In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
%matplotlib inline
```

1. (3 points) Walk through Problem 1 to 5 of Chapter 4 of Molin's book.

Molins book:

Question 1: With the earthquakes.csv file, select all the earthquakes in Japan with a magnitude of 4.9 or greater using the mb magnitude type.

Question 2: Create bins for each full number of earthquake magnitude (for instance, the first bin is (0, 1], the second is (1, 2], and so on) with the ml magnitude type and count how many are in each bin.

Question 3: Using the faang.csv file, group by the ticker and resample to monthly frequency. Make the following aggregations: a) Mean of the opening price

b) Maximum of the high price

c) Minimum of the low price

d) Mean of the closing price

e) Sum of the volume traded

Question 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 magnitude type along the columns

Question 5: Calculate the rolling 60-day aggregations of the OHLC data by ticker for the FAANG data. Use the same aggregations as exercise 3.

In [3]:
```python
quakes = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/earthquakes.csv')
faang = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/faang.csv', index_col='date', parse_dates=True)
quakes.head()
```

Out[3]:

	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

In [4]:
```python
#Question 1
quakes.query(
    "parsed_place == 'Japan' and magType == 'mb' and mag >= 4.9"
)[['mag', 'magType', 'place']]
```

Out[4]:

	mag	magType	place
1563	4.9	mb	293km ESE of Iwo Jima, Japan
2576	5.4	mb	37km E of Tomakomai, Japan
3072	4.9	mb	15km ENE of Hasaki, Japan
3632	4.9	mb	53km ESE of Hitachi, Japan

In [5]:
```python
#Question 2
quakes.query("magType == 'ml'").assign(
    mag_bin=lambda x: pd.cut(x.mag, np.arange(0, 10))
).mag_bin.value_counts()
```

Out[5]:

mag_bin	count
(1, 2]	3105
(0, 1]	2207
(2, 3]	862
(3, 4]	122
(4, 5]	2
(5, 6]	1
(6, 7]	0
(7, 8]	0
(8, 9]	0

dtype: int64

In [6]:
```python
#Question 3

faang.groupby('ticker').resample('1M').agg(
    {
        'open' : np.mean,
        'high' : np.max,
        'low' : np.min,
        'close' : np.mean,
        'volume' : np.sum
    }
)
```

	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
FB	**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
	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

In [7]:
```python
#Question 4
pd.crosstab(quakes.tsunami, quakes.magType, values=quakes.mag, aggfunc='max')
```

Out[7]:

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

In [8]:
```python
#Question 5
faang.groupby('ticker').rolling('60D').agg(
    {
        'open' : np.mean,
        'high' : np.max,
        'low' : np.min,
        'close' : np.mean,
        'volume' : np.sum
    }
)
```

```
<ipython-input-8-d5548c8d9c28>:2: FutureWarning: The provided callable <function mean at 0x788afdd34310> is currently using RollingGroupby.mean. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "mean" instead.
  faang.groupby('ticker').rolling('60D').agg(
<ipython-input-8-d5548c8d9c28>:2: FutureWarning: The provided callable <function max at 0x788afdd139a0> is currently using RollingGroupby.max. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "max" instead.
  faang.groupby('ticker').rolling('60D').agg(
<ipython-input-8-d5548c8d9c28>:2: FutureWarning: The provided callable <function min at 0x788afdd13ac0> is currently using RollingGroupby.min. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "min" instead.
  faang.groupby('ticker').rolling('60D').agg(
<ipython-input-8-d5548c8d9c28>:2: FutureWarning: The provided callable <function mean at 0x788afdd34310> is currently using RollingGroupby.mean. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "mean" instead.
  faang.groupby('ticker').rolling('60D').agg(
<ipython-input-8-d5548c8d9c28>:2: FutureWarning: The provided callable <function sum at 0x788afdd13370> is currently using RollingGroupby.sum. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "sum" instead.
  faang.groupby('ticker').rolling('60D').agg(
```

Out[8]:

ticker	date	open	high	low	close	volume
	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
AAPL	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
...	...	...	...	...	...	...
	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
NFLX	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

In [8]:

In [8]:

2. (2 points) Problem #4 and problem #6 of Chapter 5 of Molin's book, page 320

Molins Book:

Question 4: Make a line plot of the difference between the weekly maximum high price and the weekly minimum low price for Facebook. This should be a single line.

Question 6: Using matplotlib and pandas, create two subplots side-by-side showing the effect that after-hours trading has had on Facebook's stock prices:

a) The first subplot will contain a line plot of the daily difference between that day's opening price and the prior day's closing price (be sure to review the Working with time series data section of Chapter 4, Aggregating Pandas DataFrames, for an easy way to do this).

b) The second subplot will be a bar plot showing the net effect this had monthly, using resample().

c) Bonus #1: Color the bars according to whether there are gains in the stock price (green) or drops in the stock price (red).

d) Bonus #2: Modify the x-axis of the bar plot to show the three-letter abbreviation for the month.

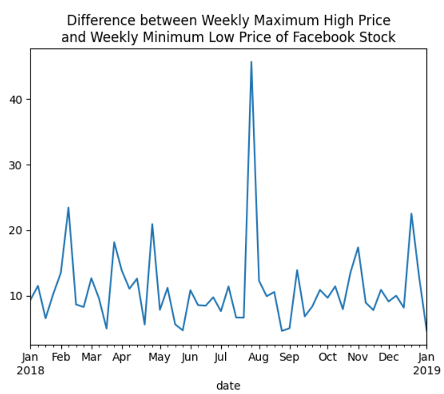
In [9]:
```python
fb = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/fb_stock_prices_2018.csv', index_col='date', parse_dates=True)
fb.head()
```

Out[9]:

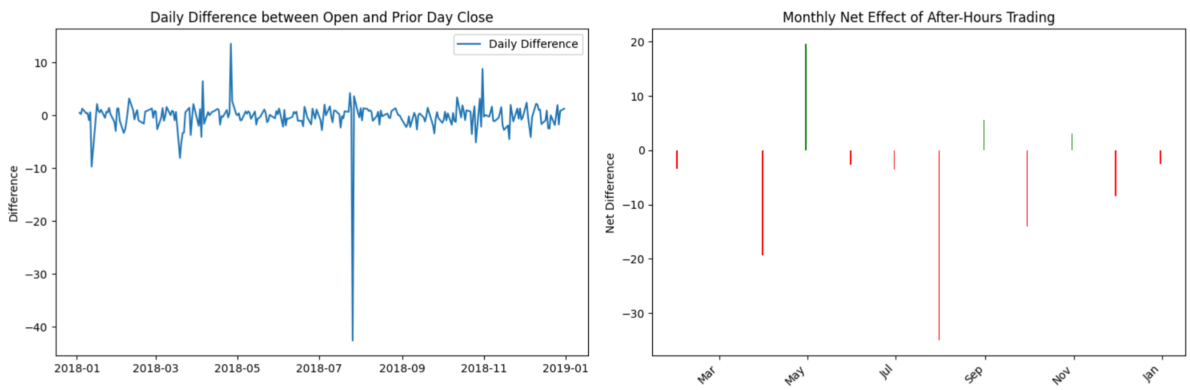
date	open	high	low	close	volume
2018-01-02	177.68	181.58	177.5500	181.42	18151903
2018-01-03	181.88	184.78	181.3300	184.67	16886563
2018-01-04	184.90	186.21	184.0996	184.33	13880896
2018-01-05	185.59	186.90	184.9300	186.85	13574535
2018-01-08	187.20	188.90	186.3300	188.28	17994726

In [10]:
```python
#Question 4

fb.resample('1W').agg(
    dict(high='max', low='min')
).assign(
    max_change_weekly=lambda x: x.high - x.low
).max_change_weekly.plot(
    title='Difference between Weekly Maximum High Price\nand Weekly Minimum Low Price of Facebook Stock'
)
```

Out[10]: <Axes: title={'center': 'Difference between Weekly Maximum High Price\nand Weekly Minimum Low Price of Facebook Stock'}, xlabel='date'>



In [11]:
```python
#Question 6

# Part A) Calculate the daily difference between the opening price and the prior day's closing price
fb['prior_close'] = fb.close.shift()
fb['daily_diff'] = fb.open - fb.prior_close

# Create the subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# First subplot: Line plot of daily difference
axes[0].plot(fb.daily_diff, label='Daily Difference')
axes[0].set_title('Daily Difference between Open and Prior Day Close')
axes[0].set_ylabel('Difference')
axes[0].legend()


# Part B) Second subplot: Bar plot of monthly net effect
# Resample to monthly frequency and sum the daily differences
monthly_diff = fb.daily_diff.resample('1M').sum()

# Part C, D, Create the bar plot with color based on positive or negative difference
colors = ['green' if diff > 0 else 'red' for diff in monthly_diff]
axes[1].bar(monthly_diff.index, monthly_diff.values, color=colors)
axes[1].set_title('Monthly Net Effect of After-Hours Trading')
axes[1].set_ylabel('Net Difference')
# Modify x-axis to show three-letter month abbreviation
axes[1].xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter('%b'))
# Rotate x-axis labels for better readability
plt.setp(axes[1].get_xticklabels(), rotation=45, ha='right')

# Display the plot
plt.tight_layout()
plt.show()
```



#Question 3

3. (3 points) Use Pandas to find the flight numbers of all flights that:

a. Had an arrival delay of two or more hours.

b. Flew to Houston (IAH or HOU).

c. Were operated by United, American, or Delta.

d. Departed in summer (July, August, and September).

e. Arrived more than two hours late but didn't leave late.

f. Were delayed by at least an hour, but made up over 30 minutes in flight.

g. Departed between midnight and 6am (inclusive). Try to use between() to simplify the code needed to answer the previous challenges.

h. How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

i. Why is NA ^ 0 not missing? Why is NA | TRUE not missing? Why is FALSE & NA not missing? Can you figure out the general rule? (NA * 0 is a tricky counter example!)

In [12]:
```python
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/nycflights13.csv')
df.head()
```

Out[12]:

	Unnamed: 0	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	...	tailnum	origin	dest	air_time	distance	hour	minute	time_hour	season	total_delay
0	46590	2013	10	22	630.0	630	0.0	852.0	838	14.0	...	N612JB	JFK	MSY	183.0	1182	6	30	10/22/2013 6:00	fall	14.0
1	220488	2013	5	30	918.0	925	-7.0	1054.0	1110	-16.0	...	N4XJAA	LGA	STL	124.0	888	9	25	5/30/2013 9:00	spring	-23.0
2	226972	2013	6	6	827.0	835	-8.0	1003.0	1020	-17.0	...	N711MQ	LGA	CLE	72.0	419	8	35	6/6/2013 8:00	spring	-25.0
3	251790	2013	7	2	1201.0	1035	86.0	1521.0	1350	91.0	...	N3CWAA	LGA	MIA	175.0	1096	10	35	7/2/2013 10:00	summer	177.0
4	176650	2013	4	13	629.0	630	-1.0	832.0	832	0.0	...	N566UW	LGA	CLT	90.0	544	6	30	4/13/2013 6:00	spring	-1.0

5 rows × 22 columns

In [13]:
```python
#Part A)
delayed_flights = df[df['arr_delay'] >= 120]['flight'].unique()
print(f"Flights with 2+ hour arrival delay: {len(delayed_flights)}")

#Part B)
houston_flights = df[df['dest'].isin(['IAH', 'HOU'])]['flight'].unique()
print(f"Flights to Houston: {len(houston_flights)}")

#Part C)
carrier_flights = df[df['carrier'].isin(['UA', 'AA', 'DL'])]['flight'].unique()
print(f"Flights by United, American, or Delta: {len(carrier_flights)}")

#Part D)
summer_flights = df[df['month'].isin([7, 8, 9])]['flight'].unique()
print(f"Flights in Summer: {len(summer_flights)}")

#Part E)
late_arrival_flights = df[(df['arr_delay'] > 120) & (df['dep_delay'] <= 0)]['flight'].unique()
print(f"Flights with late arrival but not late departure: {len(late_arrival_flights)}")

#Part G)
midnight_flights = df[df['dep_time'].between(0, 600, inclusive="both")]['flight'].unique()
print(f"Flights departing between midnight and 6am: {len(midnight_flights)}")

#Part H)
missing_dep_time = df['dep_time'].isnull().sum()
print(f"Flights with missing dep_time: {missing_dep_time}")

missing_cols = df.columns[df.isnull().any()].tolist()
print(f"Columns with missing values: {missing_cols}")
```

```
Flights with 2+ hour arrival delay: 811
Flights to Houston: 374
Flights by United, American, or Delta: 1415
Flights in Summer: 1642
Flights with late arrival but not late departure: 2
Flights departing between midnight and 6am: 318
Flights with missing dep_time: 1062
Columns with missing values: ['dep_time', 'dep_delay', 'arr_time', 'arr_delay', 'tailnum', 'air_time', 'total_delay']
```

###Part I

Question: Why is NA ^ 0 not missing? Why is NA | TRUE not missing? Why is FALSE & NA not missing?

The NA^0 It's still a true boolean value because there is a value there.

NA | TRUE: NA OR TRUE will always be TRUE, because TRUE overrides any other value in an OR operation.

FALSE & NA: FALSE AND NA will always be FALSE, because FALSE overrides any other value in an AND operation.

#Question 4

4. (2 point) Use Pandas to:

a. Sort all missing values to the start? (Hint: use isna())

b. Sort flights to find the most delayed flights. Find the flights that left earliest.

c. Sort flights to find the fastest flights.

d. Find which flights travelled the longest?

e. Which travelled the shortest?

In [14]:
```python
# a. Sort missing values to the start
df_sorted_missing = df.sort_values(by=df.columns.tolist(), na_position='first')
print("DataFrame with missing values sorted to the start:\n", df_sorted_missing.head())

# b. Sort by most delayed and earliest flights
most_delayed = df.sort_values(by=['dep_delay'], ascending=False)
earliest_flights = df.sort_values(by=['dep_time'])
print("\nMost delayed flights:\n", most_delayed[['flight', 'dep_delay']].head())
print("\nEarliest flights:\n", earliest_flights[['flight', 'dep_time']].head())

# c. Sort by fastest flights
fastest_flights = df.sort_values(by=['air_time'])
print("\nFastest flights:\n", fastest_flights[['flight', 'air_time']].head())

# d. Sort by longest flights
longest_flights = df.sort_values(by=['distance'], ascending=False)
print("\nLongest flights:\n", longest_flights[['flight', 'distance']].head())

# e. Sort by shortest flights
shortest_flights = df.sort_values(by=['distance'])
print("\nShortest flights:\n", shortest_flights[['flight', 'distance']].head())
```

```
DataFrame with missing values sorted to the start:
        Unnamed: 0  year  month  day  dep_time  sched_dep_time  dep_delay  \
28820           4  2013      1    1     554.0             600       -6.0
32934          10  2013      1    1     558.0             600       -2.0
6869           12  2013      1    1     558.0             600       -2.0
7867           14  2013      1    1     559.0             600       -1.0
234            18  2013      1    1     600.0             600        0.0

       arr_time  sched_arr_time  arr_delay  ... tailnum  origin dest air_time  \
28820     812.0             837      -25.0  ...  N668DN     LGA  ATL    116.0
32934     849.0             851       -2.0  ...  N793JB     JFK  PBI    149.0
6869      924.0             917        7.0  ...  N29129     JFK  LAX    345.0
7867      941.0             910       31.0  ...  N3DUAA     LGA  DFW    257.0
234       837.0             825       12.0  ...  N542MQ     LGA  ATL    134.0

      distance  hour  minute      time_hour  season total_delay
28820      762     6       0  1/1/2013 6:00  winter       -31.0
32934     1028     6       0  1/1/2013 6:00  winter        -4.0
6869      2475     6       0  1/1/2013 6:00  winter         5.0
7867      1389     6       0  1/1/2013 6:00  winter        30.0
234        762     6       0  1/1/2013 6:00  winter        12.0

[5 rows x 22 columns]

Most delayed flights:
        flight  dep_delay
11605    2363      800.0
22547    2042      798.0
27219    1485      753.0
7454     1091      687.0
10220     411      634.0

Earliest flights:
        flight  dep_time
32579     745       1.0
14565     363       1.0
35849    1503       1.0
26095     915       1.0
13458    4361       1.0

Fastest flights:
        flight  air_time
21590    2132      21.0
2239     4118      21.0
13285    4368      22.0
2320     4368      22.0
989      5670      22.0

Longest flights:
        flight  distance
21990      51      4983
30419      51      4983
12587      51      4983
39310      51      4983
10999      51      4983

Shortest flights:
        flight  distance
21059    4619        80
37636    4619        80
33644    4502        80
17144    4616        80
7681     4619        80
```

#Question 5

5. (2 points) Use Pandas to

a. Select dep_time, dep_delay, arr_time, and arr_delay from flights.

b. What happens if you include the name of a variable multiple times in a df.loc call?

c. What does the function isin() do? Use this function to create the new data frame below.

d. Use Pandas str.contains('TIME') to select the flights. What is the default setting of this Pandas function? Is it case sensitive or insensitive? How do you override the default setting?

In [15]:
```python
# a. Select columns
selected_cols = df[['dep_time', 'dep_delay', 'arr_time', 'arr_delay']]
print("Selected columns:\n", selected_cols.head())

# b. Demonstrate repeated column in df.loc
df_repeated = df.loc[:, ['dep_time', 'dep_delay', 'dep_time']]
print("\nDataFrame with repeated column:\n", df_repeated.head())

# c. Using isin()
filtered_df = df[df['carrier'].isin(['UA', 'AA', 'DL'])]
print("\nFiltered DataFrame using isin():\n", filtered_df.head())

# d. Using str.contains()
time_flights = df[df['dest'].str.contains('TIME')]
print("\nFlights with 'TIME' in 'dest' (case sensitive):\n", time_flights.head())

time_flights_insensitive = df[df['dest'].str.contains('TIME', case=False)]
print("\nFlights with 'TIME' in 'dest' (case insensitive):\n", time_flights_insensitive.head())
print('The default setting for str.contains() is case sensitive. To make it case insensitive, you can set the case parameter to False.')
```

```
Selected columns:
    dep_time  dep_delay  arr_time  arr_delay
0     630.0        0.0     852.0       14.0
1     918.0       -7.0    1054.0      -16.0
2     827.0       -8.0    1003.0      -17.0
3    1201.0       86.0    1521.0       91.0
4     629.0       -1.0     832.0        0.0

DataFrame with repeated column:
    dep_time  dep_delay  dep_time
0     630.0        0.0     630.0
1     918.0       -7.0     918.0
2     827.0       -8.0     827.0
3    1201.0       86.0    1201.0
4     629.0       -1.0     629.0

Filtered DataFrame using isin():
    Unnamed: 0  year  month  day  dep_time  sched_dep_time  dep_delay  \
1      220488  2013      5   30     918.0             925       -7.0
3      251790  2013      7    2    1201.0            1035       86.0
5      296785  2013      8   18    1749.0            1755       -6.0
8       82587  2013     11   30    1056.0            1100       -4.0
9      270634  2013      7   22    1356.0            1358       -2.0

   arr_time  sched_arr_time  arr_delay  ... tailnum  origin dest air_time  \
1    1054.0            1110      -16.0  ...  N4XJAA     LGA  STL    124.0
3    1521.0            1350       91.0  ...  N3CWAA     LGA  MIA    175.0
5    2049.0            2045        4.0  ...  N3HMAA     LGA  DFW    185.0
8    1347.0            1407      -20.0  ...  N380DA     JFK  SLC    268.0
9    1643.0            1649       -6.0  ...  N77261     EWR  LAX    323.0

  distance  hour  minute          time_hour  season total_delay
1      888     9      25    5/30/2013 9:00  spring       -23.0
3     1096    10      35    7/2/2013 10:00  summer       177.0
5     1389    17      55   8/18/2013 17:00  summer        -2.0
8     1990    11       0  11/30/2013 11:00    fall       -24.0
9     2454    13      58   7/22/2013 13:00  summer        -8.0

[5 rows x 22 columns]

Flights with 'TIME' in 'dest' (case sensitive):
 Empty DataFrame
Columns: [Unnamed: 0, year, month, day, dep_time, sched_dep_time, dep_delay, arr_time, sched_arr_time, arr_delay, carrier, flight, tailnum, origin, dest, air_time, distance, hour, minute, time_hour, season, total_delay]
Index: []

[0 rows x 22 columns]

Flights with 'TIME' in 'dest' (case insensitive):
 Empty DataFrame
Columns: [Unnamed: 0, year, month, day, dep_time, sched_dep_time, dep_delay, arr_time, sched_arr_time, arr_delay, carrier, flight, tailnum, origin, dest, air_time, distance, hour, minute, time_hour, season, total_delay]
Index: []

[0 rows x 22 columns]
The default setting for str.contains() is case sensitive. To make it case insensitive, you can set the case parameter to False.
```

#Question 6

a. Use Pandas to make the data tidy as shown in the table below.

carrier	dep_time	dest	dist	fl_date	fl_num	origin	weather	day_week	day_of_month	tail_num	flight status
OH	1455	JFK	184	1/1/2004	5935	BWI	0	4	1	N940CA	ontime
DH	1640	JFK	213	1/1/2004	6155	DCA	0	4	1	N405FJ	ontime
DH	1245	LGA	229	1/1/2004	7208	IAD	0	4	1	N695BR	ontime
DH	1709	LGA	229	1/1/2004	7215	IAD	0	4	1	N662BR	ontime

b. Currently dep_time and sched_dep_time are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

c. Compare air_time with arr_time - dep_time. What do you expect to see? What do you see? What do you need to do to fix it?

d. Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three numbers to be related?

e. Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for min_rank().

f. What does 1:3 + 1:10 return? Why?

In [16]:
```python
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/nycflights13.csv')
df_copy = df.copy()
df.columns
```

Out[16]:
```
Index(['Unnamed: 0', 'year', 'month', 'day', 'dep_time', 'sched_dep_time',
       'dep_delay', 'arr_time', 'sched_arr_time', 'arr_delay', 'carrier',
       'flight', 'tailnum', 'origin', 'dest', 'air_time', 'distance', 'hour',
       'minute', 'time_hour', 'season', 'total_delay'],
      dtype='object')
```

In [17]:
```python
#Part A)

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/nycflights13.csv')
df['fl_date'] = pd.to_datetime(df[['year', 'month', 'day']])
df = df.rename(columns={'distance': 'dist', 'flight': 'fl_num','season': 'weather','day': 'day_week','tailnum': 'tail_num'})
df['flight status'] = np.where(df['arr_delay'] <= 0, 'ontime', 'late')
df['day_week'] = df['fl_date'].dt.dayofweek + 1
season_mapping = {
    'Winter': 0,
    'Spring': 1,
    'Summer': 2,
    'Fall': 3,
}

season_mapping = {k.lower(): v for k, v in season_mapping.items()}

# Convert column values to lowercase and map
df['weather'] = df['weather'].str.lower().map(season_mapping)
df = df[['carrier', 'dep_time', 'dest', 'dist', 'fl_date', 'fl_num', 'origin', 'weather', 'day_week','tail_num','flight status']]
df.head()
```

Out[17]:

	carrier	dep_time	dest	dist	fl_date	fl_num	origin	weather	day_week	tail_num	flight status
0	B6	630.0	MSY	1182	2013-10-22	675	JFK	3	2	N612JB	late
1	AA	918.0	STL	888	2013-05-30	1855	LGA	1	4	N4XJAA	ontime
2	MQ	827.0	CLE	419	2013-06-06	4558	LGA	1	4	N711MQ	ontime
3	AA	1201.0	MIA	1096	2013-07-02	1879	LGA	2	2	N3CWAA	late
4	US	629.0	CLT	544	2013-04-13	1433	LGA	1	6	N566UW	ontime

In [18]:
```
df.columns
```
Out[18]:
```
Index(['carrier', 'dep_time', 'dest', 'dist', 'fl_date', 'fl_num', 'origin',
       'weather', 'day_week', 'tail_num', 'flight status'],
      dtype='object')
```

In [19]:
```
df_copy['arr_time'] = pd.to_numeric(df_copy['arr_time'], errors='coerce').fillna(0)
df['dep_time'] = pd.to_numeric(df['dep_time'], errors='coerce').fillna(0)
df_copy['sched_dep_time'] = pd.to_numeric(df_copy['sched_dep_time'], errors='coerce').fillna(0)
df_copy['dep_delay'] = pd.to_numeric(df_copy['dep_delay'], errors='coerce').fillna(0)
df_copy['air_time'] = pd.to_numeric(df_copy['air_time'], errors='coerce').fillna(0)
def time_to_minutes(time_str):
    """Converts a time string in HHMM format to minutes since midnight."""
    if pd.isnull(time_str):
        return time_str  # Handle missing values
    if isinstance(time_str, str):
        time_str = int(time_str)
    hours = time_str // 100
    minutes = time_str % 100
    return hours * 60 + minutes

# b. Convert dep_time and sched_dep_time to minutes since midnight
df['dep_time_minutes'] = df['dep_time'].apply(time_to_minutes)
df['sched_dep_time_minutes'] = df_copy['sched_dep_time'].apply(time_to_minutes)

# c. Compare air_time with arr_time - dep_time
df['arr_time_minutes'] = df_copy['arr_time'].apply(time_to_minutes)
df['flight_duration'] = df['arr_time_minutes'] - df['dep_time_minutes']
df['air_time_diff'] = df_copy['air_time'] - df['flight_duration']

# Inspect the differences
print(df['air_time_diff'].describe())
print(df[df['air_time_diff'] != 0])

# d. Compare dep_time, sched_dep_time, and dep_delay
df['dep_delay_calculated'] = df['dep_time_minutes'] - df['sched_dep_time_minutes']
df['dep_delay_diff'] = df_copy['dep_delay'] - df['dep_delay_calculated']

# Inspect the differences
print(df['dep_delay_diff'].describe())
print(df[df['dep_delay_diff'] != 0])
```

```
count    42097.000000
mean        61.127895
std        267.364705
min       -690.000000
25%        -29.000000
50%        -18.000000
75%         40.000000
max       1606.000000
Name: air_time_diff, dtype: float64
      carrier  dep_time dest  dist    fl_date  fl_num origin  weather  \
0          B6     630.0  MSY  1182 2013-10-22     675    JFK        3
1          AA     918.0  STL   888 2013-05-30    1855    LGA        1
2          MQ     827.0  CLE   419 2013-06-06    4558    LGA        1
3          AA    1201.0  MIA  1096 2013-07-02    1879    LGA        2
4          US     629.0  CLT   544 2013-04-13    1433    LGA        1
...       ...       ...  ...   ...        ...     ...    ...      ...
42092      AA     826.0  STL   888 2013-01-14    1855    LGA        0
42093      DL    1855.0  SFO  2586 2013-02-24    1967    JFK        0
42094      AA    1450.0  MIA  1089 2013-04-24    1769    JFK        1
```

In [20]:
```
#Part E)
df['dep_delay'] = df_copy['dep_delay']
df['delay_rank'] = df_copy['dep_delay'].rank(method='min', ascending=True)

# Select the 10 most delayed flights
most_delayed_flights = df[df['delay_rank'] <= 10]
most_delayed_flights.head(10)

#Part I)

#Usually 1:3 + 1:10 would be used as slices or ranges, and it would not provide element wise addition in Python like it does using this method in R.
```

Out[20]:

	carrier	dep_time	dest	dist	fl_date	fl_num	origin	weather	day_week	tail_num	flight status	dep_time_minutes	sched_dep_time_minutes	arr_time_minutes	flight_duration	air_time_diff	dep_delay_calculated	dep_delay_diff	dep_delay	delay_rank
1449	DL	1900.0	TPA	1010	2013-01-11	1435	LGA	0	5	N934DL	ontime	1140.0	1170	1353.0	213.0	-74.0	-30.0	0.0	-30.0	1.0
7565	EV	1147.0	BTV	266	2013-03-16	3267	EWR	0	6	N13903	ontime	707.0	729	769.0	62.0	-13.0	-22.0	0.0	-22.0	2.0
9647	DL	743.0	TPA	1010	2013-09-18	1109	LGA	2	3	N337NB	ontime	463.0	485	621.0	158.0	-17.0	-22.0	0.0	-22.0	2.0
10569	UA	700.0	IAH	1416	2013-10-22	261	LGA	3	2	N514UA	late	420.0	440	602.0	182.0	33.0	-20.0	0.0	-20.0	8.0
14787	AA	903.0	STL	888	2013-05-14	1855	LGA	1	2	N4WPAA	ontime	543.0	565	660.0	117.0	9.0	-22.0	0.0	-22.0	2.0
18743	B6	2110.0	FLL	1076	2013-06-01	383	LGA	1	6	N653JB	ontime	1270.0	1290	1428.0	158.0	-21.0	-20.0	0.0	-20.0	8.0
22424	B6	540.0	BOS	200	2013-04-05	380	EWR	1	5	N329JB	ontime	340.0	361	423.0	83.0	-39.0	-21.0	0.0	-21.0	6.0
32851	B6	2223.0	SYR	209	2013-09-14	1816	JFK	2	6	N274JB	ontime	1343.0	1365	1407.0	64.0	-23.0	-22.0	0.0	-22.0	2.0
33662	MQ	1908.0	RDU	431	2013-06-05	4569	LGA	1	3	N830MQ	ontime	1148.0	1169	1257.0	109.0	-40.0	-21.0	0.0	-21.0	6.0
36073	EV	2051.0	ILM	500	2013-10-01	4885	LGA	3	2	N752EV	ontime	1251.0	1270	1339.0	88.0	-22.0	-19.0	0.0	-19.0	10.0

#Question 7

7. (2 points) The height and weights data of a male sports team are provided in the file HeightWeightSportsTeam.csv.

a. Use two histograms of 8 bins to observe the distributions of both height and weight.

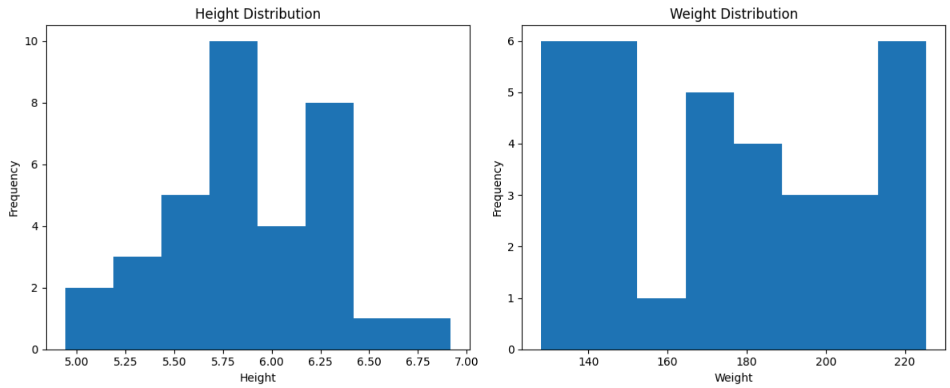
b. What are the possible distribution types of height and weight (e.g., normal or uniform)?

c. Use Q-Q plots to check how well they align with the data types you guessed above.

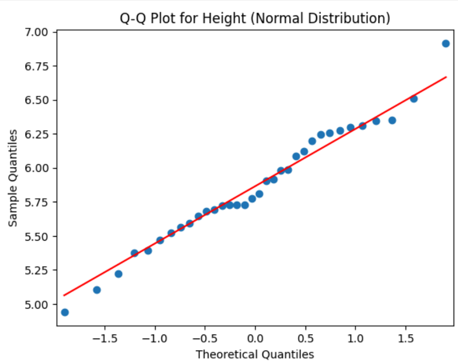
In [21]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/HeightWeightSportsTeam.csv')
df.head()
df.columns
```

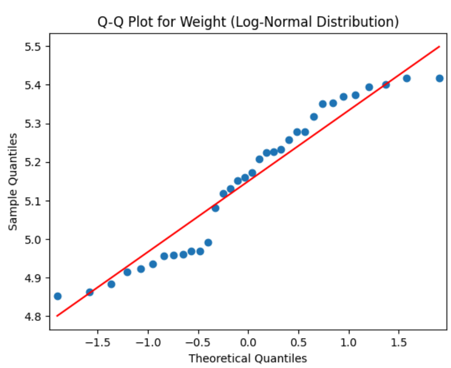
Out[21]:
```
Index(['Height', 'Weight '], dtype='object')
```

In [22]:
```python
#Part A)
df['Weight'] = df['Weight ']
df = df.drop(columns=['Weight '])
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)  # 1 row, 2 columns, first plot
plt.hist(df['Height'], bins=8)
plt.title('Height Distribution')
plt.xlabel('Height')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)  # 1 row, 2 columns, second plot
plt.hist(df['Weight'], bins=8)
plt.title('Weight Distribution')
plt.xlabel('Weight')
plt.ylabel('Frequency')

plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```



In [23]:
```python
#Part B & C)

# Height is usually distributed normally due to many environmental and genetic factors that people can't control much of. Most people are clustered around the average height.
# Weight is relatively the same answer; however, it can vary much more than height due to the variability of lifestyle choices as well as health conditions. A skewed distribution would represent this column best.
import statsmodels.api as sm
import pylab

sm.qqplot(df['Height'], line='s')
pylab.title('Q-Q Plot for Height (Normal Distribution)')
pylab.show()

# Q-Q plot for weight (assuming log-normal distribution)
sm.qqplot(np.log(df['Weight']), line='s') # Log-transforming weight
pylab.title('Q-Q Plot for Weight (Log-Normal Distribution)')
pylab.show()
```





#Question 8

8. (4 points) You may work with a friend to exchange ideas but complete your work in the end by yourself alone. Use Matplotlib and Pandas to explore the weekly moving average of the daily Page 3 of 5 changes in the new CovID-19 cases and deaths in California, Texas, Florida, New York, and Georgia from January 1st, 2021 (US_COVID-19_Cases_and_Deaths_by_State.csv).

a. First, sort the data by date (first column), select the new case column of the five states above from January 1st to the last full week. Call your resulting data frame as Corv_case_5StatesDF.

b. Cleanse the data Corv_case_5StatesDF to make the Corv_case_5StatesDF as tidy data.

c. Use rolling average to obtain the weekly moving average of daily change of both new cases and deaths and assign the new smoothened data frame as the same name.

d. Create a subplot(2, 0) for two rows and 1 column, and use five color lines to plot new cases in the top graph and five color lines to plot the deaths in the bottom graph.

e. Based on the graph, describe your comparison of the trends of cases and deaths in the five states.

f. Calculate the sum of total cases of the five states and total deaths of the five states, then identify the peaks and valleys of both curves, the time delay between the peaks of new cases and deaths, and delays of the valleys of the two curves.

g. Find the populations of the five states and divide them of each state by the Corv-19 cases of the moving curve above. Plot five lines to compare the cases per thousand residents.

b. During this period, New York and California had the most rigorous (strict) policies, such as mandatory masks and closing businesses, and Florida, Texas, and Georgia had the most business-friendly policies. Can you find clear evidence that the strict policies make a

In [24]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/US_COVID-19_Cases_and_Deaths_by_State.csv')
df.head()
df.columns
```

Out[24]:
```
Index(['submission_date', 'state', 'tot_cases', 'conf_cases', 'prob_cases',
       'new_case', 'pnew_case', 'tot_death', 'conf_death', 'prob_death',
       'new_death', 'pnew_death', 'created_at', 'consent_cases',
       'consent_deaths'],
      dtype='object')
```

In [25]:
```
#Part A & B)

states = ['CA', 'NY', 'TX', 'FL', 'GA']  # Replace with your desired states

# Filter data for selected states and date range
Corv_case_5StatesDF = df[df['state'].isin(states)][['submission_date', 'state', 'new_case','new_death']]
Corv_case_5StatesDF = Corv_case_5StatesDF.query("submission_date >= '01/01/2020' and submission_date <= '12/27/2020'")

# Convert 'submission_date' to datetime and sort
Corv_case_5StatesDF['submission_date'] = pd.to_datetime(Corv_case_5StatesDF['submission_date'], format='%m/%d/%Y')
Corv_case_5StatesDF = Corv_case_5StatesDF.sort_values(by=['submission_date'])

print(Corv_case_5StatesDF.head())
```

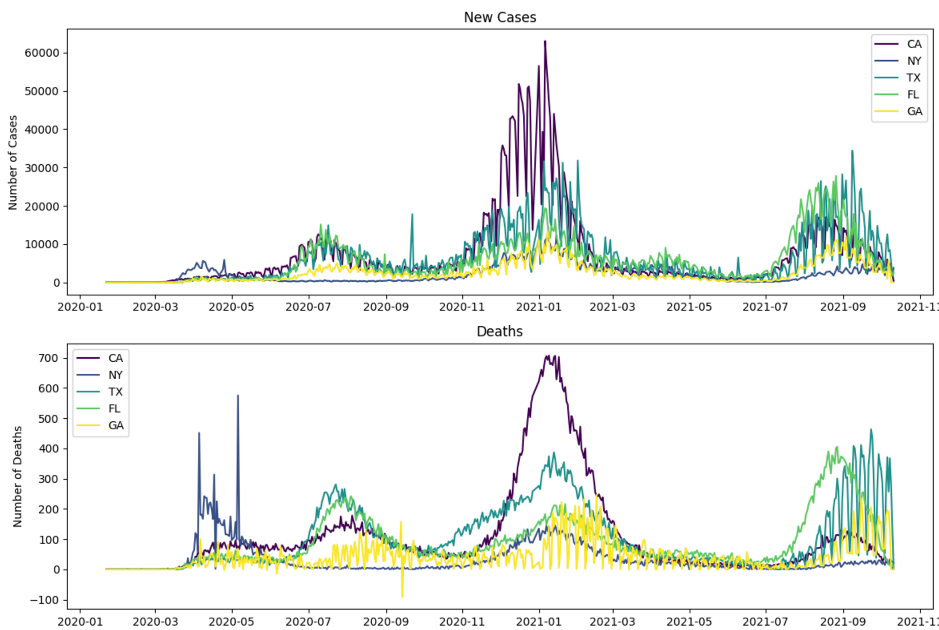
```
      submission_date state  new_case  new_death
15794      2020-01-22    FL         0          0
24738      2020-01-22    NY         0          0
11737      2020-01-22    GA         0          0
27034      2020-01-22    CA         0          0
21470      2020-01-22    TX         0          0
```

In [26]:
```
#Part C)
# Calculate the daily change for new cases
Corv_case_5StatesDF['daily_change_cases'] = Corv_case_5StatesDF.groupby('state')['new_case'].diff()
# Calculate the daily change for new deaths
Corv_case_5StatesDF['daily_change_deaths'] = Corv_case_5StatesDF.groupby('state')['new_death'].diff()

# Calculate the weekly moving average for new cases
Corv_case_5StatesDF['weekly_moving_avg_cases'] = Corv_case_5StatesDF.groupby('state')['daily_change_cases'].rolling(7).mean().reset_index(level=0, drop=True)
# Calculate the weekly moving average for new deaths
Corv_case_5StatesDF['weekly_moving_avg_deaths'] = Corv_case_5StatesDF.groupby('state')['daily_change_deaths'].rolling(7).mean().reset_index(level=0, drop=True)

# Fill NaN values with 0
Corv_case_5StatesDF['daily_change_cases'].fillna(0, inplace=True)
Corv_case_5StatesDF['daily_change_deaths'].fillna(0, inplace=True)
Corv_case_5StatesDF['weekly_moving_avg_cases'].fillna(0, inplace=True)
Corv_case_5StatesDF['weekly_moving_avg_deaths'].fillna(0, inplace=True)

print(Corv_case_5StatesDF.head())
```

```
      submission_date state  new_case  new_death  daily_change_cases  \
15794      2020-01-22    FL         0          0                 0.0
24738      2020-01-22    NY         0          0                 0.0
11737      2020-01-22    GA         0          0                 0.0
27034      2020-01-22    CA         0          0                 0.0
21470      2020-01-22    TX         0          0                 0.0

       daily_change_deaths  weekly_moving_avg_cases  weekly_moving_avg_deaths
15794                  0.0                      0.0                       0.0
24738                  0.0                      0.0                       0.0
11737                  0.0                      0.0                       0.0
27034                  0.0                      0.0                       0.0
21470                  0.0                      0.0                       0.0
```

In [27]:
```
#Part D)
fig, axes = plt.subplots(2, 1, figsize=(12, 8))  # 2 rows, 1 column

# Generate a larger color palette with enough colors for all states
num_states = len(states)
colors = plt.cm.get_cmap('viridis', num_states).colors

# Top graph: New cases
for i, state in enumerate(states):
    state_data = Corv_case_5StatesDF[Corv_case_5StatesDF['state'] == state]
    axes[0].plot(state_data['submission_date'], state_data['new_case'], color=colors[i], label=state)
axes[0].set_title('New Cases')
axes[0].set_ylabel('Number of Cases')
axes[0].legend()

# Bottom graph: Deaths
for i, state in enumerate(states):
    state_data = Corv_case_5StatesDF[Corv_case_5StatesDF['state'] == state]
    axes[1].plot(state_data['submission_date'], state_data['new_death'], color=colors[i], label=state)
axes[1].set_title('Deaths')
axes[1].set_ylabel('Number of Deaths')
axes[1].legend()

plt.tight_layout()
plt.show()
```

```
<ipython-input-27-109ebd65f849>:6: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.color
maps.get_cmap(obj)`` instead.
  colors = plt.cm.get_cmap('viridis', num_states).colors
```



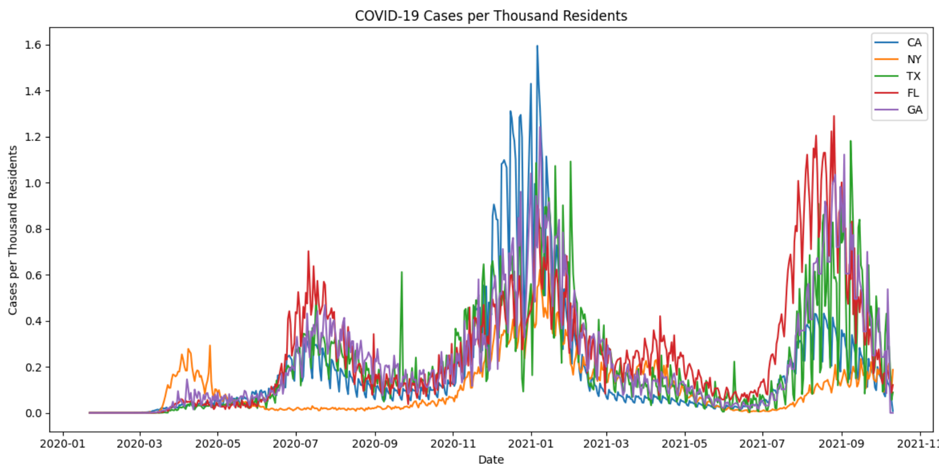
###Part E)

I think that the trends are relatively similar across all five states. The new cases directly impacts the number of deaths and they follow a very similar trend line. The biggest obvious difference is in New York around April and May. You can see that there weren't that many new cases emerging, but there were a ton of deaths in that state.

In [28]:
```
#Part F)

state_totals = Corv_case_5StatesDF.groupby('state').agg({'new_case': 'sum', 'new_death': 'sum'})

# Total cases and deaths for all five states
total_cases = state_totals['new_case'].sum()
total_deaths = state_totals['new_death'].sum()

print(f"Total Cases: {total_cases}")
print(f"Total Deaths: {total_deaths}")
```

```
Total Cases: 14972991
Total Deaths: 236454
```

In [29]:
```
#Part G)
state_populations = {
    'CA': 39538223,
    'NY': 20201249,
    'TX': 29145505,
    'FL': 21538187,
    'GA': 10711908
}
# Calculate cases per thousand residents
for state in states:
    Corv_case_5StatesDF.loc[Corv_case_5StatesDF['state'] == state, 'cases_per_thousand'] = (
        Corv_case_5StatesDF.loc[Corv_case_5StatesDF['state'] == state, 'new_case'] / state_populations[state] * 1000
    )

# Plotting
fig, ax = plt.subplots(figsize=(12, 6))

for state in states:
    state_data = Corv_case_5StatesDF[Corv_case_5StatesDF['state'] == state]
    ax.plot(state_data['submission_date'], state_data['cases_per_thousand'], label=state)

ax.set_title('COVID-19 Cases per Thousand Residents')
ax.set_xlabel('Date')
ax.set_ylabel('Cases per Thousand Residents')
ax.legend()

plt.tight_layout()
plt.show()
```



Part H:

During this period, New York and California had the most rigorous (strict) policies, such as mandatory masks and closing businesses, and Florida, Texas, and Georgia had the most business-friendly policies. Can you find clear evidence that the strict policies make a difference to mitigate the Corv-19 affection rates? Please provide your observations based on the plot above

Answer:

The early stages new york showed a very high increase in cases due to it being a very densely populated city, in my opinion. After policies were implemented to mitigate the spread of Covid-19, it seems to flatten out in the middle of the year and towards the end of the year it spiked back up around the Holidays. Usually the weather can impact the ability for viruses to spread as well since it's colder outside and immune systems are more susceptible for illness. While spiking even in the winter months, it still remained lower than all other states. Arguably, this comparison is biased due to NY having the second lowest number of residents on this graph. It's not as even of a comparison. It does provide a credible argument that the stricter policies were helping mitigate the spread of Covid if you are looking at the drastic difference between other states that have similar populations and their spread was much more out of control.

##Question 9

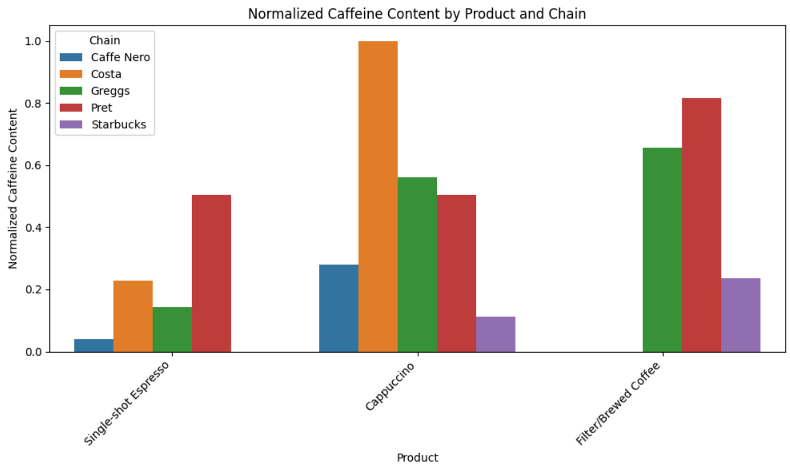
(4 points) The following visualization is about caffeine content in various coffees from different known chains in the UK. While the original tabular visualization with simple elements isn't bad, it can be significantly improved, as demonstrated in Will Sutton's example below. Sutton enhances the data by providing context alongside the horizontal bar chart, adhering to the principle of storytelling. He replaces the table with a clean, simple bar chart and focuses on a single drink, highlighting the Page 4 of 5 highest health risk by using dark red to emphasize espresso (following the principles of simplicity and robust design). Sutton guides the user through a narrative, whereas the original table lacks context and leaves unanswered questions. Now, consider an alternative approach to enhance the original visualization using the concepts and examples we've discussed in class. Think about how to present the data in a way that effectively communicates the story to your audience using Tableau. You can experiment with different types of visualizations depending on the narrative you want to build. Focus on simplicity, clarity, and highlighting key insights. The dataset, Coffee_Caffeine_Content.xlsx, is available in the folder. Remember, there's no right or wrong approach - it's about how well you present the story. Attach a screenshot of your visualization to the answer script. Hint: How about you normalize your data?

In [55]:
```
from sklearn.preprocessing import MinMaxScaler
df = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/M.S. Courses/DS 544 Data Viz/Datasets/Coffee_Caffeine_Content.xlsx')
df.head()
```

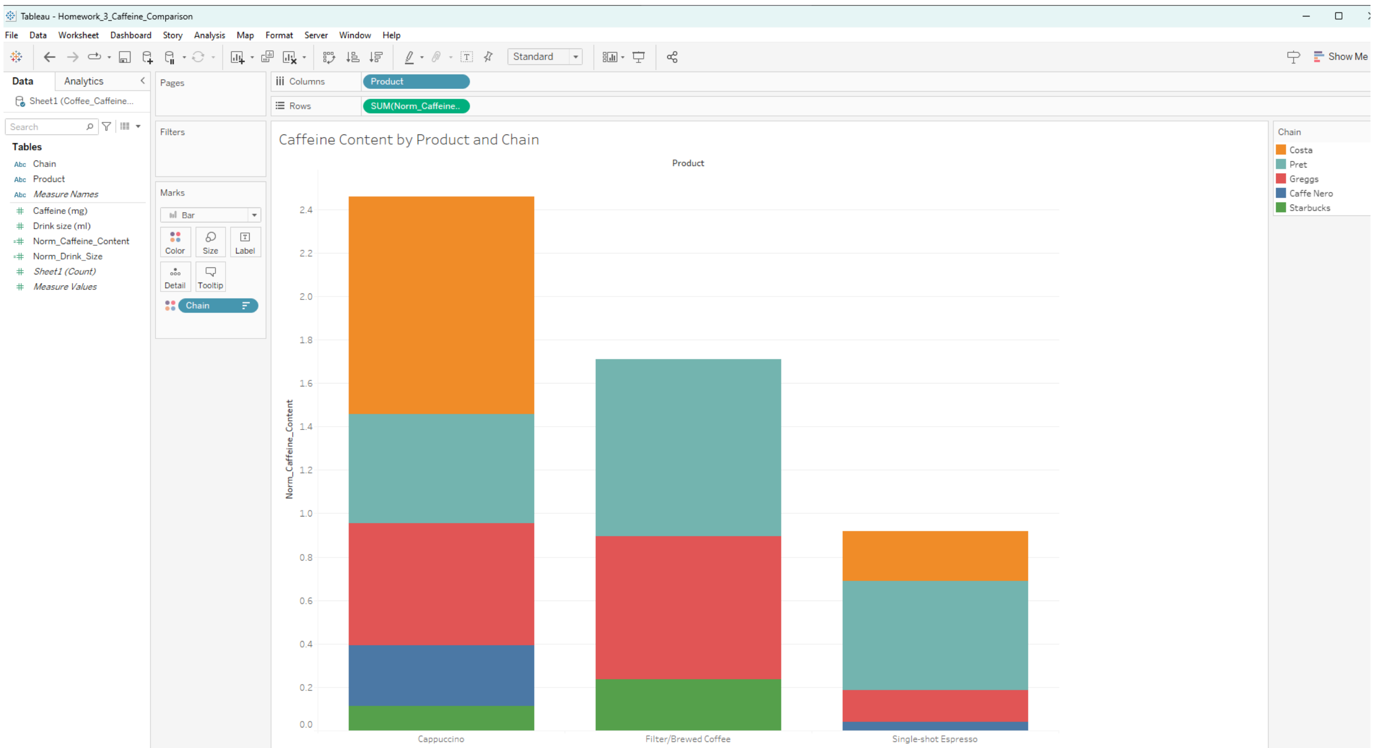
Out[55]:

	Chain	Product	Caffeine (mg)	Drink size (ml)
0	Caffe Nero	Single-shot Espresso	45	30
1	Caffe Nero	Cappuccino	115	355
2	Costa	Single-shot Espresso	100	30
3	Costa	Cappuccino	325	362
4	Greggs	Single-shot Espresso	75	28

In [56]:
```
columns_to_normalize = ['Caffeine (mg)', 'Drink size (ml)']
scaler = MinMaxScaler()
scaler.fit(df[columns_to_normalize])
df[['Norm_Caffeine_Content', 'Norm_Drink_Size']] = scaler.transform(df[columns_to_normalize])

plt.figure(figsize=(10, 6))  # Adjust figure size as needed
sns.barplot(x='Product', y='Norm_Caffeine_Content', hue='Chain', data=df)
plt.title('Normalized Caffeine Content by Product and Chain')
plt.xlabel('Product')
plt.ylabel('Normalized Caffeine Content')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels if needed
plt.tight_layout()
plt.show()
```



#Tableau Version



In [54]:

In [56]:

In [30]:

#References

https://www.cdc.gov/mmwr/volumes/69/wr/mm6915e4.htm (https://www.cdc.gov/mmwr/volumes/69/wr/mm6915e4.htm)

https://community.tableau.com/s/question/0D54T00000C6FnASAV/normalizing-the-data-in-tableau (https://community.tableau.com/s/question/0D54T00000C6FnASAV/normalizing-the-data-in-tableau)

In [30]: