8.1.4 Data Analysis and 8.1.5 Supplementary Activity

1. With the earthquakes.csv file, select all the earthquakes in Japan with a magType of mb and a magnitude of 4.9 or greater.

```
import pandas as pd
earthquakes = pd.read_csv('/content/earthquakes.csv')


# Selecting earthquakes that occurred in Japan with magnitude type 'mb' and magnitude greater than or equal to 4.9
# Filter earthquakes dataframe to include only those with place containing 'Japan',
# magType equal to 'mb', and mag greater than or equal to 4.9
japan_earthquakes = earthquakes[(earthquakes['place'].str.contains('Japan')) &
                                (earthquakes['magType'] == 'mb') &
                                (earthquakes['mag'] >= 4.9)]
japan_earthquakes
```

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

Next steps: View recommended plots

The 'japan_earthquakes' dataset is what we get after we've sorted through earthquake data to just focus on ones that happened in Japan, use a specific way of measuring magnitude called 'mb', and are strong, with a magnitude of 4.9 or more. So, it's a list of the big earthquakes in Japan that fit these criteria.

2. Create bins for each full number of magnitude (for example, the first bin is 0-1, the second is 1-2, and so on) with a magType of ml and count how many are in each bin.

```
# Filter earthquakes DataFrame to include only those with magnitude type 'ml'
ml_earthquakes = earthquakes[earthquakes['magType'] == 'ml']

# Define bins for magnitude values (from 0 to 10)
bins = [i for i in range(11)]

"""
Count the occurrences of earthquakes within each bin of magnitude values,
using pd.cut() to categorize earthquakes into magnitude bins, and then counting
"""
earthquake_counts = pd.cut(ml_earthquakes['mag'], bins=bins, right=False).value_counts().sort_index()
earthquake_counts
```

```
[0, 1)     2072
[1, 2)     3126
[2, 3)      985
[3, 4)      153
[4, 5)        6
[5, 6)        2
[6, 7)        0
[7, 8)        0
[8, 9)        0
```

```
[9, 10)       0
Name: mag, dtype: int64
```

It provides a frequency distribution of earthquake magnitudes within specified bins.

3. Using the faang.csv file, group by the ticker and resample to monthly frequency. Make the following aggregations:

Mean of the opening price

Maximum of the high price

Minimum of the low price

Mean of the closing price

Sum of the volume traded

```
"""
Reading the FAANG data from a CSV file, parsing the 'date' column as dates,
and setting the 'date' column as the index of the DataFrame
"""
faang_data = pd.read_csv('/content/faang.csv', parse_dates=['date'], index_col='date')
# Grouping the FAANG data by ticker and resampling it on a monthly basis
monthly_faang = faang_data.groupby('ticker').resample('M')

# Defining the aggregation functions for the resampled data
aggregations = {
    'open': 'mean',    # Calculating the mean of opening prices
    'high': 'max',     # Finding the maximum high price
    'low': 'min',      # Finding the minimum low price
    'close': 'mean',   # Calculating the mean of closing prices
    'volume': 'sum'    # Summing up the volume traded
}

monthly_agg = monthly_faang.agg(aggregations) # Applying the aggregation functions to the resampled data
monthly_agg
```

ticker	date	open	high	low	close	volume
AAPL	2018-01-31	170.714690	176.6782	161.5708	170.699271	659679440
	2018-02-28	164.562753	177.9059	147.9865	164.921884	927894473
	2018-03-31	172.421381	180.7477	162.4660	171.878919	713727447
	2018-04-30	167.332895	176.2526	158.2207	167.286924	666360147
	2018-05-31	182.635582	187.9311	162.7911	183.207418	620976206
	2018-06-30	186.605843	192.0247	178.7056	186.508652	527624365
	2018-07-31	188.065786	193.7650	181.3655	188.179724	393843881
	2018-08-31	210.460287	227.1001	195.0999	211.477743	700318837
	2018-09-30	220.611742	227.8939	213.6351	220.356353	678972040
	2018-10-31	219.489426	231.6645	204.4963	219.137822	789748068
	2018-11-30	190.828681	220.6405	169.5328	190.246652	961321947
	2018-12-31	164.537405	184.1501	145.9639	163.564732	898917007
AMZN	2018-01-31	1301.377143	1472.5800	1170.5100	1309.010952	96371290
	2018-02-28	1447.112632	1528.7000	1265.9300	1442.363158	137784020
	2018-03-31	1542.160476	1617.5400	1365.2000	1540.367619	130400151
	2018-04-30	1475.841905	1638.1000	1352.8800	1468.220476	129945743
	2018-05-31	1590.474545	1635.0000	1546.0200	1594.903636	71615299
	2018-06-30	1699.088571	1763.1000	1635.0900	1698.823810	85941510
	2018-07-31	1786.305714	1880.0500	1678.0600	1784.649048	97629820
	2018-08-31	1891.957826	2025.5700	1776.0200	1897.851304	96575676
	2018-09-30	1969.239474	2050.5000	1865.0000	1966.077895	94445693
	2018-10-31	1799.630870	2033.1900	1476.3600	1782.058261	183228552
	2018-11-30	1622.323810	1784.0000	1420.0000	1625.483810	139290208
	2018-12-31	1572.922105	1778.3400	1307.0000	1559.443158	154812304
FB	2018-01-31	184.364762	190.6600	175.8000	184.962857	495655736
	2018-02-28	180.721579	195.3200	167.1800	180.269474	516621991
	2018-03-31	173.449524	186.1000	149.0200	173.489524	996232472
	2018-04-30	164.163557	177.1000	150.5100	163.810476	751130388
	2018-05-31	181.910509	192.7200	170.2300	182.930000	401144183
	2018-06-30	194.974067	203.5500	186.4300	195.267619	387265765
	2018-07-31	199.332143	218.6200	166.5600	199.967143	652763259
	2018-08-31	177.598443	188.3000	170.2700	177.491957	549016789
	2018-09-30	164.232895	173.8900	158.8656	164.377368	500468912
	2018-10-31	154.873261	165.8800	139.0300	154.187826	622446235
	2018-11-30	141.762857	154.1300	126.8500	141.635714	518150415
	2018-12-31	137.529474	147.1900	123.0200	137.161053	558786249
GOOG	2018-01-31	1127.200952	1186.8900	1045.2300	1130.770476	28738485
	2018-02-28	1088.629474	1174.0000	992.5600	1088.206842	42384105

	2018-03-31	1096.108095	1177.0500	980.6400	1091.490476	45430049
	2018-04-30	1038.415238	1094.1600	990.3700	1035.696190	41773275
	2018-05-31	1064.021364	1110.7500	1006.2900	1069.275909	31849196
	2018-06-30	1136.396190	1186.2900	1096.0100	1137.626667	32103642
	2018-07-31	1183.464286	1273.8900	1093.8000	1187.590476	31953386
	2018-08-31	1226.156957	1256.5000	1188.2400	1225.671739	28820379
	2018-09-30	1176.878421	1212.9900	1146.9100	1175.808947	28863199
	2018-10-31	1116.082174	1209.9600	995.8300	1110.940435	48496167
	2018-11-30	1054.971429	1095.5700	996.0200	1056.162381	36735570
	2018-12-31	1042.620000	1124.6500	970.1100	1037.420526	40256461
NFLX	**2018-01-31**	231.269286	286.8100	195.4200	232.908095	238377533
	2018-02-28	270.873158	297.3600	236.1100	271.443684	184585819
	2018-03-31	312.712857	333.9800	275.9000	312.228095	263449491
	2018-04-30	309.129529	338.8200	271.2239	307.466190	262064417
	2018-05-31	329.779759	356.1000	305.7300	331.536818	142051114
	2018-06-30	384.557595	423.2056	352.8200	384.133333	244032001
	2018-07-31	380.969090	419.7700	328.0000	381.515238	305487432
	2018-08-31	345.409591	376.8085	310.9280	346.257826	213144082
	2018-09-30	363.326842	383.2000	335.8300	362.641579	170832156
	2018-10-31	340.025348	386.7999	271.2093	335.445652	363589920
	2018-11-30	290.643333	332.0499	250.0000	290.344762	257126498
	2018-12-31	266.309474	298.7200	231.2300	265.302368	234304628

Next steps: View recommended plots

It reads monthly stock data from a CSV file (faang), aggregates it by ticker, and calculates monthly averages, maximums, minimums, and sums for open, high, low, close prices, and volume traded.

4. Build a crosstab with the earthquake data between the tsunami column and the magType column. Rather than showing the frequency count, show the maximum magnitude that was observed for each combination. Put the magType along the columns.

```
# Defining a function to calculate the maximum magnitude in a given array of values
def max_magnitude(values):
    return values.max()

"""
'tsunami' and 'magType' are assumed to be columns in the 'earthquakes'
    DataFrame representing tsunami occurrence and magnitude type respectively
'mag' is assumed to be a column in the 'earthquakes' DataFrame representing earthquake magnitude
'aggfunc=max_magnitude' specifies that the maximum magnitude function should be applied to aggregate the data
"""
ctmax_magnitude = pd.crosstab(earthquakes['tsunami'],
                              earthquakes['magType'],
                              values=earthquakes['mag'],
                              aggfunc=max_magnitude)

# Displaying the resulting crosstab
ctmax_magnitude
monthly_agg
```

ticker	date	open	high	low	close	volume
AAPL	2018-01-31	170.714690	176.6782	161.5708	170.699271	659679440
	2018-02-28	164.562753	177.9059	147.9865	164.921884	927894473
	2018-03-31	172.421381	180.7477	162.4660	171.878919	713727447
	2018-04-30	167.332895	176.2526	158.2207	167.286924	666360147
	2018-05-31	182.635582	187.9311	162.7911	183.207418	620976206
	2018-06-30	186.605843	192.0247	178.7056	186.508652	527624365
	2018-07-31	188.065786	193.7650	181.3655	188.179724	393843881
	2018-08-31	210.460287	227.1001	195.0999	211.477743	700318837
	2018-09-30	220.611742	227.8939	213.6351	220.356353	678972040
	2018-10-31	219.489426	231.6645	204.4963	219.137822	789748068
	2018-11-30	190.828681	220.6405	169.5328	190.246652	961321947
	2018-12-31	164.537405	184.1501	145.9639	163.564732	898917007
AMZN	2018-01-31	1301.377143	1472.5800	1170.5100	1309.010952	96371290
	2018-02-28	1447.112632	1528.7000	1265.9300	1442.363158	137784020
	2018-03-31	1542.160476	1617.5400	1365.2000	1540.367619	130400151
	2018-04-30	1475.841905	1638.1000	1352.8800	1468.220476	129945743
	2018-05-31	1590.474545	1635.0000	1546.0200	1594.903636	71615299
	2018-06-30	1699.088571	1763.1000	1635.0900	1698.823810	85941510
	2018-07-31	1786.305714	1880.0500	1678.0600	1784.649048	97629820
	2018-08-31	1891.957826	2025.5700	1776.0200	1897.851304	96575676
	2018-09-30	1969.239474	2050.5000	1865.0000	1966.077895	94445693
	2018-10-31	1799.630870	2033.1900	1476.3600	1782.058261	183228552
	2018-11-30	1622.323810	1784.0000	1420.0000	1625.483810	139290208
	2018-12-31	1572.922105	1778.3400	1307.0000	1559.443158	154812304
FB	2018-01-31	184.364762	190.6600	175.8000	184.962857	495655736
	2018-02-28	180.721579	195.3200	167.1800	180.269474	516621991
	2018-03-31	173.449524	186.1000	149.0200	173.489524	996232472
	2018-04-30	164.163557	177.1000	150.5100	163.810476	751130388
	2018-05-31	181.910509	192.7200	170.2300	182.930000	401144183
	2018-06-30	194.974067	203.5500	186.4300	195.267619	387265765
	2018-07-31	199.332143	218.6200	166.5600	199.967143	652763259
	2018-08-31	177.598443	188.3000	170.2700	177.491957	549016789
	2018-09-30	164.232895	173.8900	158.8656	164.377368	500468912
	2018-10-31	154.873261	165.8800	139.0300	154.187826	622446235
	2018-11-30	141.762857	154.1300	126.8500	141.635714	518150415
	2018-12-31	137.529474	147.1900	123.0200	137.161053	558786249
GOOG	2018-01-31	1127.200952	1186.8900	1045.2300	1130.770476	28738485
	2018-02-28	1088.629474	1174.0000	992.5600	1088.206842	42384105

	2018-03-31	1096.108095	1177.0500	980.6400	1091.490476	45430049
	2018-04-30	1038.415238	1094.1600	990.3700	1035.696190	41773275
	2018-05-31	1064.021364	1110.7500	1006.2900	1069.275909	31849196
	2018-06-30	1136.396190	1186.2900	1096.0100	1137.626667	32103642
	2018-07-31	1183.464286	1273.8900	1093.8000	1187.590476	31953386
	2018-08-31	1226.156957	1256.5000	1188.2400	1225.671739	28820379
	2018-09-30	1176.878421	1212.9900	1146.9100	1175.808947	28863199
	2018-10-31	1116.082174	1209.9600	995.8300	1110.940435	48496167
	2018-11-30	1054.971429	1095.5700	996.0200	1056.162381	36735570
	2018-12-31	1042.620000	1124.6500	970.1100	1037.420526	40256461
NFLX	**2018-01-31**	231.269286	286.8100	195.4200	232.908095	238377533
	2018-02-28	270.873158	297.3600	236.1100	271.443684	184585819
	2018-03-31	312.712857	333.9800	275.9000	312.228095	263449491
	2018-04-30	309.129529	338.8200	271.2239	307.466190	262064417
	2018-05-31	329.779759	356.1000	305.7300	331.536818	142051114
	2018-06-30	384.557595	423.2056	352.8200	384.133333	244032001
	2018-07-31	380.969090	419.7700	328.0000	381.515238	305487432
	2018-08-31	345.409591	376.8085	310.9280	346.257826	213144082
	2018-09-30	363.326842	383.2000	335.8300	362.641579	170832156
	2018-10-31	340.025348	386.7999	271.2093	335.445652	363589920
	2018-11-30	290.643333	332.0499	250.0000	290.344762	257126498
	2018-12-31	266.309474	298.7200	231.2300	265.302368	234304628

Next steps: View recommended plots

This calculates the maximum earthquake magnitude for each of tsunami occurrence and magnitude type, using crosstab. The resulting DataFrame provides insights into the relationship between earthquake magnitudes, tsunami occurrences, and magnitude measurement types.

5. Calculate the rolling 60-day aggregations of OHLC data by ticker for the FAANG data. Use the same aggregations as exercise no. 3.

```
"""
Grouping the data by ticker and applying a rolling window of 60 days
The '60D' parameter specifies a rolling window of 60 days
'ticker' is the column used for grouping
This creates a rolling object with a 60-day window for each group (ticker)
"""
rolling_agg = faang_data.groupby('ticker').rolling('60D').agg(aggregations)
rolling_agg
```

ticker	date	open	high	low	close	volume
AAPL	**2018-01-02**	166.927100	169.0264	166.0442	168.987200	25555934.0
	2018-01-03	168.089600	171.2337	166.0442	168.972500	55073833.0
	2018-01-04	168.480367	171.2337	166.0442	169.229200	77508430.0
	2018-01-05	168.896475	172.0381	166.0442	169.840675	101168448.0
	2018-01-08	169.324680	172.2736	166.0442	170.080040	121736214.0
...	**...**	...	...	...	...	...
NFLX	**2018-12-24**	283.509250	332.0499	233.6800	281.931750	525657894.0
	2018-12-26	281.844500	332.0499	231.2300	280.777750	520444588.0
	2018-12-27	281.070488	332.0499	231.2300	280.162805	532679805.0
	2018-12-28	279.916341	332.0499	231.2300	279.461341	521968250.0
	2018-12-31	278.430769	332.0499	231.2300	277.451410	476309676.0

1255 rows × 5 columns

Next steps: View recommended plots

This applies rolling aggregation to the faang_data DataFrame, grouping by ticker and analyzing data within 60-day rolling windows.

6. Create a pivot table of the FAANG data that compares the stocks. Put the ticker in the rows and show the averages of the OHLC and volume traded data.

```
"""
Creating a pivot table
Specifying 'ticker' as the index for rows in the pivot table
and 'mean' as the aggregation function, which will compute the mean value for each column.
"""
pivot_faang = pd.pivot_table(faang_data, index='ticker', aggfunc='mean')
pivot_faang
```

ticker	close	high	low	open	volume
AAPL	186.986218	188.906858	185.135729	187.038674	3.402145e+07
AMZN	1641.726175	1662.839801	1619.840398	1644.072669	5.649563e+06
FB	171.510936	173.615298	169.303110	171.454424	2.768798e+07
GOOG	1113.225139	1125.777649	1101.001594	1113.554104	1.742645e+06
NFLX	319.290299	325.224583	313.187273	319.620533	1.147030e+07

Next steps: View recommended plots

This generates a pivot table named pivot_faang from a DataFrame, aggregating data based on ticker and calculating the mean values for each ticker.

7. Calculate the Z-scores for each numeric column of Netflix's data (ticker is NFLX) using apply().

```
# Selecting data related to the ticker 'NFLX' from the FAANG dataset
faang_nflx = faang_data.loc[faang_data['ticker'] == 'NFLX']

# Normalizing the selected data (open, high, low, close) using z-score normalization
# Z-score normalization is applied column-wise using lambda function
faang_data = faang_nflx[['open',
                         'high',
                         'low',
                         'close']
                        ].apply(lambda x: x.sub(x.mean()).div(x.std()))

# Adding a new column 'ticker' with value 'NFLX' to the normalized data
faang_data['ticker'] = 'NFLX'

# Setting the index of the DataFrame to the 'ticker' column
faang_data = faang_data.set_index('ticker')

# Returning the normalized data for the ticker 'NFLX'
faang_data
```

	open	high	low	close
ticker				
NFLX	-2.500753	-2.516023	-2.410226	-2.416644
NFLX	-2.380291	-2.423180	-2.285793	-2.335286
NFLX	-2.296272	-2.406077	-2.234616	-2.323429
NFLX	-2.275014	-2.345607	-2.202087	-2.234303
NFLX	-2.218934	-2.295113	-2.143759	-2.192192
...	...	...	...	...
NFLX	-1.571478	-1.518366	-1.627197	-1.745946
NFLX	-1.735063	-1.439978	-1.677339	-1.341402
NFLX	-1.407286	-1.417785	-1.495805	-1.302664
NFLX	-1.248762	-1.289018	-1.297285	-1.292137
NFLX	-1.203817	-1.122354	-1.088531	-1.055420

251 rows × 4 columns

Next steps: View recommended plots

This filters data for Netflix (NFLX) stock, adding a ticker column and setting it as the index.

8. Add event descriptions: Create a dataframe with the following three columns: ticker, date, and event. The columns should have the following values:

ticker: 'FB' date: ['2018-07-25', '2018-03-19', '2018-03-20']

event: ['Disappointing user growth announced after close.', 'Cambridge Analytica story', 'FTC investigation']

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

Merge this data with the FAANG data using an outer join

```
import pandas as pd
faang_data = pd.read_csv('/content/faang.csv')
faang_data['date'] = pd.to_datetime(faang_data['date'])

faang_data.set_index('date')
```

	ticker	open	high	low	close	volume
date						
2018-01-02	FB	177.68	181.58	177.5500	181.42	18151903
2018-01-03	FB	181.88	184.78	181.3300	184.67	16886563
2018-01-04	FB	184.90	186.21	184.0996	184.33	13880896
2018-01-05	FB	185.59	186.90	184.9300	186.85	13574535
2018-01-08	FB	187.20	188.90	186.3300	188.28	17994726
...	...	...	...	...	...	...
2018-12-24	GOOG	973.90	1003.54	970.1100	976.22	1590328
2018-12-26	GOOG	989.01	1040.00	983.0000	1039.46	2373270
2018-12-27	GOOG	1017.15	1043.89	997.0000	1043.88	2109777
2018-12-28	GOOG	1049.62	1055.56	1033.1000	1037.08	1413772
2018-12-31	GOOG	1050.96	1052.70	1023.5900	1035.61	1493722

```
# Filter Facebook events data
fb_events = faang_data.loc[(faang_data['ticker'] == 'FB') & (faang_data['date'].isin(['2018-07-25', '2018-03-19', '201

# Create DataFrame to store Facebook events
fb_events_df = pd.DataFrame(columns=['date', 'ticker', 'event'])

# Populate DataFrame with relevant data
fb_events_df['date'] = fb_events['date']
fb_events_df['ticker'] = fb_events['ticker']

# Add event descriptions based on dates
fb_events_df.loc[faang_data['date'] == '2018-03-19', 'event'] = 'Disappointing user growth announced after close.'
fb_events_df.loc[faang_data['date'] == '2018-03-20', 'event'] = 'Cambridge Analytica story'
fb_events_df.loc[faang_data['date'] == '2018-07-25', 'event'] = 'FTC investigation'
```