Homework 4

Marlene Lin

105329845

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

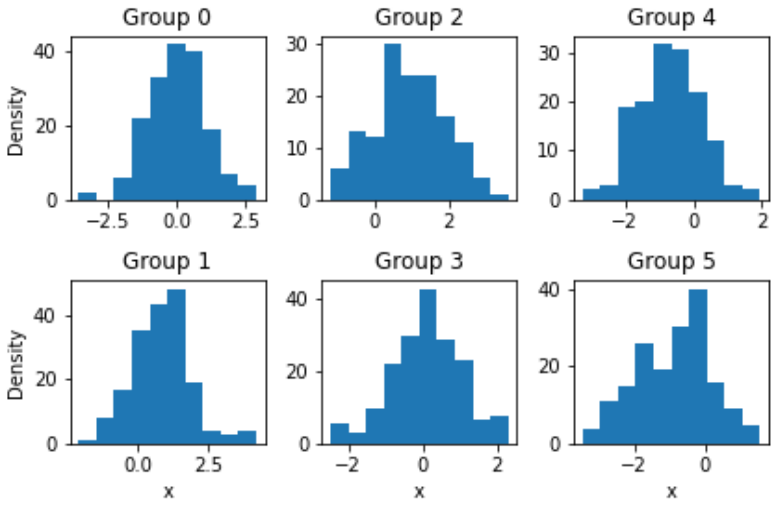
Write a function called `facet_hist()`. This function should accept five arguments:

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))
```

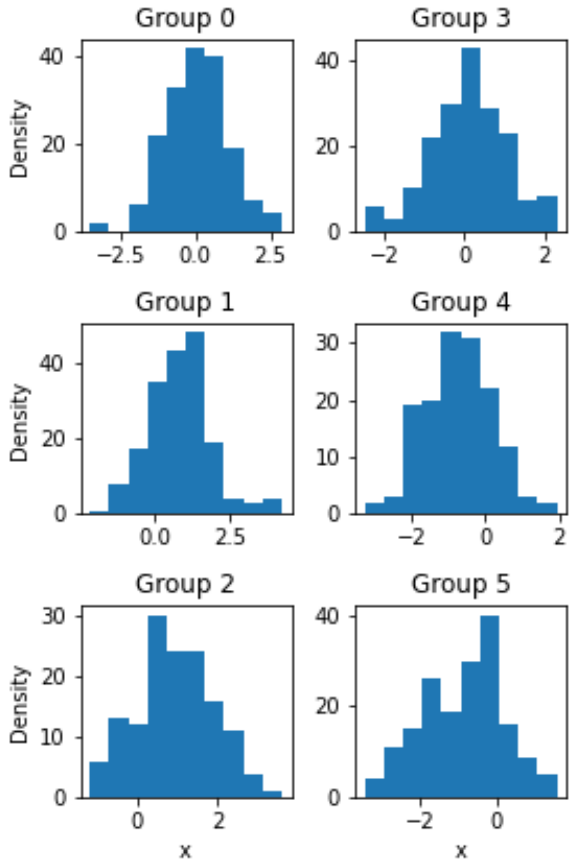
to create a plot like this:



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]:

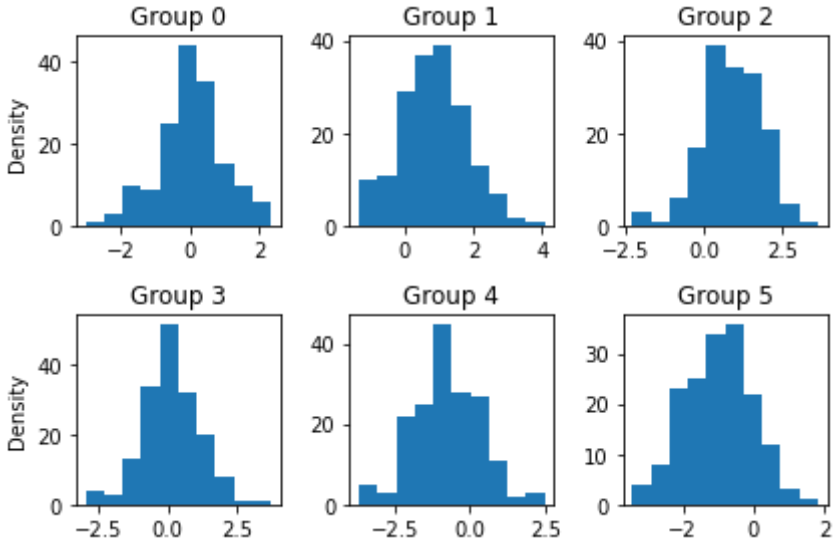
```
# your solution here
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
def facet_hist(groups,data,m_rows,m_cols, figsize,**kwargs):
    '''
    Generate a set of histograms for the
    data in each group of groups,
    arranging into m_rows rows and m_cols columns
    with optional plot parameters **kwargs
    '''

    #unique values in groups:
    labels = set(groups)
    #value error check
    if not m_rows * m_cols == len(labels):
        raise ValueError("#histograms incompatible with  #total groups")
    #create subplots
    fig, axarr = plt.subplots(m_rows,m_cols,figsize=figsize)
    for r in range(m_rows):
        for c in range(m_cols):
            idx = r * m_cols + c
            # pass in plot parameters for histograms
            axarr[r,c].hist(data[groups == idx],**kwargs) # if np, empty hist
            axarr[r,c].set_title("Group "+str(idx))
            #set left y labels
            if c == 0:
```

```
                axarr[r,c].set(ylabel = "Density")
    plt.tight_layout()
```

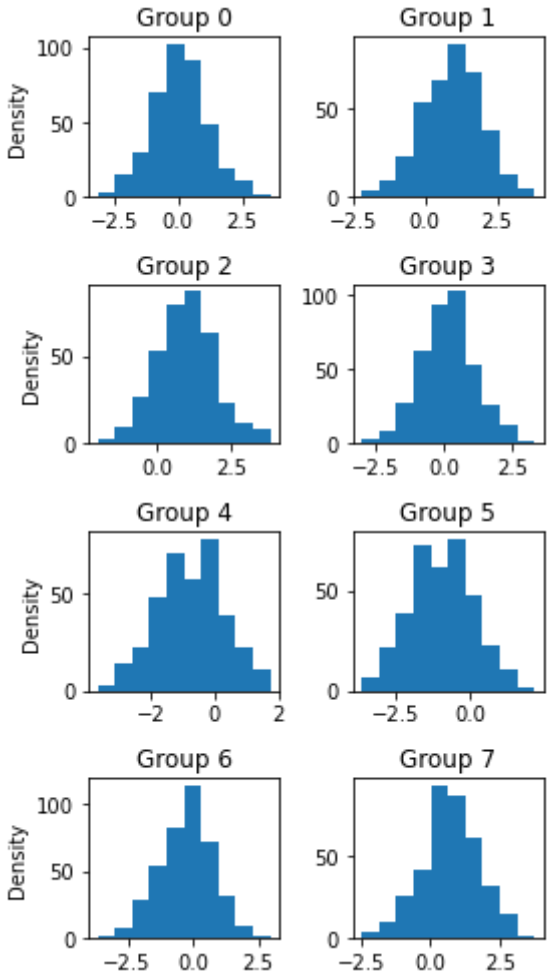
In [3]:

```
# test code
groups, data = create_data(1000, 6)
facet_hist(groups, data, 2, 3, figsize = (6, 4))
```



In [4]:

```
# test code
groups, data = create_data(3000, 8)
facet_hist(groups, data, 4, 2, figsize = (4, 7))
```

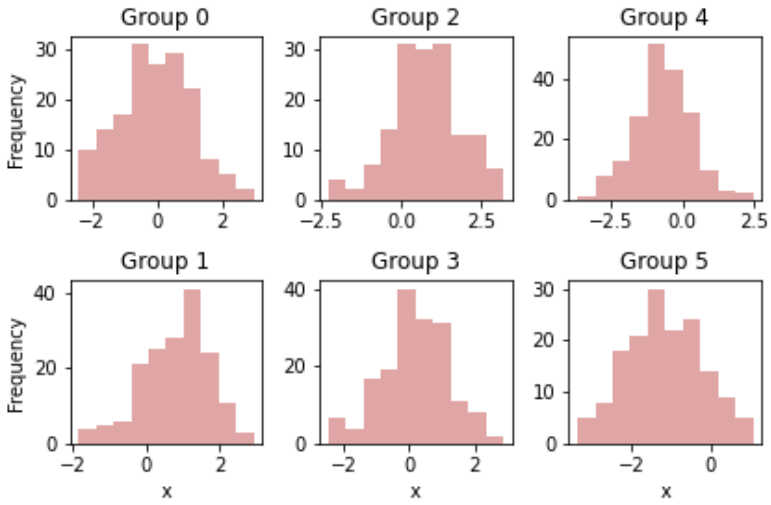


Part B

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")
```

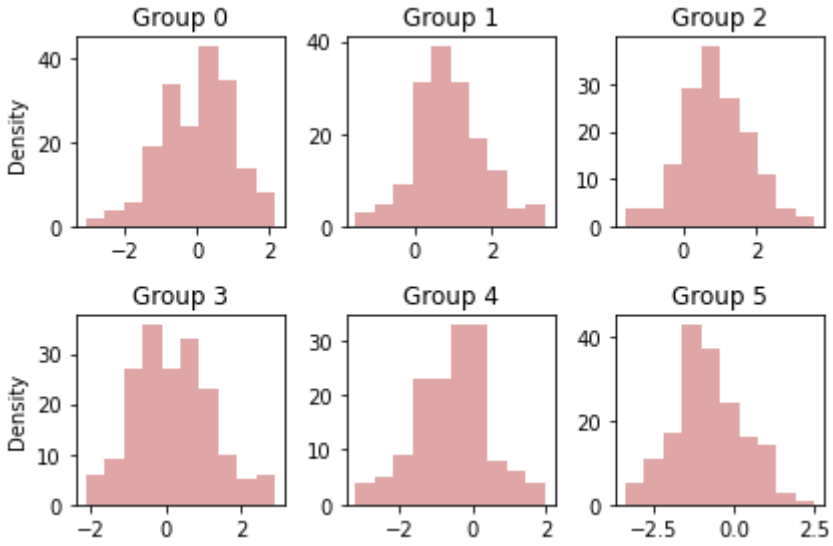
should produce



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
#happiness = pd.read_csv(url)
happiness = pd.read_csv("2019.csv")
```

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
```

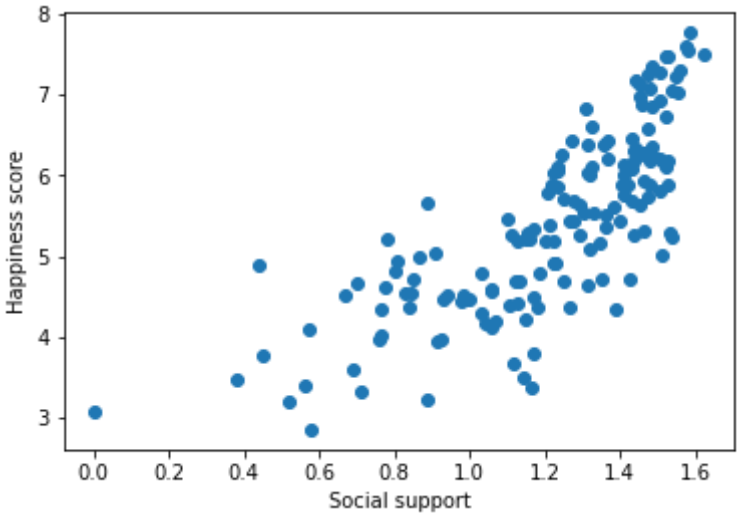
Technically, this output is a `pandas` `Series`; however, in this context (and most others) it's fine to simply think of it as a 1-dimensional `np.array()`.

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]:
```
# plotting code here
plt.scatter(happiness["Social support"],happiness["Score"])
plt.xlabel("Social support")
plt.ylabel("Happiness score")
```

Out[7]:
```
Text(0, 0.5, 'Happiness score')
```



There seems to be a **positive correlation** between happiness score and social support. It's fair because with better social support, people in general can live with less worries about healthcare, education, and other public services. People are also less likely to be in situations where they need to worry about fulfilling their basic human needs, like access to food, living quarters, and clean water

---

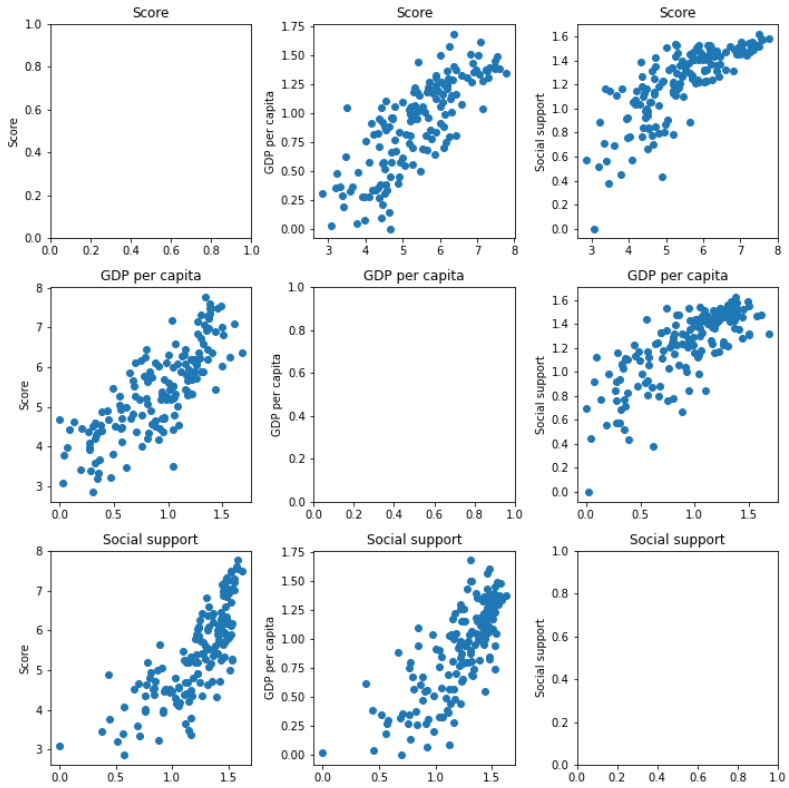
Part B

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))
```

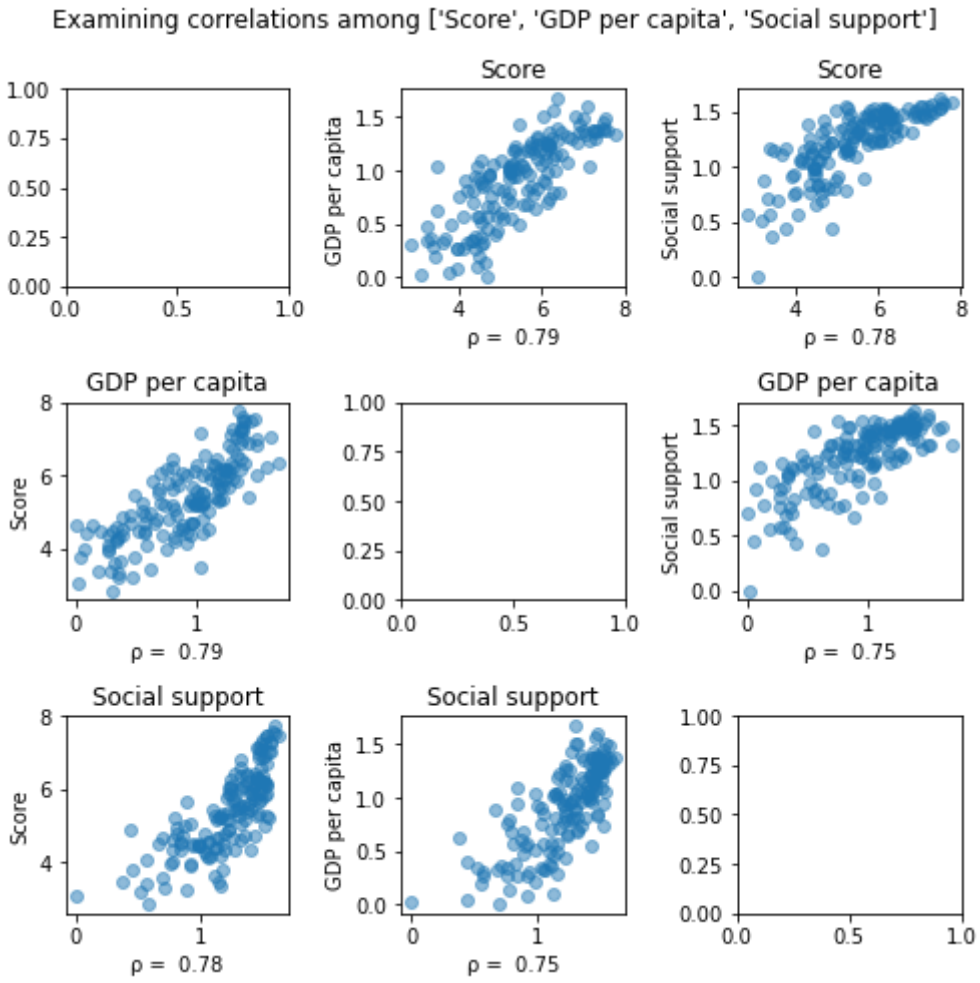


There is a separate scatterplot for each possible pair of variables. In fact, there are two: one where the first variable is on the horizontal axis, and one where it's on the vertical axis. Some analysts prefer to remove half the plots to avoid redundancy, but you don't have to bother with that. The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

Don't forget comments and docstrings!

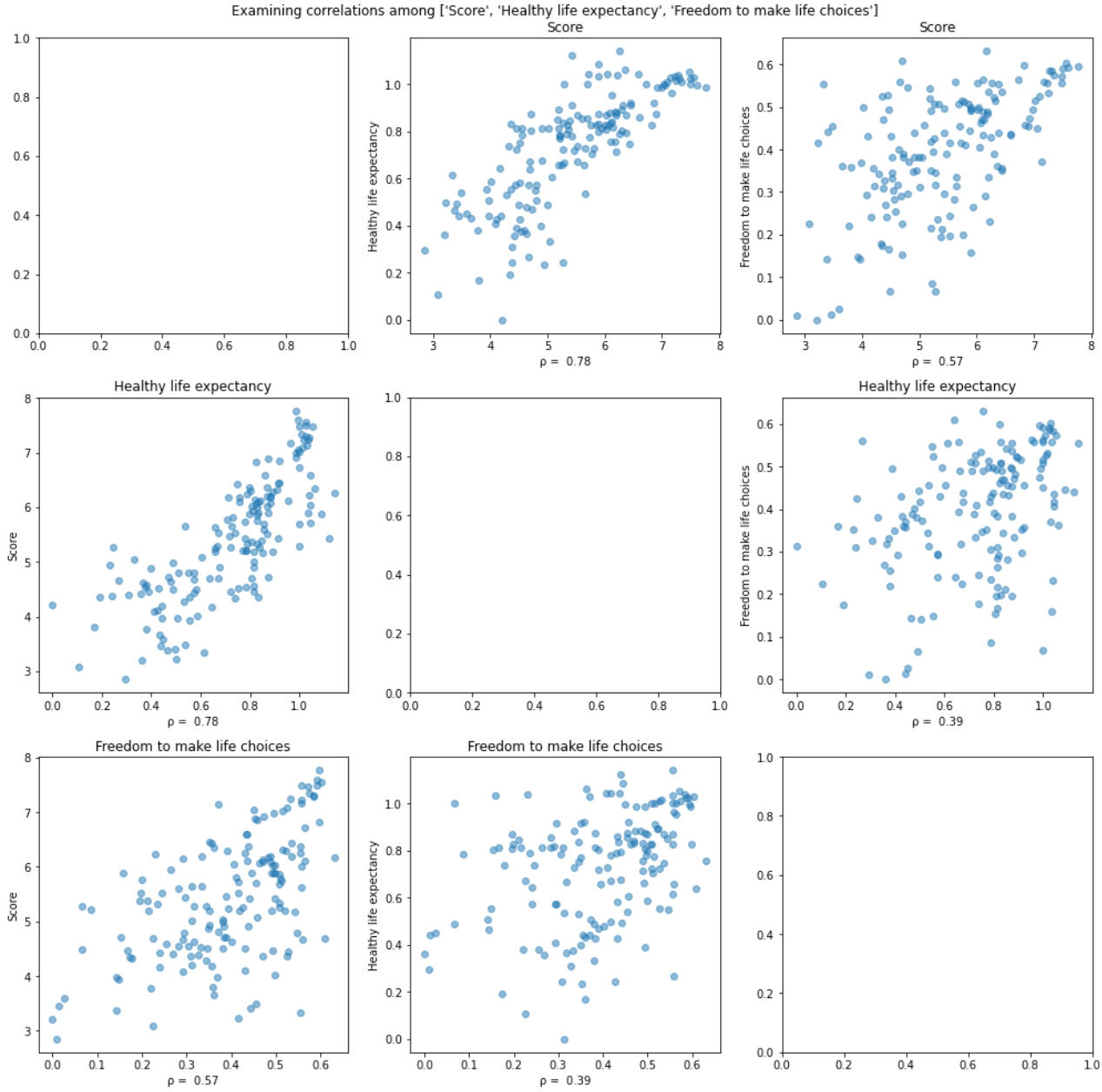
In [8]:
```
# define your function
def scatterplot_matrix(cols, figsize):
    '''
    Take in a list of variables (cols)
    and generate a matrix of scatterplots
    depicting the correlations between
    all possible pairs of different variables.
    '''
    tot = len(cols)
    fig,ax = plt.subplots(tot,tot,figsize=figsize)
    for r in range(tot):
        for c in range(tot):
            #generate sub scatterplots between diff vars
            if r is not c:
                cor = np.corrcoef(happiness[cols[r]],happiness[cols[c]])[0,1]
                ax[r,c].scatter(happiness[cols[r]],happiness[cols[c]],alpha=0.5)
                ax[r,c].set(xlabel = f'ϱ = {cor: .2f}',ylabel=cols[c])
                ax[r,c].set_title(cols[r])
    fig.suptitle("Examining correlations among "+str(cols))
    plt.tight_layout()
```

In [9]:
```
# test your code, several times if needed, and discuss the correlations you obse
# Add code cells if needed to show multiple outputs.
cols = ["Score",
        "GDP per capita",
        "Social support"]
scatterplot_matrix(cols,figsize = (7,7))
```



Discussion: There seem to be positive correlations between social support and GDP per capita, GDP per capita and happiness, as well as social support and GDP per capita. In short, better social support is affordable by the society due to high GDP per capita, better social support guarantees happiness, and happiness could lead to higher productivity which then results in higher GDP per capita. There are sort of interconnected standards of measuring human development?

In [10]:
```
cols = ["Score",
        "Healthy life expectancy",
        "Freedom to make life choices"]
scatterplot_matrix(cols,figsize = (14,14))
```



Discussion: the positive correlations between freedom to make life choices and healthy life expectancy is relatively weak, as well as that between freedom to make life choices and happiness scores. (Other indicators usually have ~0.7-0.8 coeff of cors with the happiness scores). people aren't necessarily utility maximization machines that make the best life decisions leading to happiness, or happiness is sometimes better 'guided/planned' by a greater institution than individuals or small communities ... the choice space could be limited so there really isn't much one's wishes could attain, freewil is a joke, yada yada

---

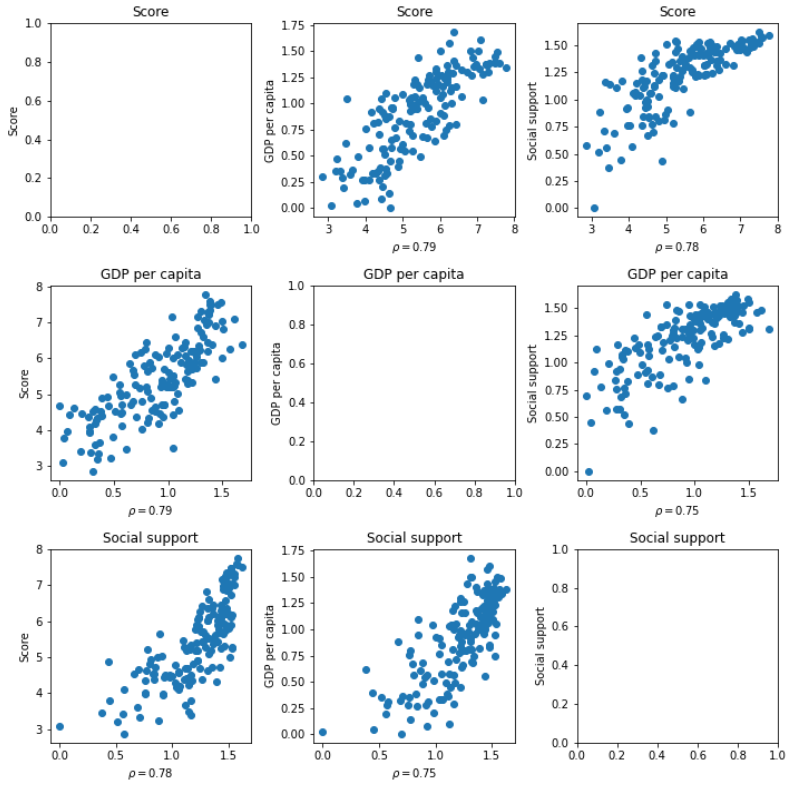
Part C

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?

When performing variable selection for prediction (in case of a ordinary linear regression model, we can examine:

1. correlation coefficients, degree of associations between two variables

2. variable inflation factor (VIF), taking into account multicollinearity when selecting for more than one variables.

Ideally, we want VIF to be within accetable ranges with the final variables we select while having high correlation coefficients.

The variables we are selecting from include GDP per capita Social support Healthy life expectancy Freedom to make life choices Generosity and Perceptions of corruption. Getting rid of country or region and overall rank which aren't numeric

We are predicting the happiness score.

Examining corr coef and VIF:

In [11]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

corcoef = np.round(np.corrcoef(happiness.iloc[:,2:],rowvar = False), 2)
d = pd.DataFrame(corcoef[0,1:]).T
d.columns = happiness.columns[3:]
predictors = happiness.iloc[:,3:]
vif = pd.DataFrame(variance_inflation_factor(predictors.values, i)
                           for i in range(len(predictors.columns))).T
vif.columns = happiness.columns[3:]
results = pd.concat([vif,d])
results.index=['vif','cor coef']
```

In [12]:
```
results
```

Out[12]:

	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity	Perceptions of corruption
vif	23.394281	31.70172	34.310687	13.359599	5.126148	3.39616
cor coef	0.790000	0.78000	0.780000	0.570000	0.080000	0.39000

The first three vars have pretty high collinearity, while the last three have relatively low cor coeff. So picking from the first three should be enough. Repeat the process:

In [13]:
```
corcoef = np.round(np.corrcoef(happiness.iloc[:,2:6],rowvar = False), 2)
d = pd.DataFrame(corcoef[0,1:]).T
d.columns = happiness.columns[3:6]
predictors = happiness.iloc[:,3:6]
vif = pd.DataFrame(variance_inflation_factor(predictors.values, i)
                           for i in range(len(predictors.columns))).T
vif.columns = happiness.columns[3:6]
results = pd.concat([vif,d])
results.index=['vif','cor coef']
results
```

Out[13]:

	GDP per capita	Social support	Healthy life expectancy

	GDP per capita	Social support	Healthy life expectancy
vif	21.160318	21.775593	32.951792
cor coef	0.790000	0.780000	0.780000

let's get rid of life expetance

In [14]:
```
corcoef = np.round(np.corrcoef(happiness.iloc[:,2:5],rowvar = False), 2)
d = pd.DataFrame(corcoef[0,1:]).T
d.columns = happiness.columns[3:5]
predictors = happiness.iloc[:,3:5]
vif = pd.DataFrame(variance_inflation_factor(predictors.values, i)
                           for i in range(len(predictors.columns))).T
vif.columns = happiness.columns[3:5]
results = pd.concat([vif,d])
results.index=['vif','cor coef']
results
```

Out[14]:

	GDP per capita	Social support
vif	13.321345	13.321345
cor coef	0.790000	0.780000

which still has a pretty high VIF, because per discussion in part b these two vars are correlated. **but picking GDP per capita (cor coeff) or keeping both social support and GDP per capita should work as good enough predictors for hte happiness score**

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 [15]:
```
# 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=15807
DJI = pd.read_csv('DJI.csv')
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('covid.csv')
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 [16]:
```
# your solution here
#number of new cases per day
per_day = np.insert(np.diff(COVID['cases']),0,COVID['cases'][0])
#corresponding date
per_day_date = COVID['date']
per_day.shape == per_day_date.shape
```

Out[16]: True

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.

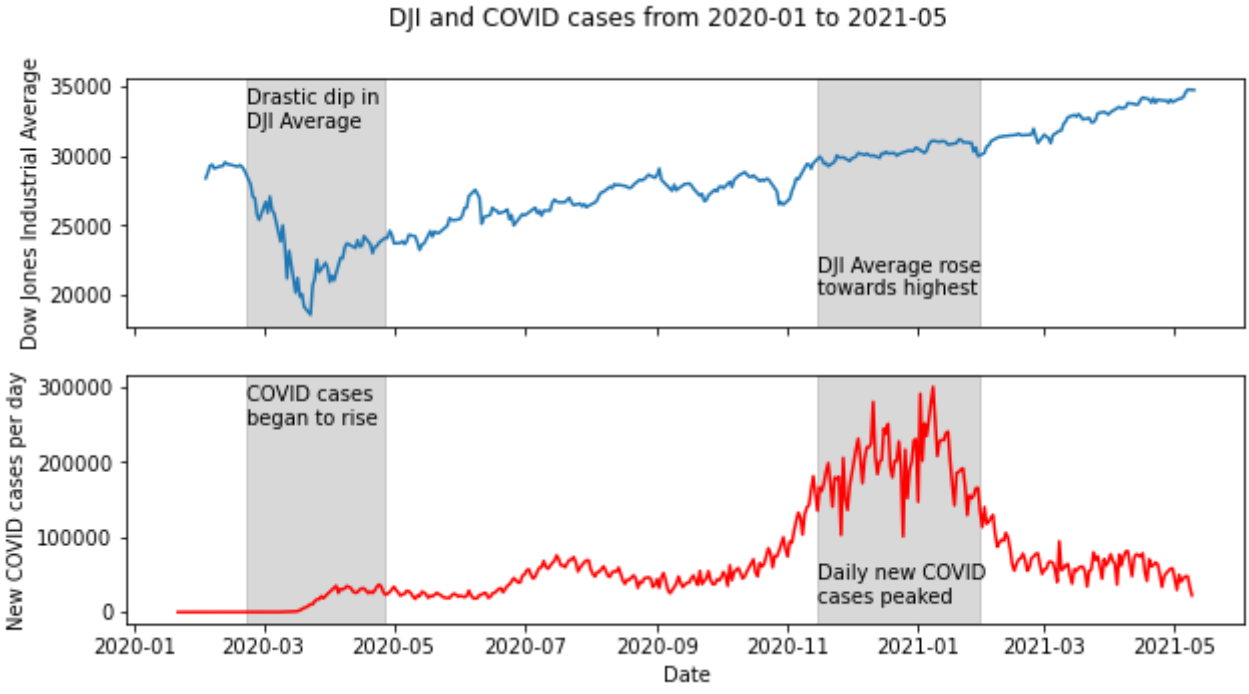
Give your plot horizontal and vertical axis labels.

In [17]:
```
# your solution here
# modify this block in the remaining parts of the problem
fig,ax = plt.subplots(2,1,figsize=(10,5),sharex=True)
ax[0].plot(DJI['date'],DJI['Close'])
ax[0].set(ylabel="Dow Jones Industrial Average")
ax[1].plot(per_day_date,per_day, color = "red")
ax[1].set(ylabel = "New COVID cases per day")
ax[1].set_xlabel("Date")

#first section
ax[0].axvspan(datetime.datetime(2020,2,22),
              datetime.datetime(2020,4,27),
```

```
                alpha = .3,
                color = "gray")
ax[0].text(datetime.datetime(2020,2,22),
           32000,
           "Drastic dip in \nDJI Average")
ax[1].axvspan(datetime.datetime(2020,2,22),
              datetime.datetime(2020,4,27),
              alpha = .3,
              color = "gray")
ax[1].text(datetime.datetime(2020,2,22),
           250000,
        "COVID cases \nbegan to rise")
#second section
ax[0].text(datetime.datetime(2020,11,15),
           20000,
           "DJI Average rose \ntowards highest")
ax[0].axvspan(datetime.datetime(2020,11,15),
              datetime.datetime(2021,1,30),
              alpha = .3,
              color = "gray")
ax[1].text(datetime.datetime(2020,11,15),
           12000,
           "Daily new COVID\ncases peaked")
ax[1].axvspan(datetime.datetime(2020,11,15),
              datetime.datetime(2021,1,30),
              alpha = .3,
              color = "gray")
fig.suptitle("DJI and COVID cases from 2020-01 to 2021-05")
```

Out[17]: Text(0.5, 0.98, 'DJI and COVID cases from 2020-01 to 2021-05')



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.