Homework 5

Fill in your name and the names of any students who helped you below.

I affirm that I personally wrote the text, code, and comments in this homework assignment.

Austin Wuthrich 5/11/22

Comments are required for all parts of this homework

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

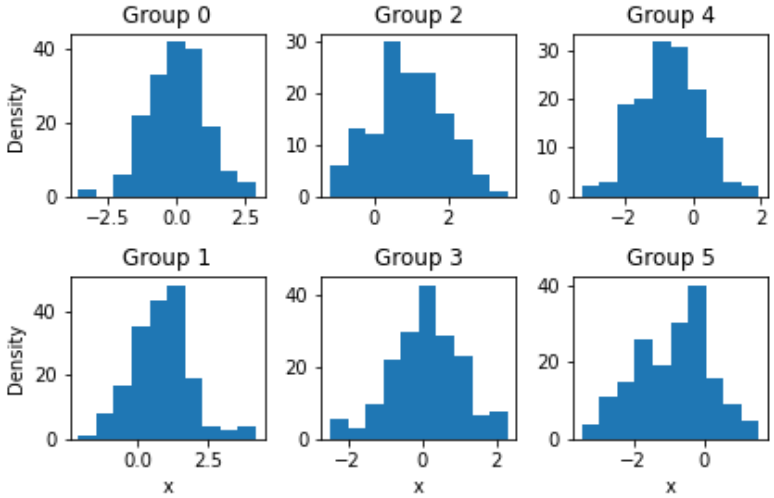
Write a function called `facet_hist()`. This function should accept five arguments:

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))
```

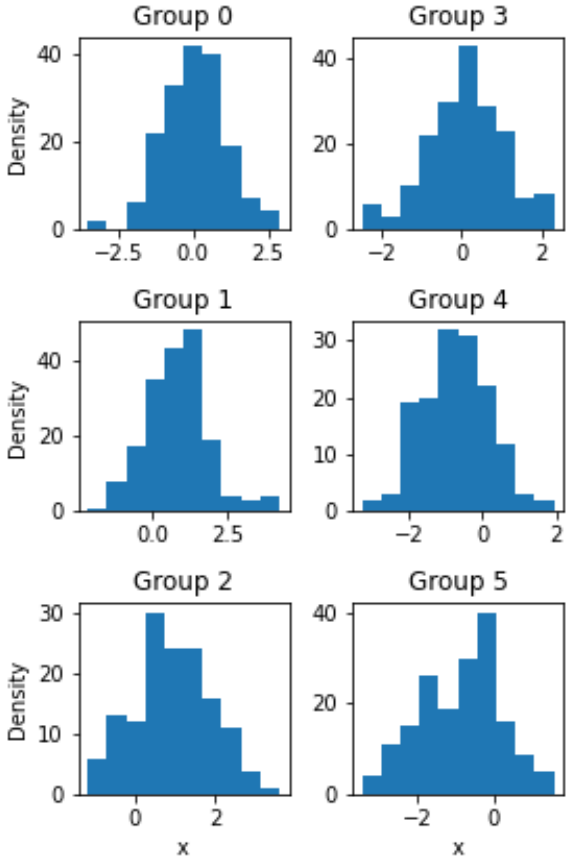
to create a plot like this:



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.

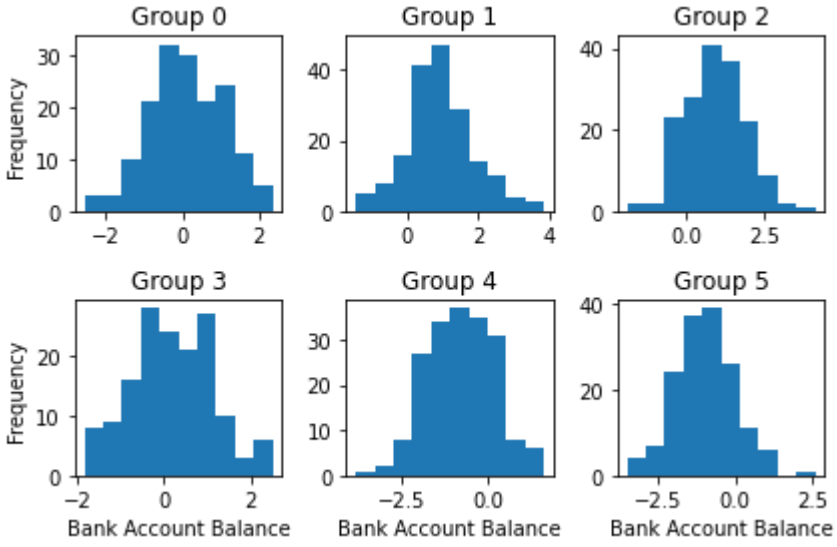
In [2]:

```
# your solution here
def facet_hist(groups, data, m_rows, m_cols, figsize, **kwargs):
    """
    Function that creates unique histograms and axes for each group of the supplied data
    Param groups: numpy integer array of number of groups in the data between 0 and n-1 data inclusive
    Param data: numpy array of data that consists of real numbers
    Param m_rows: integer representing number of desired histogram rows
    Param m_cols: integer representing the number of desired histogram columns
    Param figsize: tuple representing the size of the figure
    Returns: DNE. Function outputs faceted histogram to console
    """
    numAxes = m_rows*m_cols #calculates number of histograms

    #Checks if rows x columns is appropriate for the number of axes necessary for the data
    if ((numAxes) != len(np.unique(groups))): #if not, raises ValueError
        raise ValueError("The number of specified rows and columns does match the number of data groups!")

    #Construct figure with dimensions of subplots
    fig,ax = plt.subplots(m_rows, m_cols,figsize=figsize)

    #Add data to the subplots

    #Declaration of variables used in loop
    uniqueGroups = set(groups) #number of unique groups
    count = 0

    for i in uniqueGroups: #iterates through each group in the data

        #create boolean array that links group to data
        mask = (groups==i)

        #relevant data for unique group
        relevant = data[mask]
```

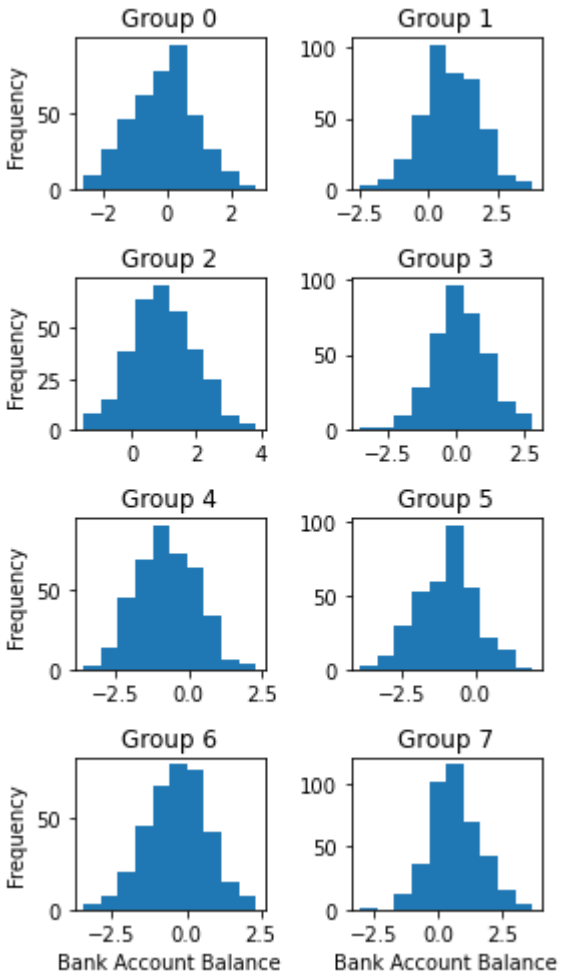
```
        #construct histogram by axes each representing unique data group
        #First, get the index of axis per data group
        rIndex = i//m_cols #row
        cIndex = i%m_cols  #column

        #Then, plot for that axis
        plt.sca(ax[rIndex,cIndex]) #sets axis of interest
        plt.hist(relevant, **kwargs) #plots relevant data with kwargs specifications
        plt.tight_layout() #improves visual formatting

        #Annotations to Plots
        axesTitle = "Group "+str(count)
        count+=1                                #incriment group number
        ax[rIndex,cIndex].set_title(axesTitle) #setting plot title
        if i >= (m_rows*m_cols)-m_cols:         #labels x axis of only last row
            ax[rIndex,cIndex].set_xlabel("Bank Account Balance")
        if cIndex == 0:                         #labels y axis of only first column
            ax[rIndex,cIndex].set_ylabel("Frequency")
```

In [4]:

```
# test code

groups, data = create_data(1000, 6)

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



In [5]:
```
# test code
groups, data = create_data(3000, 8)
facet_hist(groups, data, 4, 2, figsize = (4, 7))
```

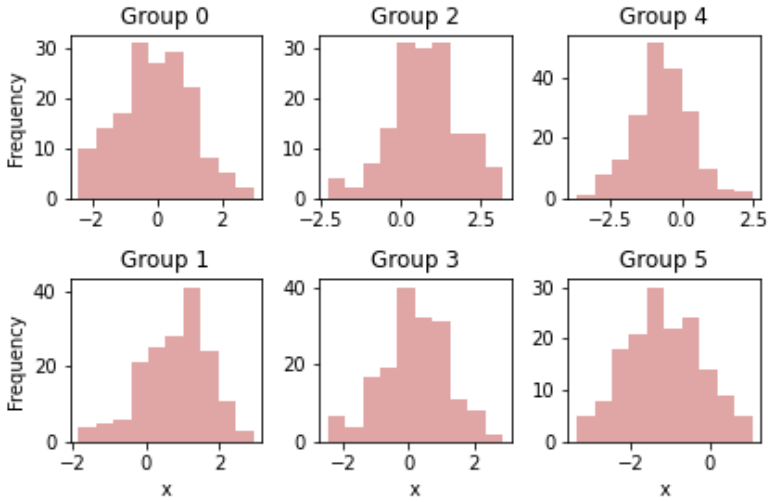


Part B

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")
```

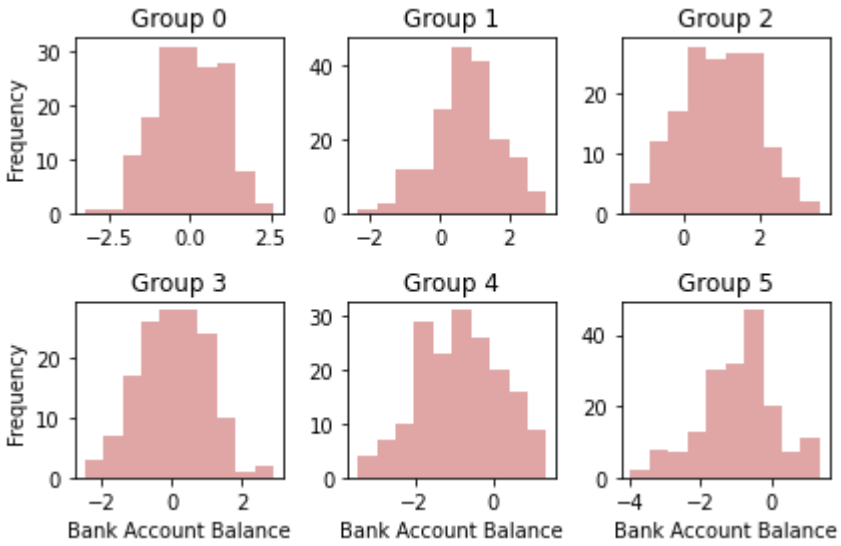
should produce



Example output.

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

In [6]:
```
# 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 [7]:
```
# 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[7]:

	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

	Overall rank	Country or region	Score	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity	Perceptions of corruption
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
```

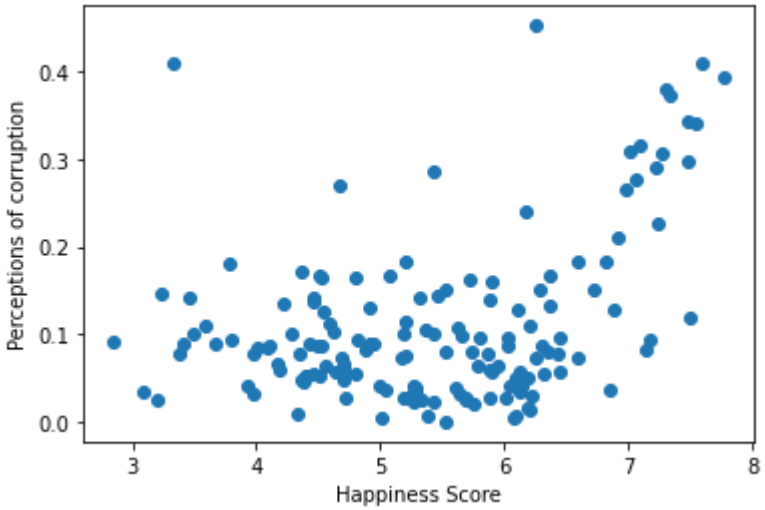
Technically, this output is a `pandas` `Series`; however, in this context (and most others) it's fine to simply think of it as a 1-dimensional `np.array()`.

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 [51]:
```
# plotting code here
fig,ax = plt.subplots(1)                                          #creation of plot
ax.scatter(happiness["Score"],happiness["Perceptions of corruption"]) #data into scatter
ax.set(xlabel="Happiness Score",ylabel="Perceptions of corruption")   #labels
```

Out[51]:
```
[Text(0.5, 0, 'Happiness Score'), Text(0, 0.5, 'Perceptions of corruption')]
```



Discussion:

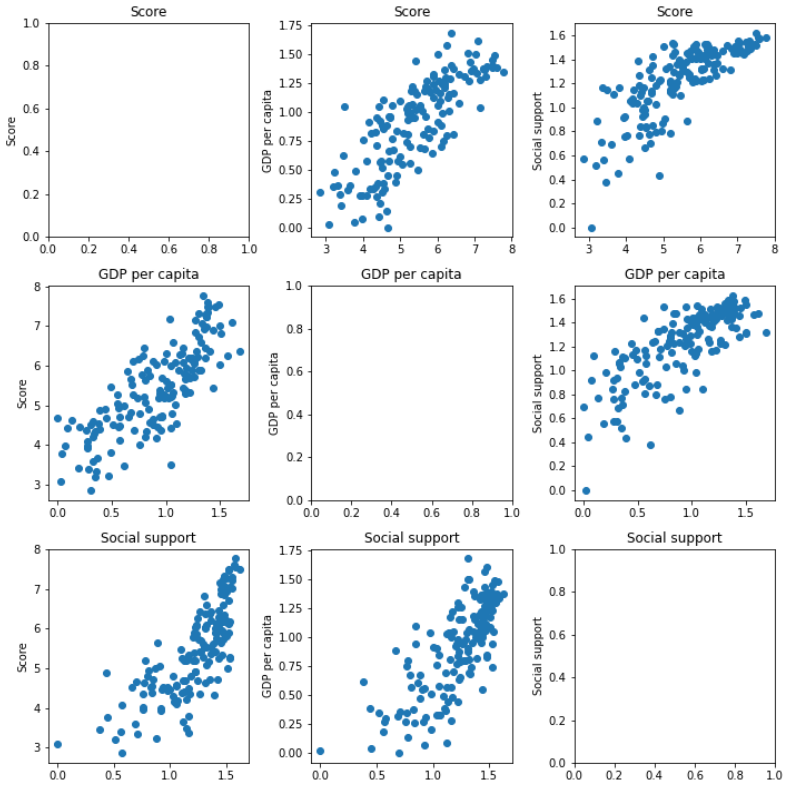
There appears to be slight if no correlation between a the happiness people report in a country and their perceptions of the level of corruption in the country. The majority of nationals from happiness levels 3 to 7 report similar levels of perceived corruption (between 0 and .2 percent). However, there are a few notable corruption outliers in the data set, particularly the countries whose citizens report an average happiness of 7 or higher. Curiously, as happiness increases past 6.5, it strongly correlates with an increased perception of corruption.

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!

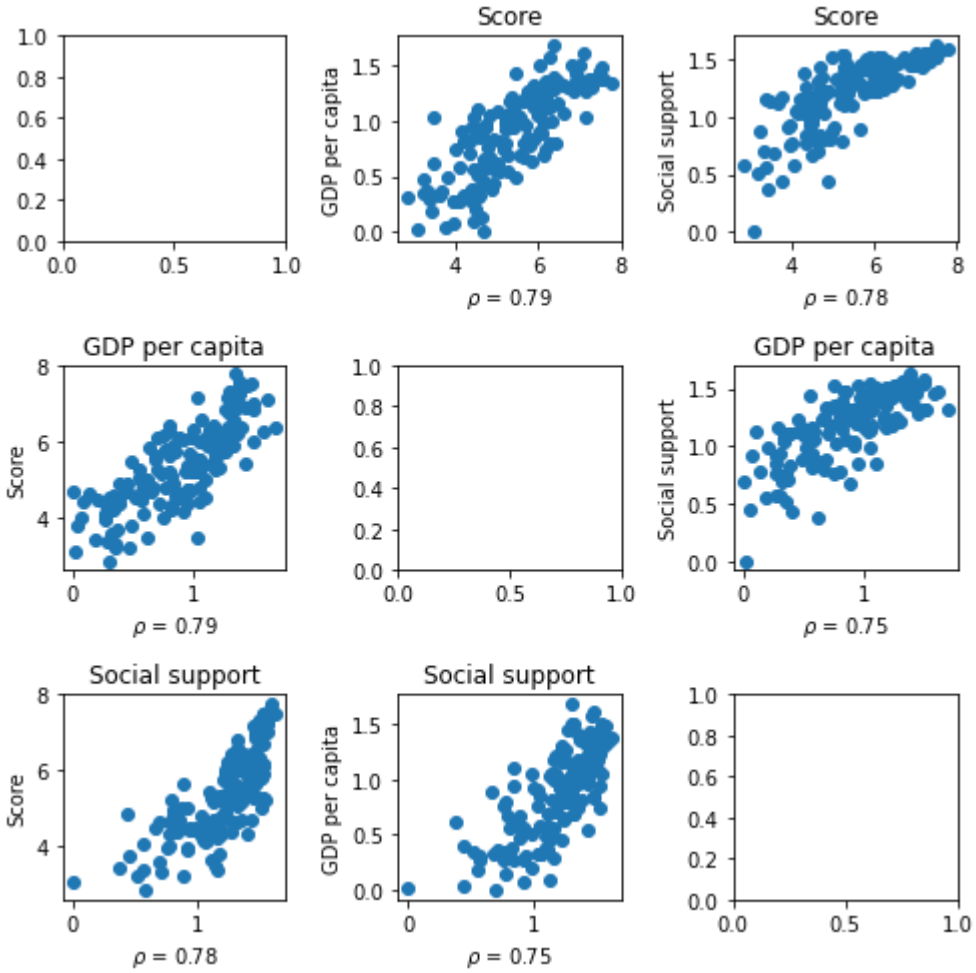
In [12]:

```
# define your function
def scatterplot_matrix(cols, figsize):
    """
    Function that produces a matrix of scatterplots representing columns of the numerical pandas
    happiness data against other columns within the set
    Param cols: list of strings representing the data columns to be included in the scatterplot matrix
    Param figsize: tuple representing the dimension specification of the figure
    Returns: DNE. Function prints matrix of scatterplots to console
    """

    #Construct plot with dimensions cols x cols meaning cols^2 axes
```

```
    pltDim = len(cols)
    fig,ax = plt.subplots(pltDim, pltDim,figsize=figsize)

    #Populate individual axis with data
    for i in range(pltDim**2): #for loop through each axis
        rIndex = i//pltDim #row
        cIndex = i%pltDim  #column

        #Retrieve data label from cols depending on row/column
        x = cols[rIndex]
        y = cols[cIndex]

        #Plotting data on axis
        if rIndex == cIndex: #prevents same column of data from being plotted against self
            continue
        ax[rIndex,cIndex].scatter(happiness[x],happiness[y]) #plotting on axis
        plt.tight_layout()

        #Calculate correlation coefficient
        xyCorr = np.corrcoef(happiness[x],happiness[y])[0,1] #returns 2D numpy array, so select entry via indexing

        #Labels and correlation coefficient added to fig
        ax[rIndex,cIndex].set(title = x, ylabel = y, xlabel = (r"$\rho$ = " + str(np.round(xyCorr, 2))))
```

In [13]:

```
# 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))
```



Discussion of Plotted Data BEFORE part C:

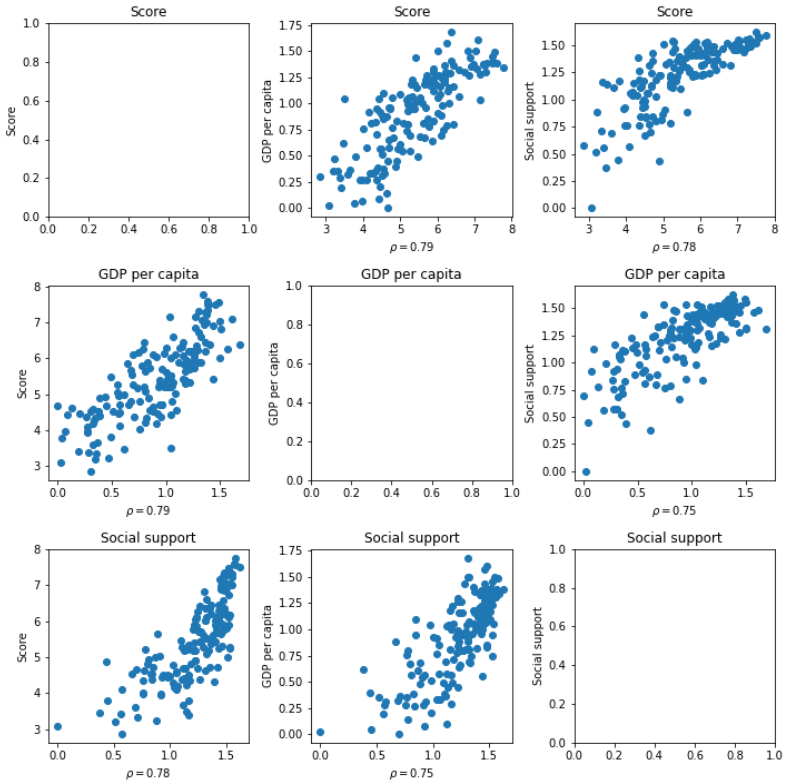
Reported happiness score, GDP per capita, social support, and healthy life expectancy all seem to be positively correlated with each other as an increase in one axis follows an increase in another axis.

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)))
```

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?

Discussion:

Positive correlations exist between all the inputted columns of data (happiness score, GDP per capita, social support, and healthy life expectancy). I'm surprised to see that the correlation coefficients between happiness score and the other three columns of data deviate the least (between .78 and .79) out of all comparisons; whereas, the healthy life expectancy correlation coefficient analyzed in relation to the other three columns deviated the most (between .72 and .84). The strongest correlation appeared between healthy life expectancy and GDP per capita at .84 which makes sense because generally the wealthier the country, the better health care and standard of living available. On the other hand, I found it interesting that healthy life expectancy correlated the least with social support because I would have imagined strong social support implies better community care and in turn wealthier countries (although the correlation between social support and GDP per capita was only .75).

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 [8]:
```
# 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&period2=1712372632&interval=1d&ever
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 [10]:
```
# your solution here

#Construct 1D numpy array per_day using diff of cases between days
#Without prepend, length of per_day constructed from np.diff is 837 representing change in days from day 1
#Prepend ([0]) to beginning so that diff starts from 0 to 1 cases and length of per_day is 838

per_day = np.diff(COVID['cases'], prepend = np.zeros(1,dtype=int)) #length 838
per_day_date = COVID['date'] #length 838
```

Part B

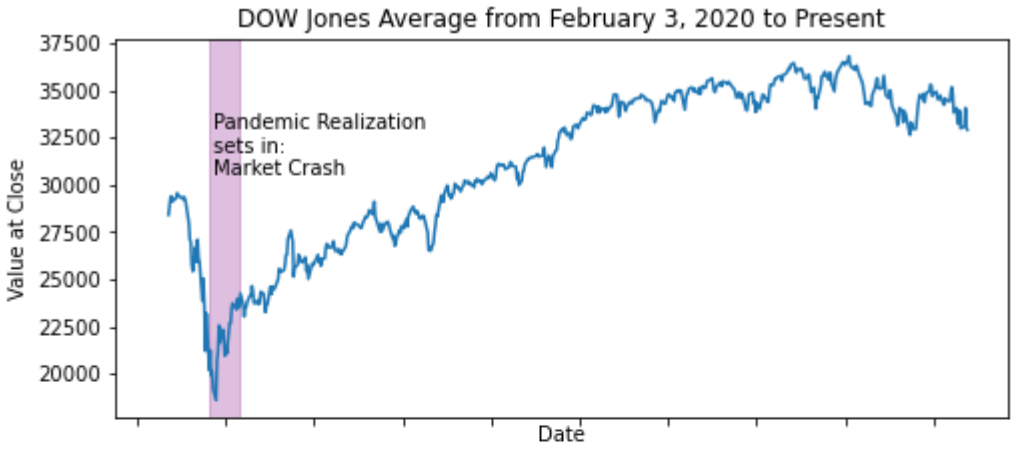
Create a figure with two very wide axes, one on top of the other (i.e. two rows, one column). Use the `sharex` argument of `plt.subplots()` to ensure that these two plots will share the same horizontal axis.

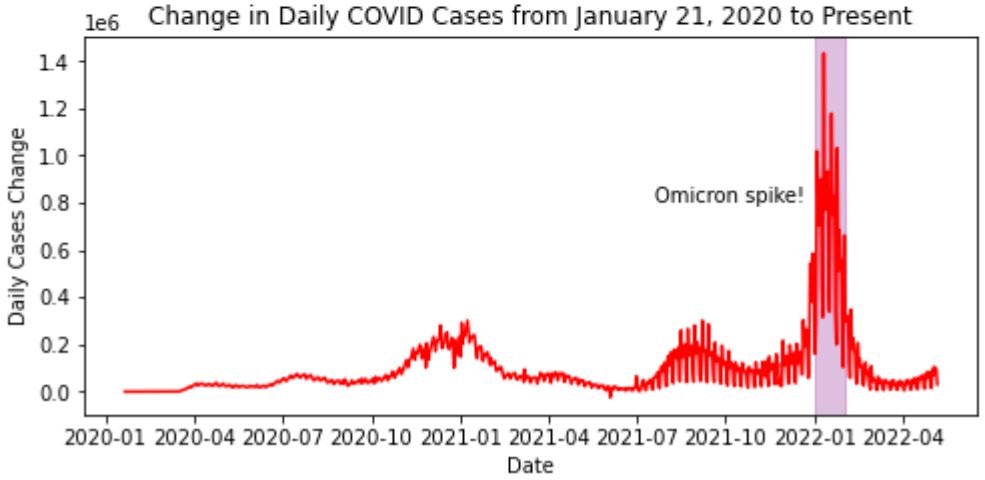
Then:

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 [89]:
```
# your solution here
# modify this block in the remaining parts of the problem

#construct figure with two axes (sharing x axis)
fig,ax = plt.subplots(2,1, sharex=True, figsize = (8,8))

#plot data: top is dow jones, bottom is covid case changes
ax[0].plot(DJI['date'],DJI['Close'])
ax[1].plot(per_day_date, per_day, color="red");

#part C: rectangular shade
ax[0].axvspan(datetime.datetime(2020,3,15), #March 15, 2020 - April 15, 2020
              datetime.datetime(2020,4,15),
              alpha = .25,
              color = "purple")
ax[1].axvspan(datetime.datetime(2022,1,1), #January 1, 2022 - January 31, 2022
              datetime.datetime(2022,1,31),
              alpha = .25,
              color = "purple")

#part D: text annotations
ax[0].text(datetime.datetime(2020,3,20),
           30500,
           "Pandemic Realization\nsets in:\nMarket Crash")
ax[1].text(datetime.datetime(2021,7,20),
           800000,
           "Omicron spike!");

#part E: title, labels
ax[0].set(xlabel = "Date", ylabel = "Value at Close",
          title = "DOW Jones Average from February 3, 2020 to Present")
ax[1].set(xlabel = "Date", ylabel = "Daily Cases Change",
          title = "Change in Daily COVID Cases from January 21, 2020 to Present")
plt.subplots_adjust(hspace=.3) #improve spacing for readability
```





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.