Homework 5

Alexandra Lansing 2/19/21

Problem 1: Faceted Histogram

Run the following code block to define a function which generates two 1-dimensional `numpy` arrays. The first array, called `groups`, consists of integers between `0` and `n_groups - 1`, inclusive. The second array, called `data`, consists of real numbers.

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt

def create_data(n, n_groups):
    """
    generate a set of fake data with group labels.
    n data points and group labels are generated.
    n_groups controls the number of distinct groups.
    Returns an np.array() of integer group labels and an
    np.array() of float data.
    """

    # random group assignments as integers between 0 and n_groups-1, inclusive
    groups = np.random.randint(0, n_groups, n)

    # function of the groups plus gaussian noise (bell curve)
    data   = np.sin(groups) + np.random.randn(n)

    return(groups, data)
```

Part A

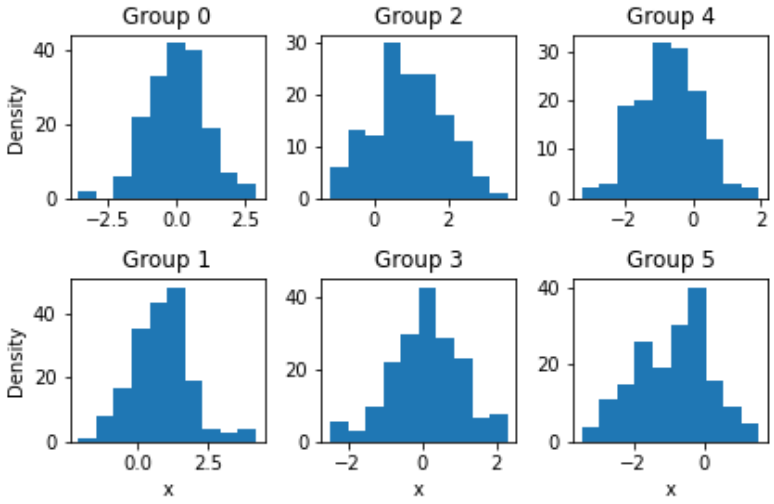
Write a function called `facet_hist()`. This function should accept five arguments:

1. `groups`, the `np.array` of group labels as output by `create_data()`.
2. `data`, the `np.array` of data as output by `create_data()`.
3. `m_rows`, the number of desired rows in your faceted histogram (explanation coming).
4. `m_cols`, the number of desired columns in your faceted histogram (explanation coming).
5. `figsize`, the size of the figure.

Your function will create faceted histograms -- that is, a separate axis and histogram for each group. For example, if there are six groups in the data, then you should be able to use the code

```
groups, data = create_data(1000, 6)
facet_hist(groups, data, m_rows = 2, m_cols = 3, figsize = (6,4))
```

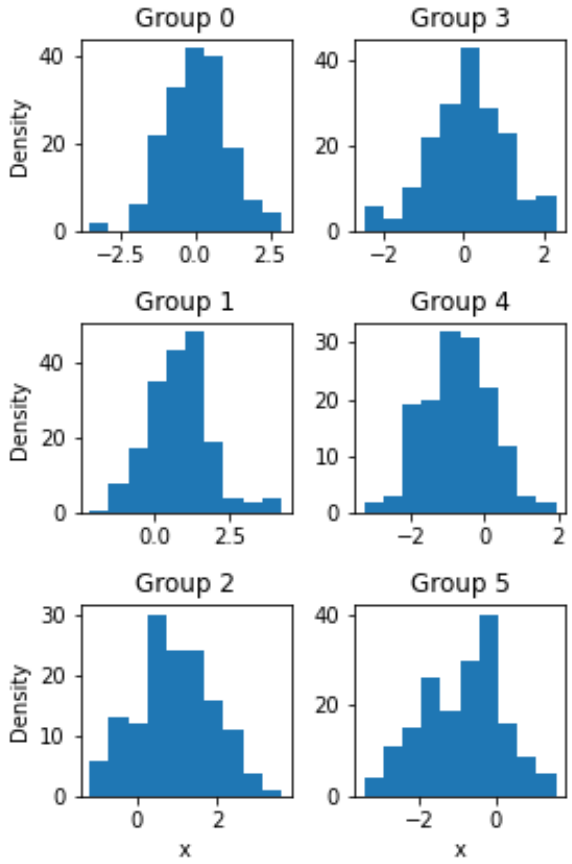
to create a plot like this:



It's fine if your group labels run left-to-right (so that the top row has labels 0, 1, and 2 rather than 0, 2, 4).

You should also be able to change the orientation by modifying `m_rows`, `m_cols`, and `figsize`.

```
facet_hist(groups, data, m_rows = 3, m_cols = 2, figsize = (4,6))
```



Requirements:

1. Your function should work **whenever `m_rows*m_cols` is equal to the total number of groups.** Your function should first check that this is the case, and raise an informative `ValueError` if not. You may assume that there is at least one data point for each group label in the data supplied.
2. For full credit, you should not loop over the individual entries of `groups` or `data`. It is acceptable to loop over the distinct values of `groups`. In general, aim to minimize `for`-loops and maximize use of `Numpy` indexing.
3. Use of `pandas` is acceptable but unnecessary, and is unlikely to make your solution significantly simpler.
4. You should include a horizontal axis label (of your choice) along **only the bottom row** of axes.
5. You should include a vertical axis label (e.g. "Frequency") along **only the leftmost column of axes.**
6. Each axis should have an axis title of the form "Group X", as shown above.
7. Comments and docstrings!

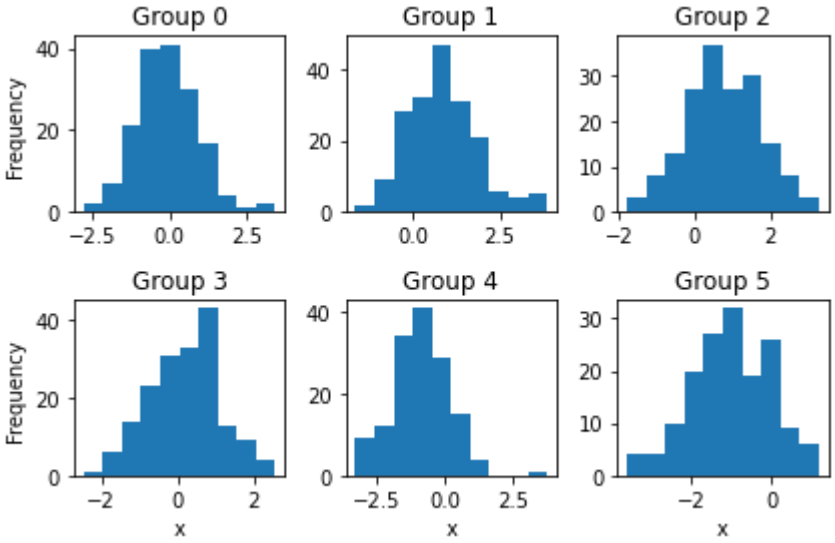
Hints

- If your plots look "squished," then `plt.tight_layout()` is sometimes helpful. Just call it after constructing your figure, with no arguments.
- Integer division `i // j` and remainders `i % j` are helpful here, although other solutions are also possible.

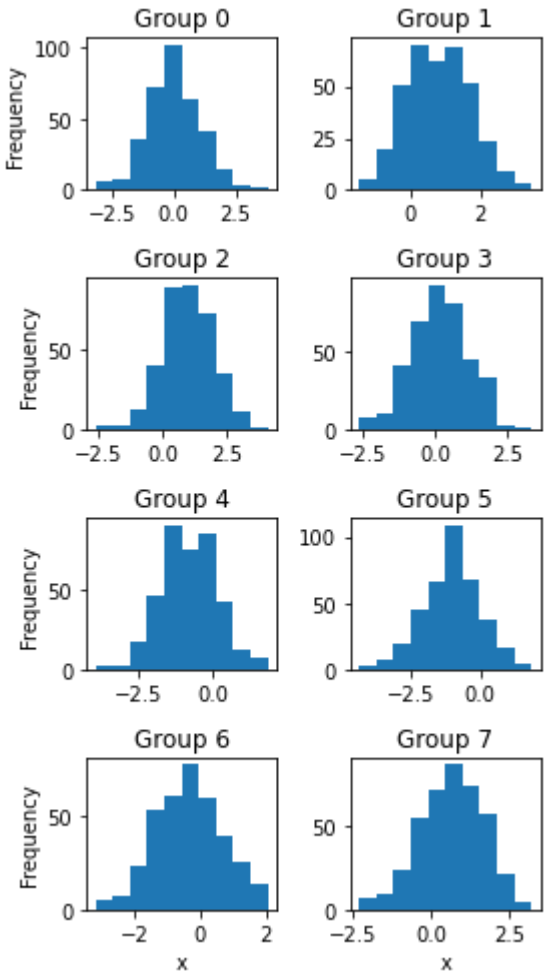
In [59]:
```
# your solution here
def facet_hist(groups, data, m_rows, m_cols, figsize, **kwargs):
    """
    Creates a histogram of data for every group in a set of m_rows by m_cols plots
    Takes in groups, a numpy array of groups from create_data function
             data, a numpy array of data generated from create_data function
             m_rows, the number of rows of plotted histograms in the faceted histogram
             m_cols, the number of columns of plotted histograms in the faceted histogr
             figsize, the overall size of the plot figures
             **kwargs argument for additional plot descriptions such as transparency an
    Returns None, just plots histograms
    """

    # Raise ValueError if row and column product doesn't equal number of  groups
    total_groups = np.max(groups)+1
    if m_rows*m_cols != total_groups:
        raise ValueError("Row and Column product is different than number of groups")

    # Create set of subplots in m_rows rows by m_cols columns, size figsize
    fig, ax = plt.subplots(m_rows, m_cols, figsize = figsize)

    # Keep track of total plots
    plot_num = 0

    # PLot histograms in set of m_rows by m_cols and index each to set title as group n
    for plot_r in range(m_rows):
        for plot_c in range(m_cols):
            ax[plot_r][plot_c].hist(data[groups==plot_num], **kwargs) # **kwargs for ad
            ax[plot_r][plot_c].set(title="Group "+ str(plot_num))
            plot_num += 1
```

```
    # x-axis labeled at bottom of all graphs
    for plot_bottom in range(m_cols):
        ax[m_rows-1][plot_bottom].set(xlabel = "x")

    # y-axis labeled at left side of all graphs
    for plot_side in range(m_rows):
        ax[plot_side][0].set(ylabel = "Frequency")

    # clean up plots if squished
    plt.tight_layout()
```

In [60]:
```
# test code
groups, data = create_data(1000, 6)
facet_hist(groups, data, 2, 3, figsize = (6, 4))
```



In [61]:
```
# test code
groups, data = create_data(3000, 8)
facet_hist(groups, data, 4, 2, figsize = (4, 7))
```

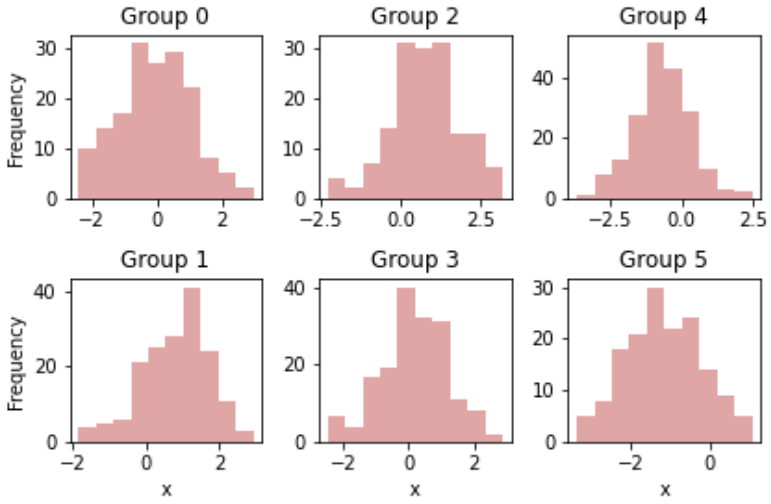


Part B

Modify your function (it's ok to modify it in place, no need for copy/paste) so that it accepts additional `**kwargs` passed to `ax.hist()`. For example,

```
facet_hist(groups, data, 2, 3, figsize = (6, 4), alpha = .4, color =
"firebrick")
```

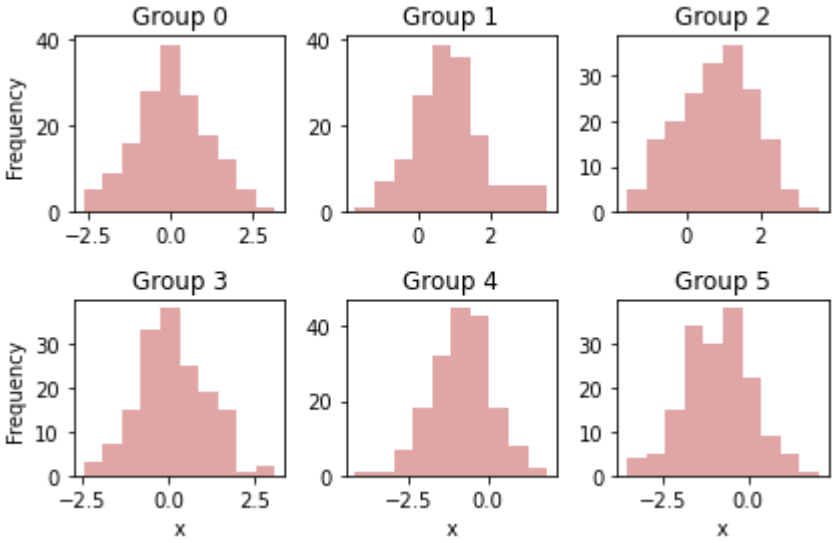
should produce



Example output.

You should be able to run this code **without defining parameters `alpha` and `color` for `facet_hist()`.**

In [5]:
```
# run this code to show that your modified function works
groups, data = create_data(1000, 6)
facet_hist(groups, data, 2, 3, figsize = (6, 4), alpha = .4, color = "firebrick")
```



Problem 2: Scatterplot Matrices

Run the following code to download, import, and display a data set from the 2019 World Happiness Report.

In [6]:
```
# if you experience ConnectionRefused errors, you may instead
# copy the url into your browser, save the file as data.csv
# in the same directory as the notebook, and then replace the
# third line with
# happiness = pd.read_csv("data.csv")

import pandas as pd
url = "https://philchodrow.github.io/PIC16A/datasets/world_happiness_report/2019.csv"
```

```
happiness = pd.read_csv(url)
happiness
```

Out[6]:

	Overall rank	Country or region	Score	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity	Perceptions of corruption
0	1	Finland	7.769	1.340	1.587	0.986	0.596	0.153	0.393
1	2	Denmark	7.600	1.383	1.573	0.996	0.592	0.252	0.410
2	3	Norway	7.554	1.488	1.582	1.028	0.603	0.271	0.341
3	4	Iceland	7.494	1.380	1.624	1.026	0.591	0.354	0.118
4	5	Netherlands	7.488	1.396	1.522	0.999	0.557	0.322	0.298
...	...	...	...	...	...	...	...	...	...
151	152	Rwanda	3.334	0.359	0.711	0.614	0.555	0.217	0.411
152	153	Tanzania	3.231	0.476	0.885	0.499	0.417	0.276	0.147
153	154	Afghanistan	3.203	0.350	0.517	0.361	0.000	0.158	0.025
154	155	Central African Republic	3.083	0.026	0.000	0.105	0.225	0.235	0.035
155	156	South Sudan	2.853	0.306	0.575	0.295	0.010	0.202	0.091

156 rows × 9 columns

This is a `pandas` data frame. Observe the following:

1. Each row corresponds to a country or region.
2. The `Score` column is the overall happiness score of the country, evaluated via surveys.
3. The other columns give indicators of different features of life in the country, including GDP, level of social support, life expectancy, freedom, generosity of compatriots, and perceptions of corruption in governmental institutions.

   You can extract each of these columns using dictionary-like syntax:

```
happiness["Score"]
```

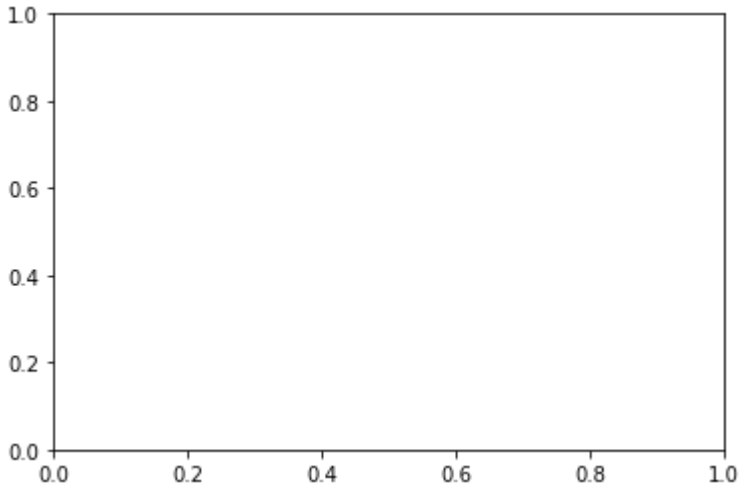
```
0      7.769
1      7.600
2      7.554
3      7.494
4      7.488
     ...
151    3.334
152    3.231
153    3.203
154    3.083
155    2.853
Name: Score, Length: 156, dtype: float64
```

Technically, this output is a `pandas` `Series`; however, in this context (and most others) it's fine to simply think of it as a 1-dimensional `np.array()`.

Part A

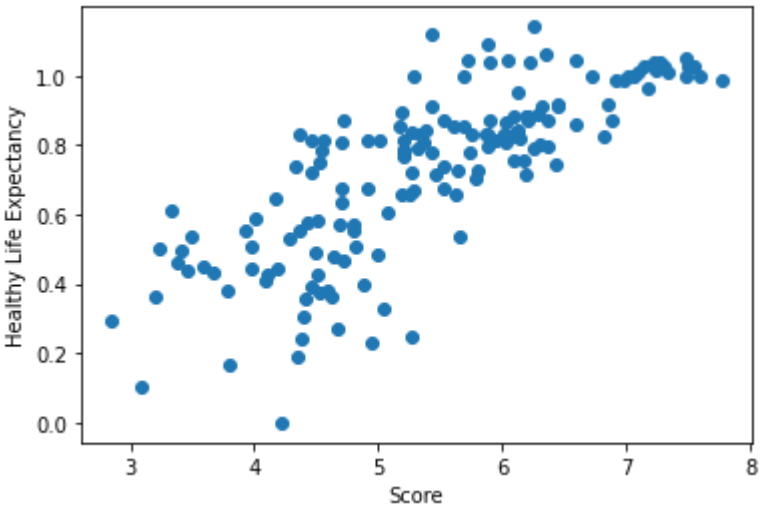
As a warmup, create a scatterplot of the overall `Score` column against a numerical column of your choice. Give the horizontal and vertical axes appropriate labels. Discuss your result. Is there a correlation? Does that correlation make sense to you?

In [7]:
```
fig, ax = plt.subplots(1)
```



In [8]:
```
# plotting code here
ax.scatter(happiness["Score"], happiness["Healthy life expectancy"])
ax.set(xlabel = "Score",
       ylabel = "Healthy Life Expectancy")
fig
```

Out[8]:



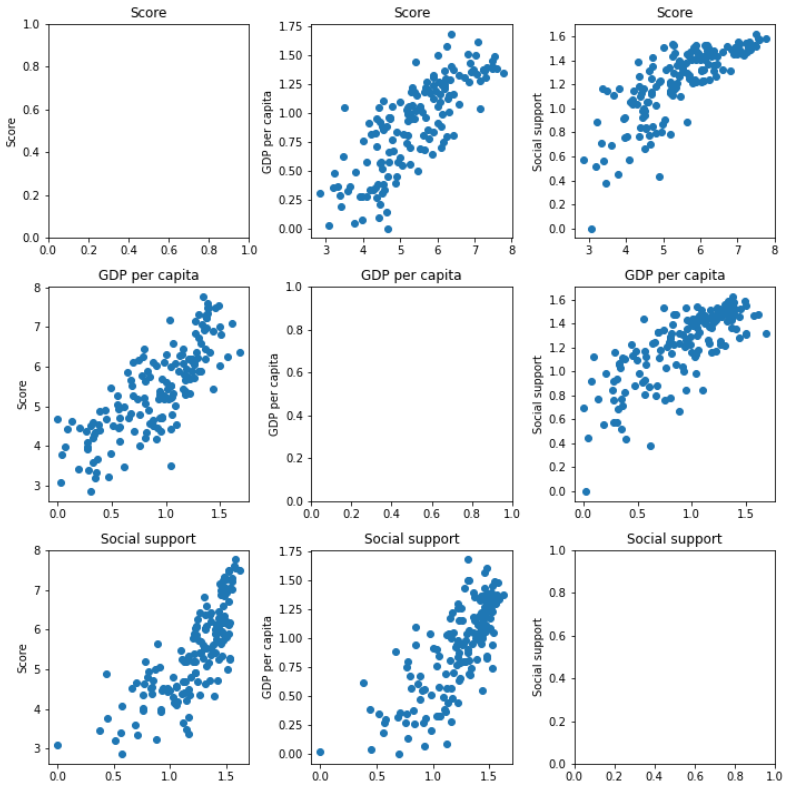
There appears to be a positive correlation between Score and Healthy Life Expectancy, where there is an upwards trend in the scatterplot data. This makes sense because a higher "happiness Score" means more happiness, and more happiness may lead to longer healthy life expectancies.

Part B

That plot you made may have helped you understand whether or not there's a relationship between the overall happiness score and the variable that you chose to plot. However, there are several variables in this data set, and we don't want to manually re-run the plot for each pair of variables. Let's see if we can get a more systematic view of the correlations in the data.

Write a function called `scatterplot_matrix()`, with arguments `cols` and `figsize`. The `cols` argument should be a list of strings, each of which are the name of one of the columns above, for example `cols = ["Score", "GDP per capita", "Social support"]`. Your function should create a *scatterplot matrix*, like this:

```
cols = ["Score",
        "GDP per capita",
        "Social support"]

scatterplot_matrix(cols,figsize = (7,7))
```



There is a separate scatterplot for each possible pair of variables. In fact, there are two: one where the first variable is on the horizontal axis, and one where it's on the vertical axis. Some analysts prefer to remove half the plots to avoid redundancy, but you don't have to bother with that. The diagonal is empty, since there's no point in investigating the relationship between a variable and itself.

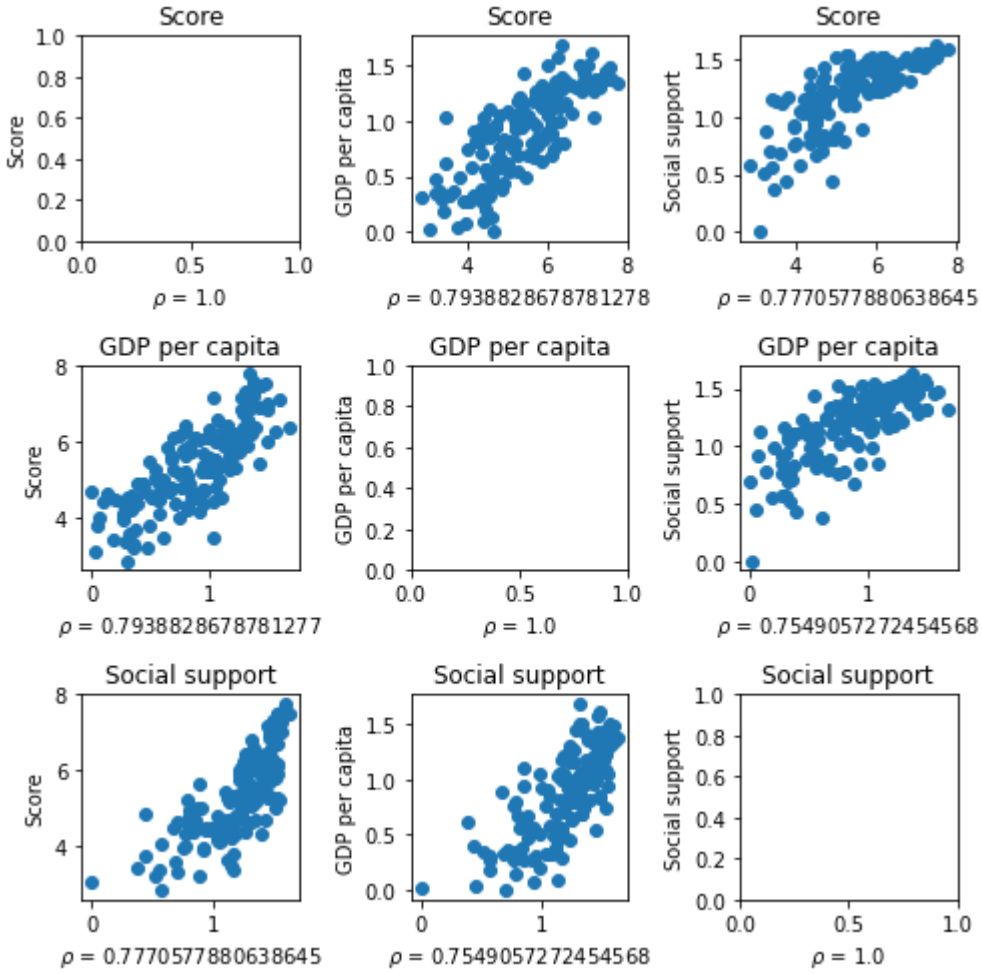
Don't forget comments and docstrings!

In [62]:
```
# define your function
def scatterplot_matrix(cols, figsize):
```

```
    """
    Create a cols x cols matrix of scatterplots with data from happiness data
    Takes in cols, a list of strings locating the specified columns of data in happines
             figsize, the overall size of the scatterplot figures
    Returns None, just plots scatterplots
    """

    # create figure with subplot matrix of cols x cols, sets plot sizes to figsize
    fig, ax = plt.subplots(len(cols), len(cols), figsize = figsize)

    # index each plot in subplot matrix
    for plot in range(len(cols)):
        for i in reversed(range(len(cols))):

            # don't plot the graphs of variables vs themselves (the diagonal)
            if plot != i:
                ax[plot, i].scatter(happiness[cols[plot]], happiness[cols[i]])

            # set corresponding x and y axis labels as the specified column titles from
            ax[plot, i].set(xlabel = cols[plot],
                            ylabel = cols[i])

            # clean up plots if squished
            plt.tight_layout()
    return
```

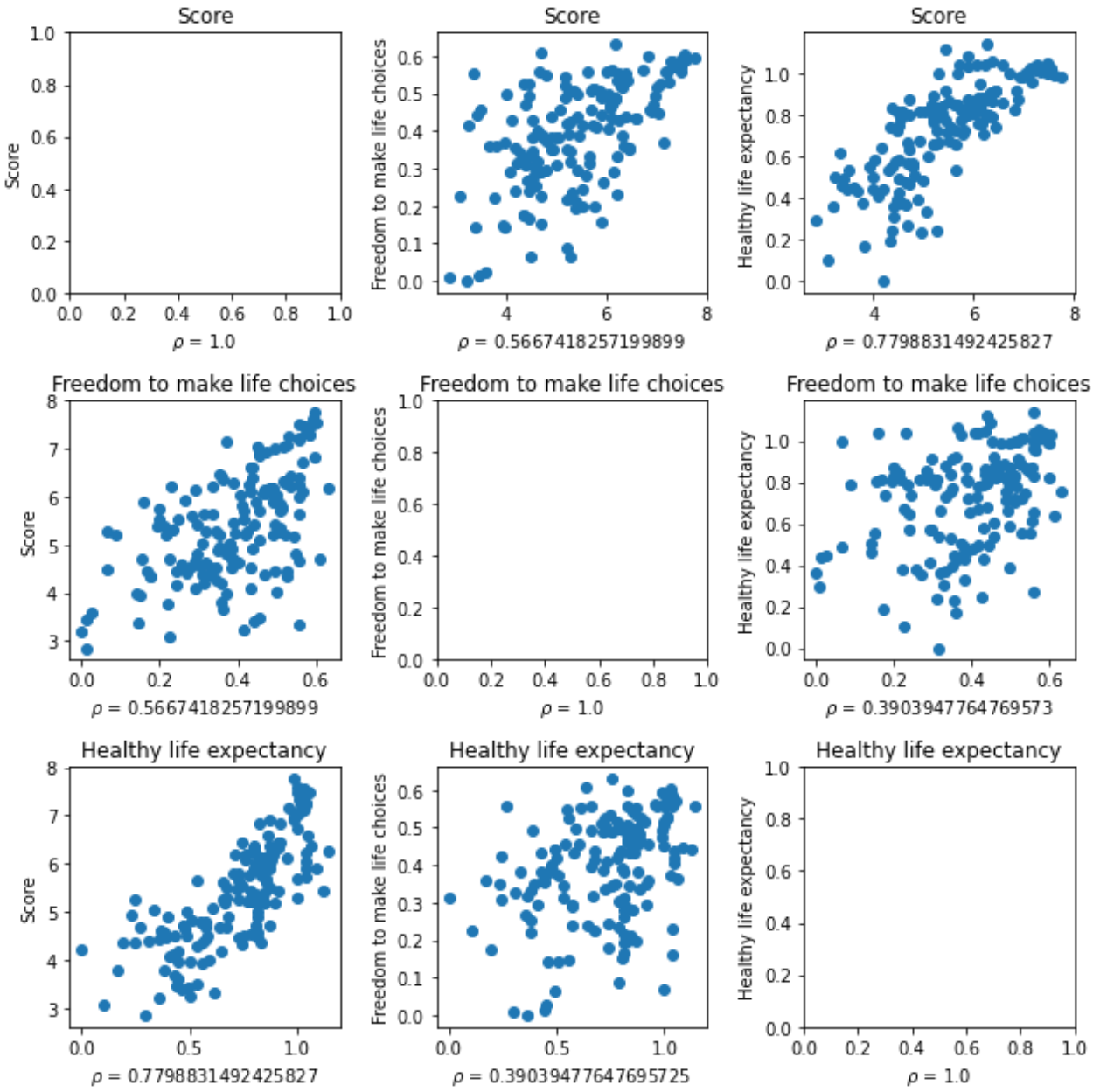
In [71]:

```
# test your code, several times if needed, and discuss the correlations you observe.
# Add code cells if needed to show multiple outputs.

cols = ["Score",
        "GDP per capita",
        "Social support"]

scatterplot_matrix(cols,figsize = (7,7))
```

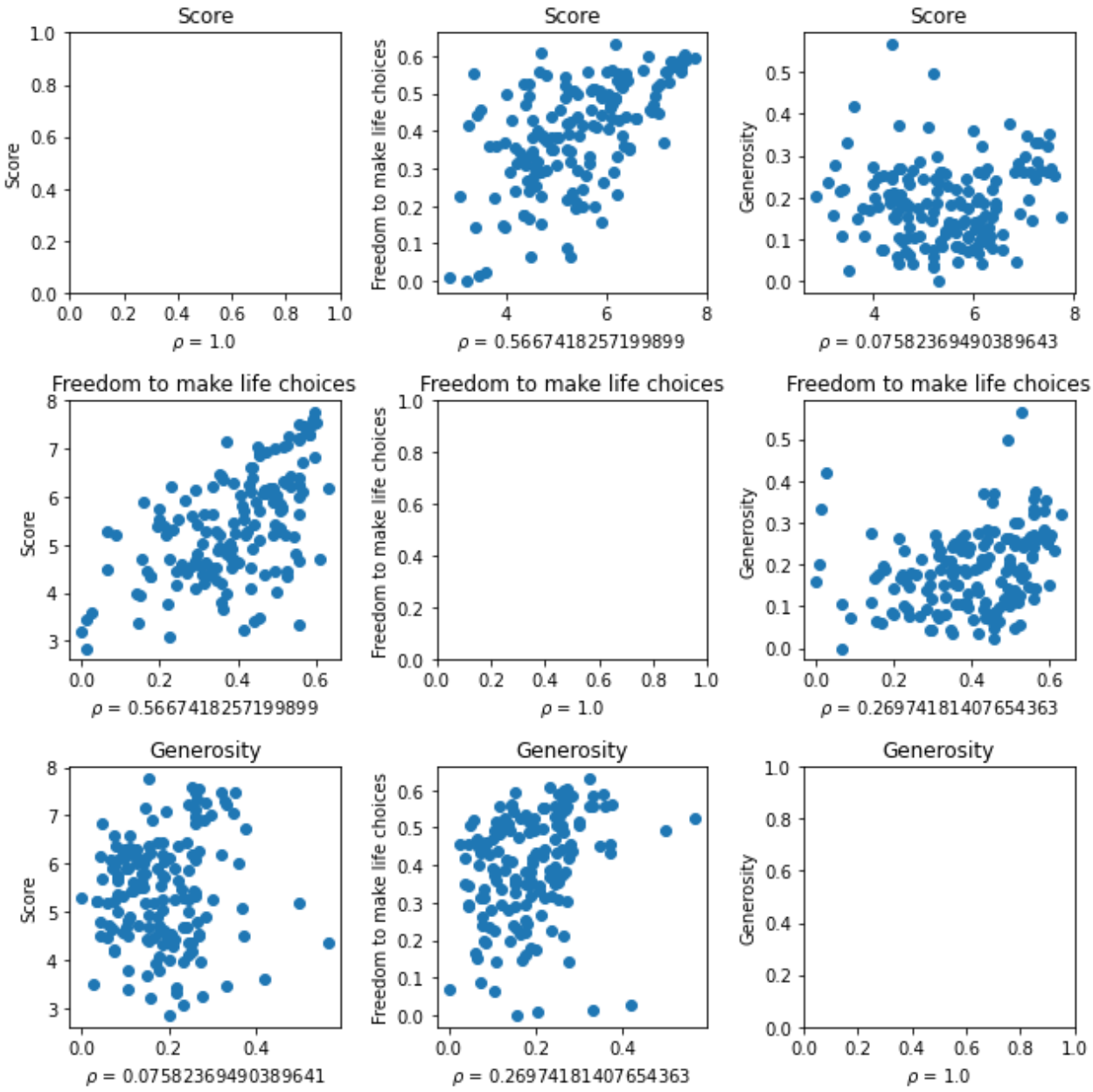


In [72]:

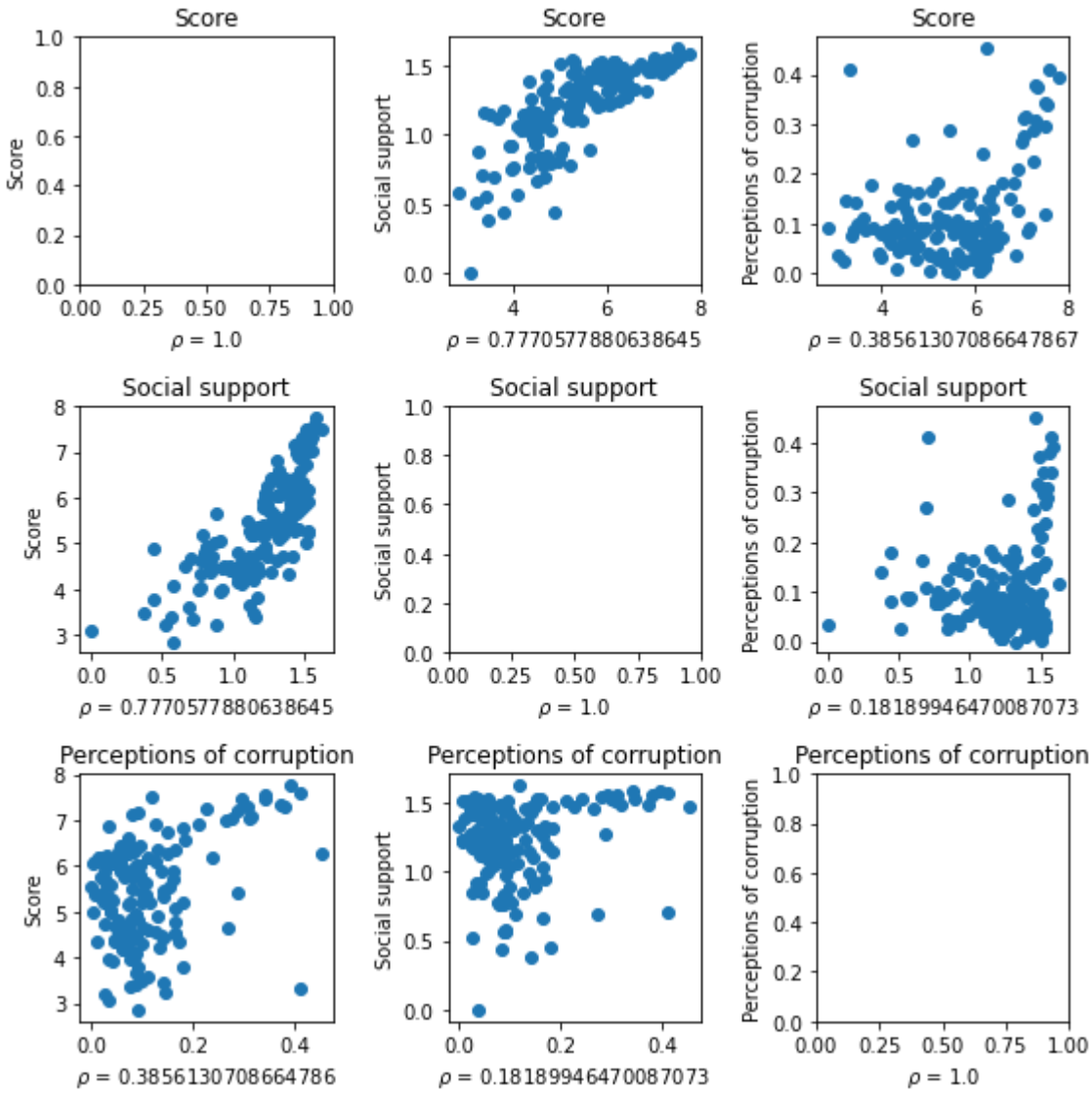
```
cols = ["Score",
        "Freedom to make life choices",
        "Healthy life expectancy"]

scatterplot_matrix(cols,figsize = (9,9))
```



In [73]:
```
cols = ["Score",
        "Freedom to make life choices",
        "Generosity"]

scatterplot_matrix(cols,figsize = (9,9))
```



In [74]:
```
cols = ["Score",
        "Social support",
        "Perceptions of corruption"]

scatterplot_matrix(cols,figsize = (8,8))
```

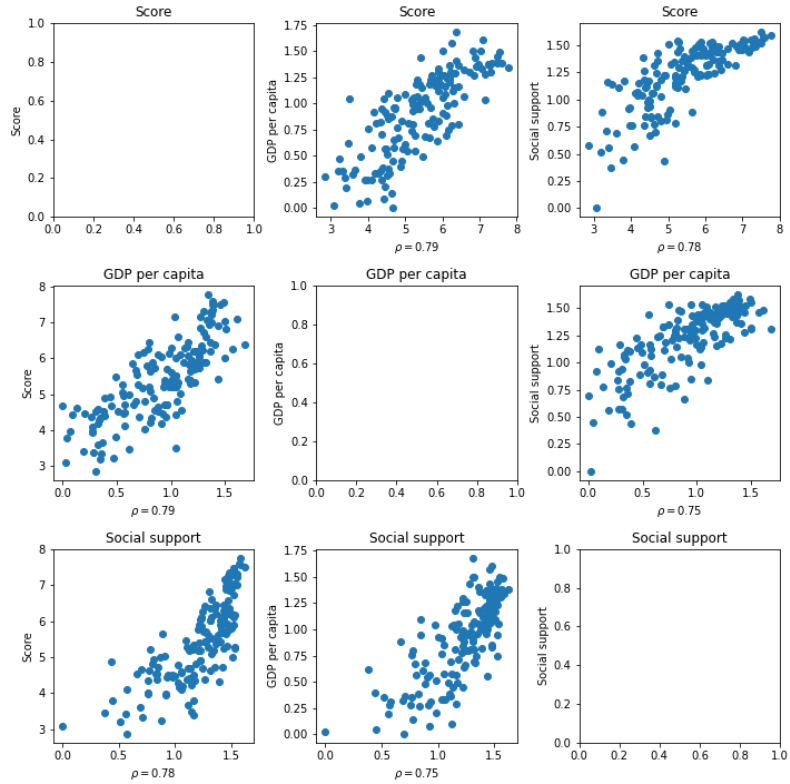


Part C

The *correlation coefficient* is a measure of linear correlation between two variables. The correlation coefficient between X and Y is high if X tends to be high when Y is, and vice versa. Correlation coefficients lie in the interval $[-1, 1]$.

`numpy` provides a function to conveniently compute the correlation coefficient between two or more variables. Find it, and then use it to add "captions" (as horizontal axis labels) to each panel of your plot giving the correlation coefficient between the plotted variables. For example,

```
cols = ["Score",
        "GDP per capita",
        "Social support"]

scatterplot_matrix(cols,figsize = (7,7))
```



It's not required that you add the Greek letter ρ (the classical symbol for correlation coefficients), but if you do want to, here's how. You can also tweak the rounding as desired.

```
ax.set(xlabel = r"$\rho$ = " + str(np.round(my_number, 2)))
```

In [70]:
```
def scatterplot_matrix(cols, figsize):
    '''
    Create a cols x cols matrix of scatterplots with data from happiness data
    Takes in cols, a list of strings locating the specified columns of data in happines
             figsize, the overall size of the scatterplot figures
    Returns None, just plots scatterplots
    '''
    # same code as Part B
    fig, ax = plt.subplots(len(cols), len(cols), figsize = figsize)
    for plot in range(len(cols)):
        for i in reversed(range(len(cols))):
            if plot != i:
                ax[plot, i].scatter(happiness[cols[plot]], happiness[cols[i]])

            # put the x-axis label on top instead as a title, so x-axis below becomes t
            ax[plot, i].set(title = cols[plot],
                            ylabel = cols[i],
                            # use np.corrcoef to calculate corr. coefficient between th
                            xlabel = r"$\rho$ = " + str(np.corrcoef(happiness[cols[plot
            plt.tight_layout()
    return
```

Run your code on several different subsets of the columns. It's ok to simply re-run your Part B results where they are and show the output including the correlation coefficient. Discuss your findings. What positive correlations do you observe? Do they make sense? Are there any negative correlations? Do the quantitative results match what you see "by eye"?

If you were going to create a model to attempt to predict overall happiness from other indicators, which columns would you use? Why?

Observations

In the four different subsets I ran, I plotted different combinations of all the variables other than region and overall rank. After plotting, it was clear that Score, GDP per capita, and Social Support all had a relatively stronger correlation, with a correlation coefficient of 0.79 (between Score and GDP), 0.78 (between Score and Soc Supp), and 0.75 (between GDP and Soc Supp). In another subset, it appeared that Score and Freedom to make life choices also had a positive, but weaker, correlation, with a correlation coefficient of 0.57. The correlation coefficients of these positive relationships make sense because they are closer to positive 1, meaning that when the X variable is bigger, the Y variable is bigger. Additionally, the positive relationships make sense between these variables because with a higher GDP, people presumably are happier in a more affluent economy (higher GDP, higher happiness score). And people must be happier with more Social support, as well as more Freedom to make life choices (higher Soc supp, higher Score/ higher Freedom, higher Score). Further, with a higher GDP, there is probably more Social support available (higher GDP, higher Soc supp), as seen in the first subset.

The other subsets I ran demonstrated less correlation. In my 3rd subset, Generosity's relationship with both Score and Freedom to make life choices had nearly no correlation, with a correlation coefficient of 0.07 (between Score and Generosity) and 0.27 (between Freedom to m.l.f. and Generosity). These correlation coefficients make sense because corr. coefficients closer to zero yield less correlation. This low correlation between the variables also makes sense because Generosity is most likely not affected by happiness nor freedom of choice (people can be selfish!). The Generosity data was also plotted on the lower end throughout the graph, demonstrating that people might just lack generosity overall.

My last subset showed the relationship between Perception of Corruption, Score, and Social Support. The relationship between Score and Social support was positive, as explained in the paragraph above. But Perception of Corruption had low correlation with the two other variables, with correlation coefficients 0.39 (between Score and Percep of Corrupt) and 0.18 (between Soc Supp and Percep of Corrupt). What was most interesting about these graphs, though, was although they had low correlation coefficients, there seemed to be a spike in Perception of Corruption when both Score and Social support were high. This might be explained by the fact that people may have the luxury to percieve more corruption when they're really comfortable, happy, and supported.

Problem 3: Plotting Time Series

Run the following code to download two time series data sets:

- Historical data on the Dow Jones Industrial Average (a composite performance measure of the US stock market), retrieved from Yahoo Finance.
- Cumulative COVID19 cases over time, from the New York Times.

In [28]:
```
# run this block
```

```
# if you experience ConnectionRefused errors, you may instead
# copy the urls into your browser, save the files as DJI.csv
# and COVID.csv respectively in the same directory as the notebook.
# Then, in the lines using the function pd.read_csv(), replace
# the url with "DJI.csv" and "COVID.csv"

import pandas as pd
import datetime

url = "https://query1.finance.yahoo.com/v7/finance/download/%5EDJI?period1=1580750232&p
DJI = pd.read_csv(url)
DJI['date'] = pd.to_datetime(DJI['Date'])
DJI = DJI.drop(["Date"], axis = 1)

url = "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us.csv"
COVID = pd.read_csv(url)
COVID['date'] = pd.to_datetime(COVID['date'])
```

Part A

The series `COVID['cases']` is essentially a `numpy` array containing the cumulative case counts over time. The COVID19 case data is cumulative, but we would like to see the number of new cases per day (i.e. as in this kind of plot). Check the documentation for the `np.diff` function and figure out what it does. Use it appropriately to construct a new array, called `per_day`, giving the number of new cases per day. Then, make a new array called `per_day_date` that gives the appropriate date for each case count. In particular, you will need to ensure that `per_day` and `per_day_date` have the same shape.

In [31]:
```
# your solution here
# np.diff returns array of the difference values between
# consecutive numbers in a numpy array

per_day = np.diff(COVID['cases'])
per_day_date = COVID['date'][1:395]
# 1 to 395 because the new cases reported pair up with the day after
# also ensures that both are same length

print(per_day.size)
print(per_day_date.size)
```

```
394
394
```

Part B

Create a figure with two very wide axes, one on top of the other (i.e. two rows, one column). Use the `sharex` argument of `plt.subplots()` to ensure that these two plots will share the same horizontal axis.

Then:

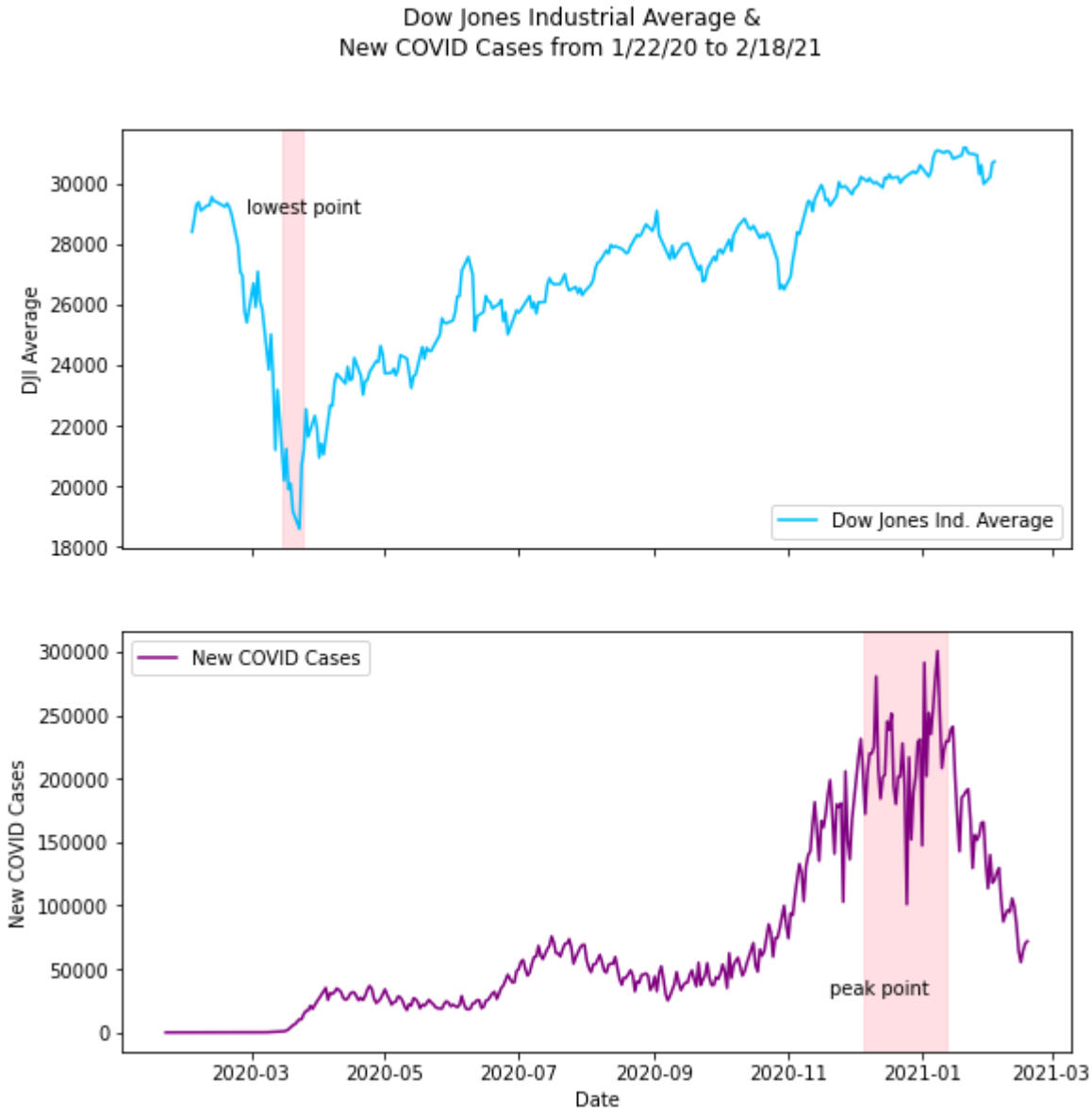
1. On the upper axis, plot the Dow Jones Industrial Average over time. For the horizontal axis use `DJI['date']`; the for the vertical use `DJI['Close']`.

2. On the lower axis, plot the variables `per_day_date` and `per_day` to visualize the progress of the COVID19 pandemic over time. Use a different color for the trendline.

Give your plot horizontal and vertical axis labels.

In [77]:
```
# your solution here
# modify this block in the remaining parts of the problem

# create 2 subplots that share x axis
fig, ax = plt.subplots(2, sharex = True, figsize = (9, 9))

# create one overall title, one x label below (for shared x axis)
fig.suptitle("Dow Jones Industrial Average &\nNew COVID Cases from 1/22/20 to 2/18/21")
plt.xlabel("Date")

#plot DJI averages vs date, plot Covid cases vs date
ax[0].plot(DJI['date'], DJI['Close'], color = "deepskyblue", )
ax[1].plot(per_day_date, per_day, color = "purple")

# highlighted region and text to show low of DJI averages
ax[0].axvspan(datetime.datetime(2020,3,15),
              datetime.datetime(2020,3,25),
              alpha = .5,
              color = "pink")
ax[0].text(datetime.datetime(2020,2,28),
           29000,
           "lowest point")

# highlighted region and text to show high of COVID cases
ax[1].axvspan(datetime.datetime(2020,12,5),
              datetime.datetime(2021,1,12),
              alpha = .5,
              color = "pink")
ax[1].text(datetime.datetime(2020,11,20),
           30000,
           "peak point")

# create legends and y-axis labels for both graphs
ax[0].legend(['Dow Jones Ind. Average'])
ax[0].axes.set_ylabel('DJI Average')

ax[1].legend(['New COVID Cases'])
ax[1].axes.set_ylabel('New COVID Cases')
```

Out[77]:
```
Text(0, 0.5, 'New COVID Cases')
```



Part C

The command

```
ax[0].axvspan(datetime.datetime(2020,6,1),
              datetime.datetime(2020,6,30),
              alpha = .3,
              color = "gray")
```

will add a simple rectangular shade which can be used to highlight specific portions of a time-series. In the given code, this shade runs through the month of June 2020. Add at least two such rectangular shades to your figure corresponding to important time intervals. You can put two shades on one axis, or one on each. If you're not sure what time periods are important, just choose intervals at random. Feel free to modify the color and transparency as desired. You can modify your figure code from Part B -- no need for copy/paste.

Part D

The command

```
ax[0].text(datetime.datetime(2020,9,15),
           22000,
           "penguins?\npenguins!")
```

will add a fun text annotation to your plot, with the first letter in horizontal position corresponding to September 15th, and at vertical position 22,000. Annotate each of your shaded regions with a few words describing their significance. Again, just modify your Part B code.

Part E

Add an overall title, spruce up your axis labels, and add anything else you think will make the plot look good. Again, you can just modify your Part B code, without copy/paste.

Then, submit a job application at www.FiveThirtyEight.com and show Nate Silver your cool data visualization.