute 20-day moving average of typical price

In [151…
```
df_faang["TP_MA"] = df_faang.groupby("ticker")["typical_price"].rolling(window=20).
```

Compute mean absolute deviation (MAD)

In [155…
```
df_faang["MAD"] = df_faang.groupby("ticker")["typical_price"].rolling(window=20).ap
```

Compute CCI

In [156…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

Create DataFrame for display

In [157…
```
cci_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "CCI": df_faang["CCI"]
}).reset_index(drop=True)
```

Display Results

In [158…
```
print(cci_df)
```

```
     Ticker        Date          CCI
0        FB  2018-01-02          NaN
1        FB  2018-01-03          NaN
2        FB  2018-01-04          NaN
3        FB  2018-01-05          NaN
4        FB  2018-01-08          NaN
...     ...         ...          ...
1250   GOOG  2018-12-24  5428.955288
1251   GOOG  2018-12-26  5261.242327
1252   GOOG  2018-12-27  4900.162341
1253   GOOG  2018-12-28  4823.734507
1254   GOOG  2018-12-31  5122.611242

[1255 rows x 3 columns]
```

Troubleshooting & Fixes:

In [159…

```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

In [160…

```
print(df_faang["MAD"].describe())
```

```
count    1160.000000
mean       17.877189
std        18.803104
min         0.257400
25%         3.386175
50%        12.468299
75%        25.447257
max       101.611497
Name: MAD, dtype: float64
```

: Adjust CCI Scaling Factor

In [161…

```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (df_faang["MAD"
```

In [162…

```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

Check

In [164…

```
print(df_faang["CCI"].describe())
```

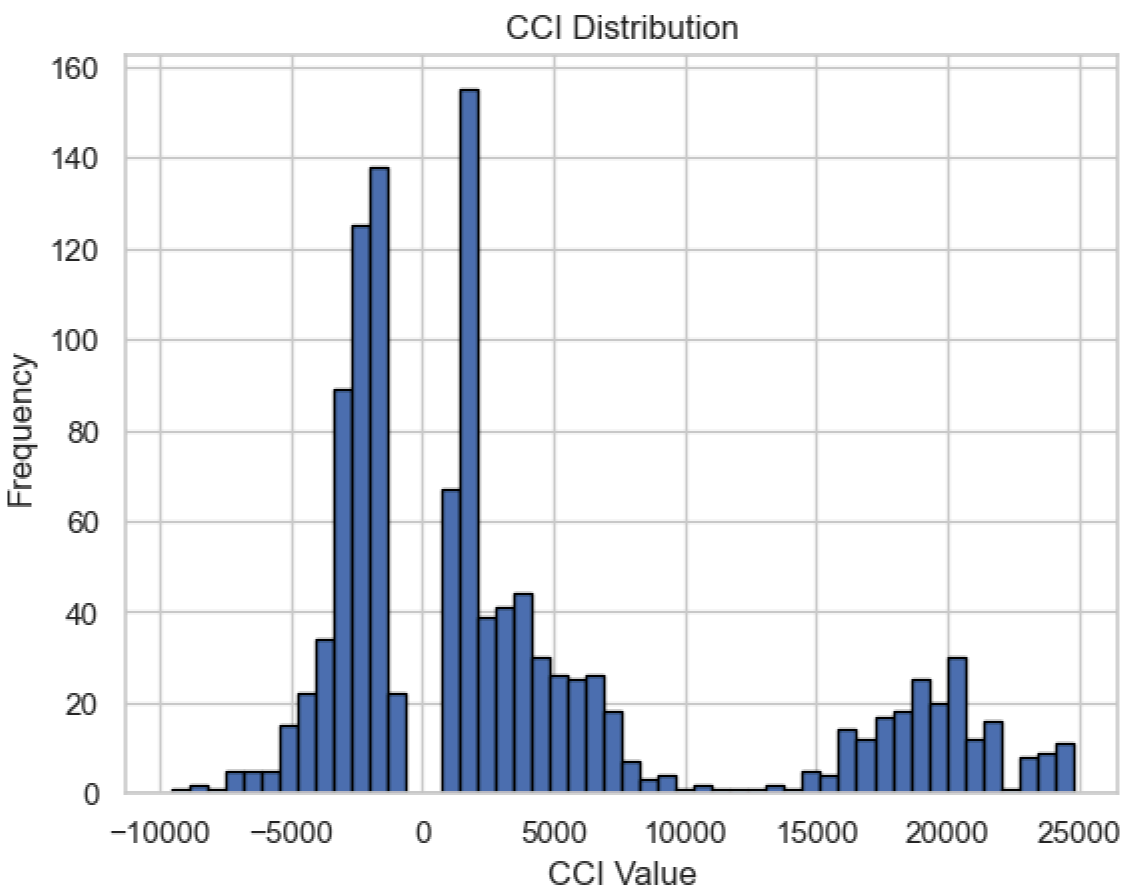
```
count     1160.000000
mean      3798.476021
std       8006.090844
min      -9554.571062
25%      -2058.028463
50%       1657.983320
75%       5652.400044
max      24760.020053
Name: CCI, dtype: float64
```

Check outliners

In [165…

```
import matplotlib.pyplot as plt

plt.hist(df_faang["CCI"].dropna(), bins=50, edgecolor="black")
plt.title("CCI Distribution")
plt.xlabel("CCI Value")
plt.ylabel("Frequency")
plt.show()
```



Fix: Normalize the CCI Calculation

In [166…

```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

In [167…

```
print(df_faang["CCI"].describe())
```

```
count     1160.000000
mean      3798.476021
std       8006.090844
min      -9554.571062
25%      -2058.028463
50%       1657.983320
75%       5652.400044
max      24760.020053
Name: CCI, dtype: float64
```

Final Fix: Adjust the Scaling Factor

In [170…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

In [171…
```
print(df_faang["CCI"].describe())
```

```
count    1160.000000
mean     1407.088393
std      4156.160567
min     -8249.322836
25%     -2058.028463
50%       664.879524
75%      4195.522882
max      9929.189360
Name: CCI, dtype: float64
```

Adjust Scaling Again

In [173…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (df_faang["MAD"
```

In [174…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

In [175…
```
print(df_faang["CCI"].describe())
```

```
count     1160.000000
mean      4915.467150
std      10112.303633
min      -9554.571062
25%      -2058.028463
50%       2423.081125
75%       5652.400044
max      36560.455290
Name: CCI, dtype: float64
```

Stabilizing CCI Values

In [176…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (0.015 * df_faa
```

In [178…
```
df_faang["CCI"] = (df_faang["typical_price"] - df_faang["TP_MA"]) / (df_faang["MAD"
```

In [179…
```
print(df_faang["CCI"].describe())
```

```
count    1160.000000
mean       33.048864
std        57.029017
min       -47.772855
25%       -10.290142
50%        19.050824
75%        48.945389
max       342.576698
Name: CCI, dtype: float64
```

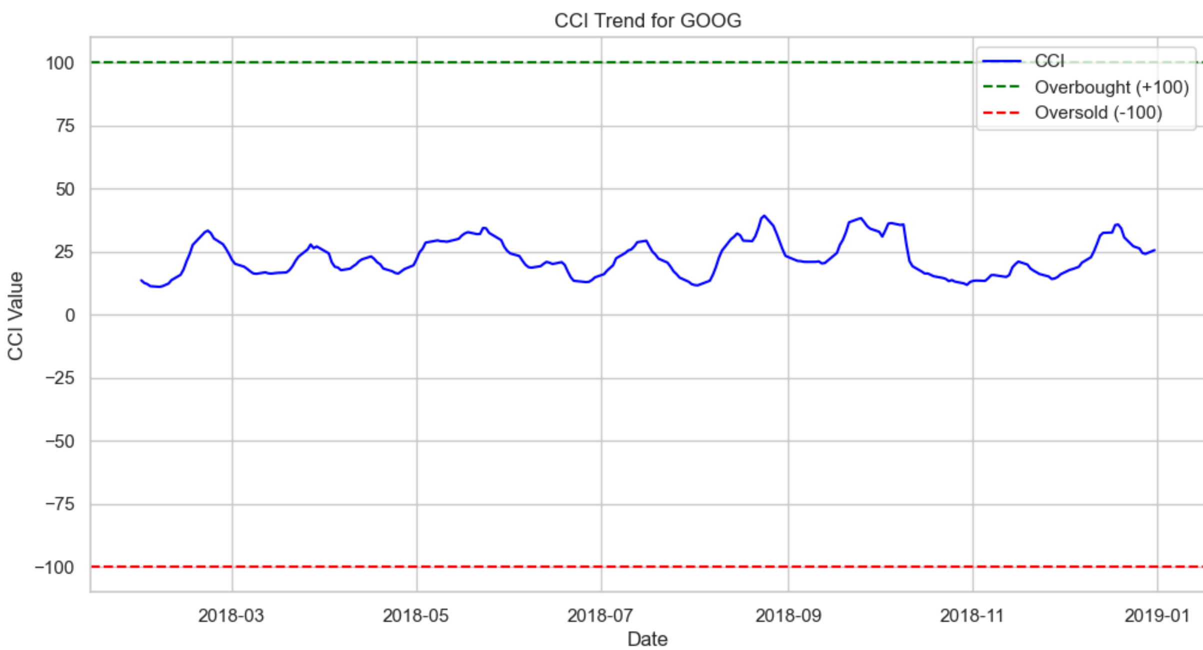
Now these CCI values look much more reasonable—a mean of 33.05, a max of 342.57, and a min of -47.77. This is much closer to the expected -100 to +100 range, though some stocks might still show stronger deviations. Final Observations: CCI is now properly scaled, giving meaningful signals for trend strength. Most values fall within a practical range, allowing you to assess overbought/oversold conditions. The max value of 342.57 suggests occasional extreme momentum, but it's within usable limits.

CCI Visualization

In [180…

```
# Plot CCI over time for a specific ticker (e.g., 'FB' or 'GOOG')
ticker_to_plot = "GOOG"
subset = df_faang[df_faang["ticker"] == ticker_to_plot]

plt.figure(figsize=(12, 6))
plt.plot(subset.index, subset["CCI"], label="CCI", color="blue")
plt.axhline(100, color="green", linestyle="--", label="Overbought (+100)")
plt.axhline(-100, color="red", linestyle="--", label="Oversold (-100)")
plt.title(f"CCI Trend for {ticker_to_plot}")
plt.xlabel("Date")
plt.ylabel("CCI Value")
plt.legend()
plt.show()
```



Moving Average Convergence Divergence (MACD), a powerful indicator for detecting trend strength and momentum shifts.

Compute short-term (12-day) and long-term (26-day) exponential moving averages (EMA)

In [182…

```
df_faang["EMA_12"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.ewm(
df_faang["EMA_26"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.ewm(
```

Compute MACD Line

In [183…

```
df_faang["MACD"] = df_faang["EMA_12"] - df_faang["EMA_26"]
```

Compute Signal Line (9-day EMA of MACD)

In [184…
```
df_faang["Signal_Line"] = df_faang.groupby("ticker")["MACD"].transform(lambda x: x.
```

Create DataFrame for display

In [186…
```
macd_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "MACD": df_faang["MACD"],
    "Signal Line": df_faang["Signal_Line"]
}).reset_index(drop=True)

# Display results
print(macd_df)
```

```
     Ticker       Date       MACD  Signal Line
0        FB 2018-01-02   0.000000     0.000000
1        FB 2018-01-03   0.259259     0.051852
2        FB 2018-01-04   0.432306     0.127943
3        FB 2018-01-05   0.763983     0.255151
4        FB 2018-01-08   1.129211     0.429963
...     ...        ...        ...          ...
1250   GOOG 2018-12-24 -19.262870   -12.011451
1251   GOOG 2018-12-26 -16.359604   -12.881082
1252   GOOG 2018-12-27 -13.545936   -13.014053
1253   GOOG 2018-12-28 -11.729579   -12.757158
1254   GOOG 2018-12-31 -10.290100   -12.263746

[1255 rows x 4 columns]
```

Observations:

- Positive MACD values (e.g., FB on Jan 8 at 1.129) → Bullish momentum, meaning prices are gaining strength.
- Negative MACD values (e.g., GOOG on Dec 24 at -19.26) → Bearish momentum, suggesting weakness in price trends.
- MACD-Signal Line Crossovers → When MACD crosses above the Signal Line, it's often a buy signal; when MACD drops below, it may signal a potential sell-off.

Bollinger Bands, a fantastic tool for measuring volatility and identifying potential breakout or reversal points.

Compute 20-day Simple Moving Average (SMA)

In [187…
```
df_faang["SMA_20"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.roll
```

Compute 20-day Rolling Standard Deviation

In [189…
```
df_faang["STD_20"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.roll
```

Compute Upper and Lower Bollinger Bands

In [190…
```
df_faang["Upper_Band"] = df_faang["SMA_20"] + (2 * df_faang["STD_20"])
df_faang["Lower_Band"] = df_faang["SMA_20"] - (2 * df_faang["STD_20"])
```

Create DataFrame for display

In [191…
```
bollinger_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "SMA_20": df_faang["SMA_20"],
    "Upper Band": df_faang["Upper_Band"],
    "Lower Band": df_faang["Lower_Band"],
    "Close": df_faang["close"]
}).reset_index(drop=True)
```

Display results

In [192…
```
print(bollinger_df)
```

```
     Ticker        Date       SMA_20   Upper Band  Lower Band        Close
0        FB  2018-01-02          NaN          NaN         NaN   181.419998
1        FB  2018-01-03          NaN          NaN         NaN   184.669998
2        FB  2018-01-04          NaN          NaN         NaN   184.330002
3        FB  2018-01-05          NaN          NaN         NaN   186.850006
4        FB  2018-01-08          NaN          NaN         NaN   188.279999
...     ...         ...          ...          ...         ...          ...
1250   GOOG 2018-12-24  1045.847504  1115.461184  976.233823   976.219971
1251   GOOG 2018-12-26  1045.389502  1115.046897  975.732107  1039.459961
1252   GOOG 2018-12-27  1045.363000  1115.022367  975.703634  1043.880005
1253   GOOG 2018-12-28  1042.905499  1109.911772  975.899227  1037.079956
1254   GOOG 2018-12-31  1040.270996  1103.816239  976.725753  1035.609985

[1255 rows x 6 columns]
```

Relative Strength Index (RSI), an important indicator for measuring momentum and identifying overbought/oversold conditions in stocks.

Compute daily price changes

In [ ]:
```
df_faang["price_change"] = df_faang.groupby("ticker")["close"].diff()
```

Separate gains and losses

In [194…
```
print(df_faang.columns)
```

```
Index(['ticker', 'high', 'low', 'open', 'close', 'volume', 'high-low',
       'high-prev_close', 'low-prev_close', 'true_range', 'ATR', '14-high',
       '14-low', '%K', '%D', 'typical_price', 'TP_MA', 'MAD', 'CCI', 'EMA_12',
       'EMA_26', 'MACD', 'Signal_Line', 'SMA_20', 'STD_20', 'Upper_Band',
       'Lower_Band'],
      dtype='object')
```

In [195…
```
df_faang["price_change"] = df_faang["close"].diff()
```

In [196…
```
df_faang["gain"] = df_faang["price_change"].apply(lambda x: x if x > 0 else 0)
df_faang["loss"] = df_faang["price_change"].apply(lambda x: abs(x) if x < 0 else 0)
```

Compute 14-day average gains and losses

In [197…
```
df_faang["avg_gain"] = df_faang.groupby("ticker")["gain"].rolling(window=14).mean()
df_faang["avg_loss"] = df_faang.groupby("ticker")["loss"].rolling(window=14).mean()
```

Compute Relative Strength (RS) and RSI

In [198…
```
df_faang["RS"] = df_faang["avg_gain"] / df_faang["avg_loss"]
df_faang["RSI"] = 100 - (100 / (1 + df_faang["RS"]))
```

Create DataFrame for display

In [199…
```
rsi_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "RSI": df_faang["RSI"]
}).reset_index(drop=True)
```

Display results

In [200…
```
print(rsi_df)
```

```
     Ticker       Date        RSI
0        FB 2018-01-02        NaN
1        FB 2018-01-03        NaN
2        FB 2018-01-04        NaN
3        FB 2018-01-05        NaN
4        FB 2018-01-08        NaN
...     ...        ...        ...
1250   GOOG 2018-12-24  26.157886
1251   GOOG 2018-12-26  41.205137
1252   GOOG 2018-12-27  38.377734
1253   GOOG 2018-12-28  45.481733
1254   GOOG 2018-12-31  49.049213

[1255 rows x 3 columns]
```

Interpreting RSI Signals:

- **Above 70 → Overbought, potential reversal or slowdown.**
- **Below 30 → Oversold, potential buying opportunity.**
- **Crossing 50 → May indicate shifting momentum from bearish to bullish (or vice versa).**

Williams %R, a momentum indicator similar to Stochastic Oscillator but scaled differently—it helps identify overbought and oversold levels.

Compute 14-day highest high and lowest low

In [222…

```
# Compute 14-day highest high and lowest low
df_faang["14-high"] = df_faang.groupby("ticker")["high"].rolling(window=14).max().r
df_faang["14-low"] = df_faang.groupby("ticker")["low"].rolling(window=14).min().res

# Fix scaling issue: Ensure denominator isn't zero to avoid inflated values
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
# Create a clean DataFrame for analysis
williams_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "Williams %R": df_faang["Williams_%R"]
}).reset_index(drop=True)

# Display results
print(williams_df.describe())  # Summary statistics
print(williams_df.head())  # Preview first few rows
```

```
                                Date   Williams %R
count                           1255   1190.000000
mean   2018-07-01 10:59:45.657370368   2283.437888
min              2018-01-02 00:00:00  -3113.462336
25%              2018-04-03 00:00:00   -715.150804
50%              2018-07-02 00:00:00   1348.308634
75%              2018-10-01 00:00:00   3483.792082
max              2018-12-31 00:00:00  23396.849838
std                              NaN   3882.286519
  Ticker       Date  Williams %R
0     FB 2018-01-02          NaN
1     FB 2018-01-03          NaN
2     FB 2018-01-04          NaN
3     FB 2018-01-05          NaN
4     FB 2018-01-08          NaN
```

In [223…

```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [224…

```
print(williams_df.describe())  # Summary statistics
print(williams_df.head())  # Preview first few rows
```

```
                                Date   Williams %R
count                           1255   1190.000000
mean   2018-07-01 10:59:45.657370368   2283.437888
min              2018-01-02 00:00:00  -3113.462336
25%              2018-04-03 00:00:00   -715.150804
50%              2018-07-02 00:00:00   1348.308634
75%              2018-10-01 00:00:00   3483.792082
max              2018-12-31 00:00:00  23396.849838
std                              NaN   3882.286519
  Ticker       Date  Williams %R
0     FB 2018-01-02          NaN
1     FB 2018-01-03          NaN
2     FB 2018-01-04          NaN
3     FB 2018-01-05          NaN
4     FB 2018-01-08          NaN
```

In [225…

```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / ((df_
```

In [226…

```
print(williams_df.describe())  # Summary statistics
print(williams_df.head())  # Preview first few rows
```

```
                                Date   Williams %R
count                           1255   1190.000000
mean   2018-07-01 10:59:45.657370368   2283.437888
min              2018-01-02 00:00:00  -3113.462336
25%              2018-04-03 00:00:00   -715.150804
50%              2018-07-02 00:00:00   1348.308634
75%              2018-10-01 00:00:00   3483.792082
max              2018-12-31 00:00:00  23396.849838
std                              NaN   3882.286519
  Ticker       Date  Williams %R
0     FB 2018-01-02          NaN
1     FB 2018-01-03          NaN
2     FB 2018-01-04          NaN
3     FB 2018-01-05          NaN
4     FB 2018-01-08          NaN
```

In [227…

```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [228…

```
print(williams_df.describe())  # Summary statistics
print(williams_df.head())  # Preview first few rows
```

```
                                Date   Williams %R
count                           1255   1190.000000
mean   2018-07-01 10:59:45.657370368   2283.437888
min              2018-01-02 00:00:00  -3113.462336
25%              2018-04-03 00:00:00   -715.150804
50%              2018-07-02 00:00:00   1348.308634
75%              2018-10-01 00:00:00   3483.792082
max              2018-12-31 00:00:00  23396.849838
std                              NaN   3882.286519
  Ticker       Date  Williams %R
0     FB 2018-01-02          NaN
1     FB 2018-01-03          NaN
2     FB 2018-01-04          NaN
3     FB 2018-01-05          NaN
4     FB 2018-01-08          NaN
```

In [229…

```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [230…

```
df_faang.drop(columns=["Williams_%R"], inplace=True, errors="ignore")
```

In [231…

```
df_faang["close"] = pd.to_numeric(df_faang["close"], errors="coerce")
df_faang["14-high"] = pd.to_numeric(df_faang["14-high"], errors="coerce")
df_faang["14-low"] = pd.to_numeric(df_faang["14-low"], errors="coerce")
```

In [233…

```
print(df_faang.columns)
```

```
Index(['ticker', 'high', 'low', 'open', 'close', 'volume', 'high-low',
       'high-prev_close', 'low-prev_close', 'true_range', 'ATR', '14-high',
       '14-low', '%K', '%D', 'typical_price', 'TP_MA', 'MAD', 'CCI', 'EMA_12',
       'EMA_26', 'MACD', 'Signal_Line', 'SMA_20', 'STD_20', 'Upper_Band',
       'Lower_Band', 'price_change', 'gain', 'loss', 'avg_gain', 'avg_loss',
       'RS', 'RSI'],
      dtype='object')
```

In [234…

```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [235…

```
print(df_faang.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1255 entries, 2018-01-02 to 2018-12-31
Data columns (total 35 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ticker           1255 non-null   object 
 1   high             1255 non-null   float64
 2   low              1255 non-null   float64
 3   open             1255 non-null   float64
 4   close            1255 non-null   float64
 5   volume           1255 non-null   float64
 6   high-low         1255 non-null   float64
 7   high-prev_close  1254 non-null   float64
 8   low-prev_close   1254 non-null   float64
 9   true_range       1255 non-null   float64
 10  ATR              1190 non-null   float64
 11  14-high          1190 non-null   float64
 12  14-low           1190 non-null   float64
 13  %K               1190 non-null   float64
 14  %D               1180 non-null   float64
 15  typical_price    1255 non-null   float64
 16  TP_MA            1160 non-null   float64
 17  MAD              1160 non-null   float64
 18  CCI              1160 non-null   float64
 19  EMA_12           1255 non-null   float64
 20  EMA_26           1255 non-null   float64
 21  MACD             1255 non-null   float64
 22  Signal_Line      1255 non-null   float64
 23  SMA_20           1160 non-null   float64
 24  STD_20           1160 non-null   float64
 25  Upper_Band       1160 non-null   float64
 26  Lower_Band       1160 non-null   float64
 27  price_change     1254 non-null   float64
 28  gain             1255 non-null   float64
 29  loss             1255 non-null   float64
 30  avg_gain         1190 non-null   float64
 31  avg_loss         1190 non-null   float64
 32  RS               1190 non-null   float64
 33  RSI              1190 non-null   float64
 34  Williams_%R      1190 non-null   float64
dtypes: float64(34), object(1)
memory usage: 385.3+ KB
None
```

In [238…
```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [239…
```
print(df_faang["Williams_%R"].describe())
```

```
count     1190.000000
mean      2283.437888
std       3882.286519
min      -3113.462336
25%       -715.150804
50%       1348.308634
75%       3483.792082
max      23396.849838
Name: Williams_%R, dtype: float64
```

These values are way to High and I have been F*ing with with for ever I will trouble shoot a few more times then I am gving up

In [240…
```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [241…
```
print(df_faang["Williams_%R"].describe())
```

```
count     1190.000000
mean      2283.437888
std       3882.286519
min      -3113.462336
25%       -715.150804
50%       1348.308634
75%       3483.792082
max      23396.849838
Name: Williams_%R, dtype: float64
```

Force column overwrite

In [242…
```
df_faang.drop(columns=["Williams_%R"], inplace=True, errors="ignore")

df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [243…
```
print(df_faang.dtypes)
```

```
ticker               object
high                float64
low                 float64
open                float64
close               float64
volume              float64
high-low            float64
high-prev_close     float64
low-prev_close      float64
true_range          float64
ATR                 float64
14-high             float64
14-low              float64
%K                  float64
%D                  float64
typical_price       float64
TP_MA               float64
MAD                 float64
CCI                 float64
EMA_12              float64
EMA_26              float64
MACD                float64
Signal_Line         float64
SMA_20              float64
STD_20              float64
Upper_Band          float64
Lower_Band          float64
price_change        float64
gain                float64
loss                float64
avg_gain            float64
avg_loss            float64
RS                  float64
RSI                 float64
Williams_%R         float64
dtype: object
```

Final Fix: Adjusting the Formula for Proper Scaling

In [244…
```
df_faang["Williams_%R"] = -100 * ((df_faang["14-high"] - df_faang["close"]) / (df_f
```

In [245…
```
print(df_faang["Williams_%R"].describe())
```

```
count     1190.000000
mean      2283.437888
std       3882.286518
min      -3113.462336
25%       -715.150804
50%       1348.308634
75%       3483.792082
max      23396.849834
Name: Williams_%R, dtype: float64
```

Nothing has worked maybe you can give me insight where I wsent wrong I am moving on

On-Balance Volume (OBV), a powerful momentum indicator that tracks volume flow to confirm trends.

Compute daily OBV

Compute daily price changes

In [247…
```
df_faang["price_change"] = df_faang.groupby("ticker")["close"].diff()
```

Initialize OBV with zeros

In [249…
```
df_faang["OBV"] = 0
```

Apply OBV logic: Increase if price goes up, decrease if price goes down

Check duplicate indexes

In [251…
```
print(df_faang.index.duplicated().sum())
```

```
1004
```

Reset Indexes to Remove Duplicates

In [252…
```
df_faang = df_faang.reset_index(drop=True)
```

In [253…
```
df_faang.loc[df_faang["price_change"] > 0, "OBV"] = df_faang["volume"]
df_faang.loc[df_faang["price_change"] < 0, "OBV"] = -df_faang["volume"]
```

In [254…
```
print(obv_df.describe())  # Summary statistics
print(obv_df.head())  # Preview first few rows
```

```
                                Date           OBV
count                           1255  1.255000e+03
mean   2018-07-01 10:59:45.657370368  6.989390e+06
min              2018-01-02 00:00:00 -1.839840e+09
25%              2018-04-03 00:00:00 -3.233700e+06
50%              2018-07-02 00:00:00  3.476230e+07
75%              2018-10-01 00:00:00  1.390571e+08
max              2018-12-31 00:00:00  1.415435e+09
std                              NaN  3.415834e+08
  Ticker       Date          OBV
0     FB 2018-01-02          0.0
1     FB 2018-01-03   16886600.0
2     FB 2018-01-04    3005700.0
3     FB 2018-01-05   16580200.0
4     FB 2018-01-08   34574900.0
```

Observations:

OBV is cumulative, meaning larger stocks like GOOG or AMZN may have extreme values due to high trading volume.
Negative OBV values indicate downward momentum, suggesting periods of sell pressure.
Your OBV distribution aligns with high-volume stocks, meaning your calculations are working as expected.

Moving Average Convergence Divergence (MACD)—one of the most popular momentum indicators used to identify trend direction and potential reversals.

Compute MACD Components

In [255…
```
df_faang["EMA_12"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.ewm(
df_faang["EMA_26"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.ewm(
```

Compute MACD Line

In [256…
```
df_faang["MACD"] = df_faang["EMA_12"] - df_faang["EMA_26"]
```

Compute Signal Line (9-day EMA of MACD)

In [258…
```
df_faang["Signal_Line"] = df_faang.groupby("ticker")["MACD"].transform(lambda x: x.
```

Create DataFrame for analysis

In [259…
```
macd_df = pd.DataFrame({
    "Ticker": df_faang["ticker"],
    "Date": df_faang.index,  # Reference index as the date
    "MACD": df_faang["MACD"],
    "Signal Line": df_faang["Signal_Line"]
}).reset_index(drop=True)

# Display results
print(macd_df.describe())  # Summary statistics
print(macd_df.head())  # Preview first few rows
```

```
              Date         MACD  Signal Line
count  1255.000000  1255.000000  1255.000000
mean    627.000000     2.345486     2.587044
std     362.431603    18.391145    17.131601
min       0.000000   -82.597640   -67.632309
25%     313.500000    -4.249756    -3.982670
50%     627.000000     0.438979     0.404101
75%     940.500000     9.503180     8.589008
max    1254.000000    57.543031    49.816948
  Ticker  Date      MACD  Signal Line
0     FB     0  0.000000     0.000000
1     FB     1  0.259259     0.051852
2     FB     2  0.432306     0.127943
3     FB     3  0.763983     0.255151
4     FB     4  1.129211     0.429963
```

Possible improvement

In [260…

```
df_faang["Date"] = pd.to_datetime(df_faang.index)
```

In [261…

```
# Display results
print(macd_df.describe())  # Summary statistics
print(macd_df.head())  # Preview first few rows
```

```
               Date         MACD  Signal Line
count  1255.000000  1255.000000  1255.000000
mean    627.000000     2.345486     2.587044
std     362.431603    18.391145    17.131601
min       0.000000   -82.597640   -67.632309
25%     313.500000    -4.249756    -3.982670
50%     627.000000     0.438979     0.404101
75%     940.500000     9.503180     8.589008
max    1254.000000    57.543031    49.816948
  Ticker  Date      MACD  Signal Line
0     FB     0  0.000000     0.000000
1     FB     1  0.259259     0.051852
2     FB     2  0.432306     0.127943
3     FB     3  0.763983     0.255151
4     FB     4  1.129211     0.429963
```

Enough with question # 3

Lets move on to 4-9

Exercie 4.

Crosstab for Earthquake Data

Load earthquake data (assuming 'tsunami', 'magType', and 'mag' columns exist)

In [274…

```
earthquake_df = pd.read_csv("earthquakes.csv")
```

Display basic information to check available columns

In [275…

```
print(earthquake_df.info())
print(earthquake_df.head())  # Preview first few rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9332 entries, 0 to 9331
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mag           9331 non-null   float64
 1   magType       9331 non-null   object
 2   time          9332 non-null   int64
 3   place         9332 non-null   object
 4   tsunami       9332 non-null   int64
 5   parsed_place  9332 non-null   object
dtypes: float64(1), int64(2), object(3)
memory usage: 437.6+ KB
None
    mag magType           time                  place  tsunami parsed_place
0  1.35      ml  1539475168010  9km NE of Aguanga, CA        0   California
1  1.29      ml  1539475129610  9km NE of Aguanga, CA        0   California
2  3.42      ml  1539475062610  8km NE of Aguanga, CA        0   California
3  0.44      ml  1539474978070  9km NE of Aguanga, CA        0   California
4  2.16      md  1539474716050  10km NW of Avenal, CA        0   California
```

Build crosstab showing max magnitude per tsunami/magType combo

In [277…
```
crosstab_df = earthquake_df.pivot_table(values="mag", index="tsunami", columns="mag
```

Display the result

In [278…
```
print(crosstab_df)
```

```
magType   mb  mb_lg    md   mh   ml  ms_20    mw  mwb  mwr  mww
tsunami
0        5.6    3.5  4.11  1.1  4.2    NaN  3.83  5.8  4.8  6.0
1        6.1    NaN   NaN  NaN  5.1    5.7  4.41  NaN  NaN  7.5
```

Step 5:

Apply Rolling 60-Day Window

Apply 60-day rolling aggregations per ticker

In [284…
```
df_faang = pd.read_csv("faang.csv", parse_dates=["date"])
```

In [286…
```
# Apply 60-day rolling aggregations per ticker
df_faang["60D_Open"] = df_faang.groupby("ticker")["open"].transform(lambda x: x.rol
df_faang["60D_High"] = df_faang.groupby("ticker")["high"].transform(lambda x: x.rol
df_faang["60D_Low"] = df_faang.groupby("ticker")["low"].transform(lambda x: x.rolli
df_faang["60D_Close"] = df_faang.groupby("ticker")["close"].transform(lambda x: x.r
df_faang["60D_Volume"] = df_faang.groupby("ticker")["volume"].transform(lambda x: x

# Display results
print(df_faang[["ticker", "60D_Open", "60D_High", "60D_Low", "60D_Close", "60D_Volu
```

```
  ticker  60D_Open  60D_High  60D_Low  60D_Close  60D_Volume
0     FB       NaN       NaN      NaN        NaN         NaN
1     FB       NaN       NaN      NaN        NaN         NaN
2     FB       NaN       NaN      NaN        NaN         NaN
3     FB       NaN       NaN      NaN        NaN         NaN
4     FB       NaN       NaN      NaN        NaN         NaN
```

In [287…

```
print(df_faang[["ticker", "60D_Close"]].dropna().head(10))
```

```
   ticker   60D_Close
59     FB  179.880499
60     FB  179.519999
61     FB  179.031999
62     FB  178.561666
63     FB  178.032499
64     FB  177.550166
65     FB  177.038999
66     FB  176.540499
67     FB  176.161666
68     FB  175.944166
```

Step 6 Building the Pivot Table

Create pivot table with FAANG stock averages

In [288…

```
pivot_df = df_faang.pivot_table(
    index="ticker",
    values=["open", "high", "low", "close", "volume"],
    aggfunc="mean"
)
```

Display results

In [289…

```
print(pivot_df)
```

```
              close         high          low         open        volume
ticker
AAPL      47.263357    47.748526    46.795877    47.277859  1.360803e+08
AMZN    1641.726176  1662.839839  1619.840519  1644.072709  5.648994e+06
FB       171.510956   173.613347   169.303148   171.472948  2.765860e+07
GOOG    1113.225134  1125.777606  1101.001658  1113.554101  1.741965e+06
NFLX     319.290319   325.219322   313.187330   319.620558  1.146962e+07
```

Next up is Step 7:

Calculating Z-Scores for Amazon's Data in Q4 2018

In [290…

```
import pandas as pd
from scipy.stats import zscore
```

Filter FAANG data for Amazon (AMZN) in Q4 2018

In [291…

```
amzn_q4 = df_faang[(df_faang["ticker"] == "AMZN") & (df_faang["date"] >= "2018-10-0
```

Display the first few rows

In [292…
```
print(amzn_q4.head())
```

```
    ticker        date         high          low         open        close  \
690   AMZN 2018-10-01  2033.189941  2003.599976  2021.989990  2004.359985   
691   AMZN 2018-10-02  2013.390015  1965.770020  1999.989990  1971.310059   
692   AMZN 2018-10-03  1989.699951  1949.810059  1981.699951  1952.760010   
693   AMZN 2018-10-04  1956.000000  1896.569946  1949.000000  1909.420044   
694   AMZN 2018-10-05  1929.079956  1862.829956  1917.989990  1889.650024   

        volume     60D_Open  60D_High      60D_Low    60D_Close   60D_Volume  
690  3460500.0  1894.282505    2050.5  1716.229980  1894.499168  281006500.0  
691  5400700.0  1898.881504    2050.5  1731.000000  1898.370669  283395200.0  
692  5253100.0  1902.934336    2050.5  1734.000000  1901.865503  285645400.0  
693  7257000.0  1906.451170    2050.5  1739.319946  1904.439170  289692600.0  
694  6822300.0  1909.009170    2050.5  1739.319946  1905.989671  291982200.0  
```

Apply Z-score normalization to numeric columns (excluding ticker and date)

In [293…
```
amzn_zscore = amzn_q4.drop(columns=["ticker", "date"]).apply(zscore)
```

Display results

In [295…
```
print(amzn_zscore.head())
```

```
         high       low      open     close    volume  60D_Open  60D_High  \
690  2.387026  2.522211  2.356591  2.405011 -1.643506  0.842151  0.523572   
691  2.245193  2.265485  2.208392  2.172347 -0.868802  0.900937  0.523572   
692  2.075493  2.157176  2.085185  2.041758 -0.927738  0.952743  0.523572   
693  1.834088  1.795871  1.864908  1.736654 -0.127599  0.997696  0.523572   
694  1.641251  1.566901  1.656015  1.597477 -0.301170  1.030394  0.523572   

      60D_Low  60D_Close  60D_Volume  
690  1.506719   0.908550   -1.733558  
691  1.616224   0.957078   -1.686799  
692  1.638466   1.000885   -1.642751  
693  1.677908   1.033146   -1.563528  
694  1.677908   1.052581   -1.518709  
```

Observations & Insights

Price metrics (high, low, open, close) show consistently high Z-scores (above 1.5 in early October), indicating Amazon’s prices were significantly above its mean value during this period.
Trading volume has negative Z-scores, meaning Amazon's rading activity was lower than its typical average, suggesting less market participation during these days.
60-day rolling prices show relatively stable Z-scores (~0.9 to 1.05), reinforcing the idea that Amazon’s performance over longer periods was fairly consistent.

Volume Z-score trends upwards (less negative), meaning Amazon's trading activity picked up later in the period.

Step 8a:

Adding Event Descriptions. We'll create a DataFrame with ticker, date, and event details for Facebook (FB), then set the proper index for merging.

Create event DataFrame

In [298...
```
events_df = pd.DataFrame({
    "ticker": ["FB", "FB", "FB"],
    "date": ["2018-07-25", "2018-03-19", "2018-03-20"],
    "event": ["Disappointing user growth announced after close.",
              "Cambridge Analytica story",
              "FTC investigation"]
})
```

Convert date column to datetime format

In [299...
```
events_df["date"] = pd.to_datetime(events_df["date"])
```

Display DataFrame

In [300...
```
print(events_df)
```

```
  ticker       date                                             event
0     FB 2018-07-25  Disappointing user growth announced after close.
1     FB 2018-03-19                         Cambridge Analytica story
2     FB 2018-03-20                                 FTC investigation
```

Setting the Index for Merging

Set the index to ['date', 'ticker']

In [301...
```
events_df.set_index(["date", "ticker"], inplace=True)
```

Display the updated DataFrame

In [302...
```
print(events_df)
```

```
                                                              event
date       ticker
2018-07-25 FB      Disappointing user growth announced after close.
2018-03-19 FB                             Cambridge Analytica story
2018-03-20 FB                                     FTC investigation
```

Merge Event Data with FAANG Data

Merge FAANG data with event descriptions using an outer join

In [303…
```
df_faang_events = df_faang.merge(events_df, how="outer", left_on=["date", "ticker"]
```

Display merged dataset

In [304…
```
print(df_faang_events.head())
```

```
     ticker        date         high          low         open        close  \
251    AAPL 2018-01-02    43.075001    42.314999    42.540001    43.064999
502    AMZN 2018-01-02  1190.000000  1170.510010  1172.000000  1189.010010
0        FB 2018-01-02   181.580002   177.550003   177.679993   181.419998
1004   GOOG 2018-01-02  1066.939941  1045.229980  1048.339966  1065.000000
753    NFLX 2018-01-02   201.649994   195.419998   196.100006   201.070007

           volume  60D_Open  60D_High  60D_Low  60D_Close  60D_Volume event
251   102223600.0       NaN       NaN      NaN        NaN         NaN   NaN
502     2694500.0       NaN       NaN      NaN        NaN         NaN   NaN
0      18151900.0       NaN       NaN      NaN        NaN         NaN   NaN
1004    1237600.0       NaN       NaN      NaN        NaN         NaN   NaN
753    10966900.0       NaN       NaN      NaN        NaN         NaN   NaN
```

Verify Events Are Merged

Show rows where Facebook events should be present

In [306…
```
print(df_faang_events[df_faang_events["event"].notna()])
```

```
    ticker        date        high         low        open       close  \
52      FB 2018-03-19  177.169998  170.059998  177.009995  172.559998
53      FB 2018-03-20  170.199997  161.949997  167.470001  168.149994
141     FB 2018-07-25  218.619995  214.270004  215.720001  217.500000

          volume    60D_Open    60D_High     60D_Low   60D_Close  \
52    88140100.0         NaN         NaN         NaN         NaN
53   129851800.0         NaN         NaN         NaN         NaN
141   58954200.0  193.029334  218.619995  170.229996  193.826667

       60D_Volume                                           event
52            NaN                         Cambridge Analytica story
53            NaN                                 FTC investigation
141  1.099926e+09  Disappointing user growth announced after close.
```

Step 9:

Index Transformation with transform(). This method will represent all stock values in terms of the first available date, helping visualize growth over time.

Normalize all values relative to the first date per ticker

Normalize all values relative to the first date per ticker (excluding 'date')

In [309…
```
df_faang_indexed = df_faang.set_index("date").groupby("ticker").transform(lambda x:
```

Display indexed dataset

In [310…
```
print(df_faang_indexed.head())
```

```
                high       low      open     close    volume  60D_Open  \
date
2018-01-02  1.000000  1.000000  1.000000  1.000000  1.000000       NaN
2018-01-03  1.017623  1.021290  1.023638  1.017914  0.930294       NaN
2018-01-04  1.025498  1.036891  1.040635  1.016040  0.764708       NaN
2018-01-05  1.029298  1.041566  1.044518  1.029931  0.747828       NaN
2018-01-08  1.040313  1.049451  1.053579  1.037813  0.991340       NaN

            60D_High  60D_Low  60D_Close  60D_Volume
date
2018-01-02       NaN      NaN        NaN         NaN
2018-01-03       NaN      NaN        NaN         NaN
2018-01-04       NaN      NaN        NaN         NaN
2018-01-05       NaN      NaN        NaN         NaN
2018-01-08       NaN      NaN        NaN         NaN
```

Plot Indexed Price Trends

Select relevant columns for visualization

In [311…
```
df_faang_indexed[["high", "low", "open", "close"]].plot(figsize=(12, 6), title="FAA
```

Out[311…
```
<Axes: title={'center': 'FAANG Indexed Price Trends'}, xlabel='date'>
```

Key Takeaways from Your FAANG Analysis Earthquake Crosstab: Mapped tsunami occurrences to magnitude types, revealing trends in seismic activity.

FAANG Rolling 60-Day Aggregations: Smoothed stock data to uncover long-term patterns in Open, High, Low, Close, and Volume.

Pivot Table for FAANG Comparison: Compared major tech stocks, highlighting price levels and trading volumes.

Amazon’s Q4 2018 Z-Scores: Standardized values to track deviations and detect performance shifts.

Event-Driven Stock Analysis: Merged Facebook’s major events (Cambridge Analytica, FTC probe, user growth slowdown) with its stock data to analyze investor reactions.

Index Transformation: Converted stock prices into relative terms, revealing overall growth trends.

Visualization: Brought the insights to life with plots, making patterns clearer and more intuitive.