Chapter 06 - Task 04 (chap 06 exercises)

```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

quakes = pd.read_csv('./data/earthquakes.csv')
```

Exercise 01

1. Using seaborn, create a heatmap to visualize the correlation coefficients between earthquake magnitude and whether there was a tsunami for earthquakes measured with the mb magnitude type.

```
# Exercise 01
sns.heatmap(
    # filter quakes to only consider magType = mb
    # of the matching quakes, get correlation between mag and tsnumami
    quakes.query('magType == "mb"')[['mag','tsunami']].corr(),
    # put numbers on each square; color the grid
    annot=True, cmap='coolwarm'
)
```

exercise 01

Exercise 02

Create a box plot of Facebook volume traded and closing prices, and draw reference lines for the bounds of a Tukey fence with a multiplier of 1.5. The bounds will be at Q1 − 1.5 × IQR and Q3 + 1.5 × IQR. Be sure to use the quantile() method on the data to make this easier. (Pick whichever orientation you prefer for the plot, but make sure to use subplots.)

```
fb = pd.read_csv('./data/fb_stock_prices_2018.csv')
fb_melted = pd.melt(fb[['volume', 'close']], var_name='Variables', value_name='Values')


# create the subplots
plt.figure(figsize=(12, 6))
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
# sns.boxplot(ax=axes[0], y=fb['Values'][fb['Variables'] == 'volume'])
# sns.boxplot(ax=axes[1], y=fb['Values'][fb['Variables'] == 'close'])

# populate plot for each column, add tukey fence lines
for ax, var in zip(axes, ['volume', 'close']):
```

```
    sns.boxplot(ax=ax, y=fb_melted['Values'][fb_melted['Variables'] == var])

    # create bounds to be used for tukey
    Q1, Q3 = fb[var].quantile(0.25), fb[var].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound, upper_bound = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR

    # draw lines for tukey fence bound
    [ax.axhline(bound, color='r', linestyle='--') for bound in [lower_bound, upper_bound]]
    # featuring unnecessary list comp
    ax.set_title(f'Box plot of Facebook {var.title()}')
    ax.set_xlabel(var.title())
    ax.set_ylabel('Values')
```

exercise 02

Exercise 03

> Plot the evolution of cumulative COVID-19 cases worldwide, and add a dashed vertical line on the date that it surpassed 1 million. Be sure to format the tick labels on the y-axis accordingly.

```
from matplotlib.ticker import EngFormatter

covid = pd.read_csv('./data/covid19_cases.csv')
covid['dateRep'] = pd.to_datetime(covid['dateRep'])
covid.set_index('dateRep', inplace=True)  # Setting dateRep as the index

# my method (without using Grouper) displays dates differently by default. i tried using
# an example from chapter 05 (monthly_ticks commented out below), but it doesnt cooperate
# with both cumulative_cases options i tested

# cumulative_cases = covid.groupby(pd.Grouper(freq='1D')).sum().cases.cumsum()
cumulative_cases = covid.groupby('dateRep')['cases'].sum().cumsum()

# they must be using a different data set because not only is the sampling much less freque
# atleast from what i see, it only really moves for them inbetween March and April
# (next cell has proof)

ax = cumulative_cases.plot(title='Cumulative COVID-19 Cases Worldwide\n(source: ECDC)', figs
ax.set_ylabel('cumulative COVID-19 cases')
# engineering notation, auto makes the numbers easier to read
ax.yaxis.set_major_formatter(EngFormatter())

over_1M = cumulative_cases[cumulative_cases >= 1e6].index[0]
ax.axvline(over_1M, linestyle='dashed', color='b', label=f"Reached 1M on {over_1M.strftime('
```

```
# monthly_ticks = pd.date_range(start=covid.index.min(), end=covid.index.max(), freq='MS')
# ax.set_xticks(monthly_ticks)
# ax.set_xticklabels(monthly_ticks.strftime('%b %Y'), rotation=45)

ax.legend()
```

exercise 03

```
# print(cumulative_cases.iloc[::10])
print(cumulative_cases.iloc[30:33])

# date when solution example says that 1m is reached
print(cumulative_cases.loc['2020-04-03'])
```