Homework 6

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.

I received help from Brian Burrous who gave me suggestions on Problem 1 regarding how to use the loc accessor with reset_index().

Austin Wuthrich 5/18/22

Comments/docstrings are required for problems 1 and 3. They are not required for problems 2.

Problem 0

It is highly recommended that you work with your group to fully complete the recent Discussion assignments related to the Palmer Penguins data set, as these will directly help with your project.

Problem 1

Gapminder is a foundation, based on Sweden, that aims to enhance basic awareness of basic facts about the socioeconomic global world. As part of their efforts, they collect detailed statistics on life expectancy, population, and GDP, sometimes going back over many years.

Here, we'll work with an excerpt of the Gapminder data. This excerpt has been packaged up and made available via Jenny Bryan's `gapminder` repo on Github.

Run the code below to retrieve the data and take a look. As usual, you can also directly download the data by pasting the url into your browswer, saving the file, and reading it in locally via `pandas.read_csv` .

In [74]:
```
import pandas as pd

url = "https://philchodrow.github.io/PIC16A/datasets/gapminder.csv"
```

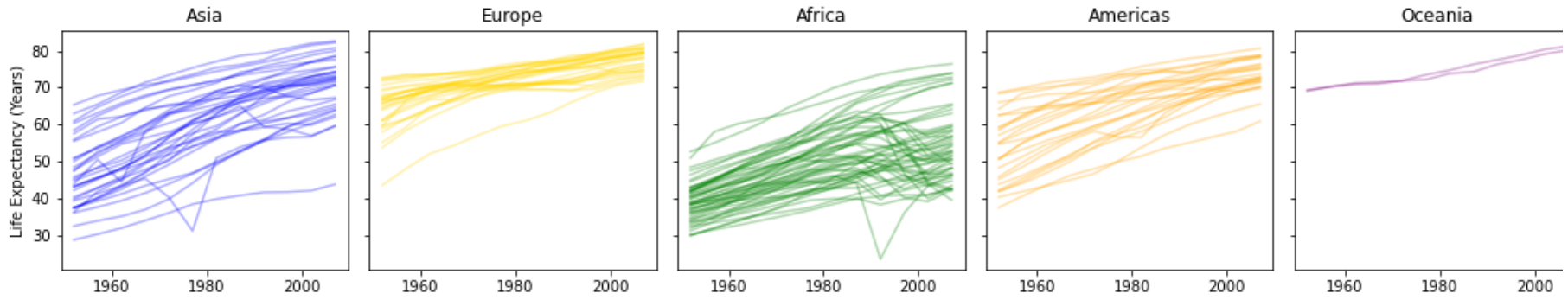
```
gapminder = pd.read_csv(url)
gapminder
```

Out[74]:

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
...	...	...	...	...	...	...
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298

1704 rows × 6 columns

Use the `gapminder` data to create the folowing visualization. Here, each trendline corresponds to a distinct country, which in turn lies on the stated continent.



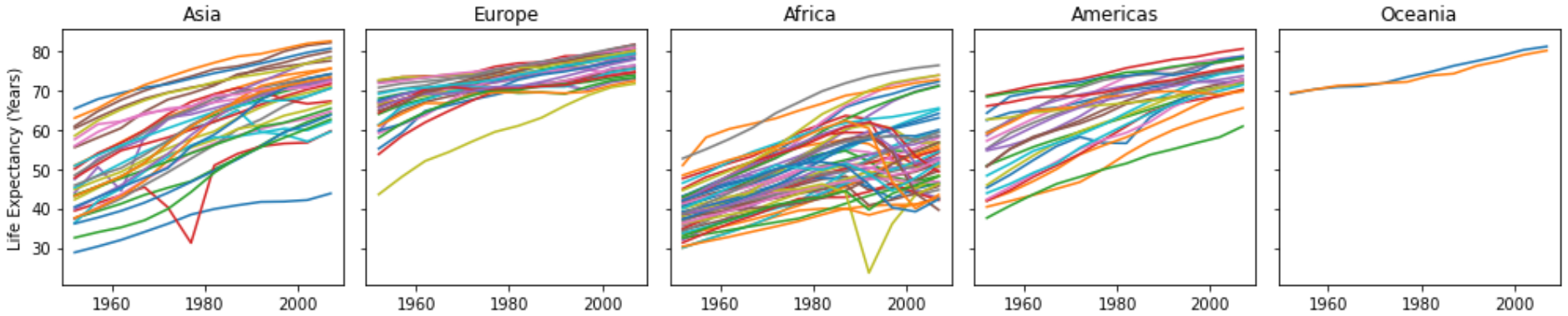
You should achieve this result **without for-loops**, and also without manually creating the plot on each axis. You may find it useful to define additional data structures, such as dictionaries, that assign colors or axis indices to each continent. Feel free to modify aesthetic details of the plots, such as the colors.

Hint: Try `df.groupby(["country"]).apply(f)`, where `f` is some very smart function that you have figured out.

In [75]:

```
# your solution here
import numpy as np
from matplotlib import pyplot as plt

#make copies of dataframe
df = gapminder.copy()
copyDf = df.copy()

#edit data so that continents correspond to axis index
convertContinent = {"Asia": 0,
                    "Europe": 1,
                    "Africa": 2,
                    "Americas": 3,
                    "Oceania": 4}
df["continent"] = df['continent'].map(convertContinent) #map dictionary to data frame and overwrite strings as ints

#create figure
fig,ax=plt.subplots(1,5,figsize=(14,3),sharex=True,sharey=True)

#Function that can be applied to the grouped data
def graph(df,col_x,col_y,copyDf):
    """
    Function that inputs a grouped dataframe by country and creates an axis for each continent
        with date in years on the x-axis and life expectancy in years on the y axis.
    Param df: data frame grouped by country
    Param col_x: string representing the column of data for the x axis
    Param col_y: string represneting the column of data for the y axis
    Param copyDf: copy of the original data frame before being edited by the mapping dictionary
    Returns: DNE. Prints out plots to console
    """
    import numpy as np
    from matplotlib import pyplot as plt

    #For axis labels later. Getting unique names for the continents from the data frames
```

```
    intermediate = copyDf["continent"].unique()

    #get data from df for x, y, and continent number
    x=df[col_x]
    y=df[col_y]

    #groupby gives a dataframe for each country. Below resets each data frame to have same indices
    dfPerCountry = df.reset_index()
    countryCode = dfPerCountry.loc[0,"continent"] #accessing value of first row in each data frame under continent

    #plot data
    ax[countryCode].plot(df[col_x],df[col_y])

    #annotations + edits
    plt.xticks([1960,1980,2000])
    ax[0].set(ylabel="Life Expectancy (Years)")
    #accessing unique elements of unique list of continent names created earlier
    ax[countryCode].set_title(label=intermediate[countryCode])

#applying the function on the data
test=df.groupby(["country"]).apply(graph,'year','lifeExp',copyDf)

#edits
plt.tight_layout()
```



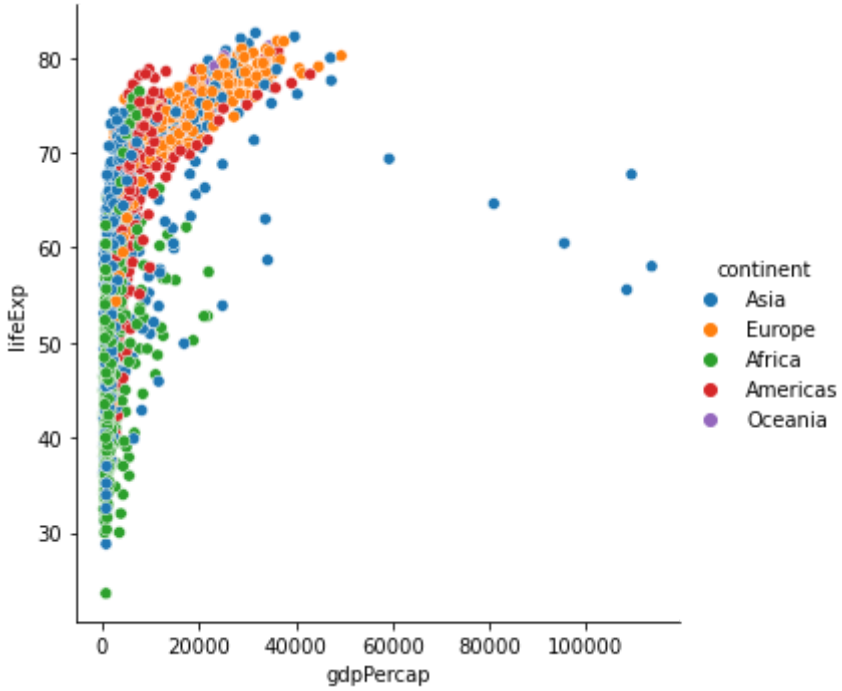
Problem 2

In this problem, you will create several attractive visualizations of the `gapminder` data set using the Seaborn package. Seaborn is a high-level interface to Matplotlib specially designed for working with Pandas data frames. Seaborn makes it relatively easy to create complex, multifaceted data graphics. The drawback is that it can sometimes be more difficult to exercise control over figure details, such as axis labels or figure size.

Part of the learning objective for this problem is for you to **practice learning from documentation and examples**. I have given you some pointers in each problem, but it will be up to you to fill in the blanks.

Import the `seaborn` package with the line `import seaborn as sns`. Then, run the code block below to create a scatterplot of `gdpPercap` (gross domestic product per capita) against `lifeExp` (life expectancy), with different colors for each continent.

In [77]:
```
from matplotlib import pyplot as plt
import seaborn as sns

fgrid = sns.relplot(x = "gdpPercap",
                    y = "lifeExp",
                    hue = "continent",
                    data = gapminder)
```



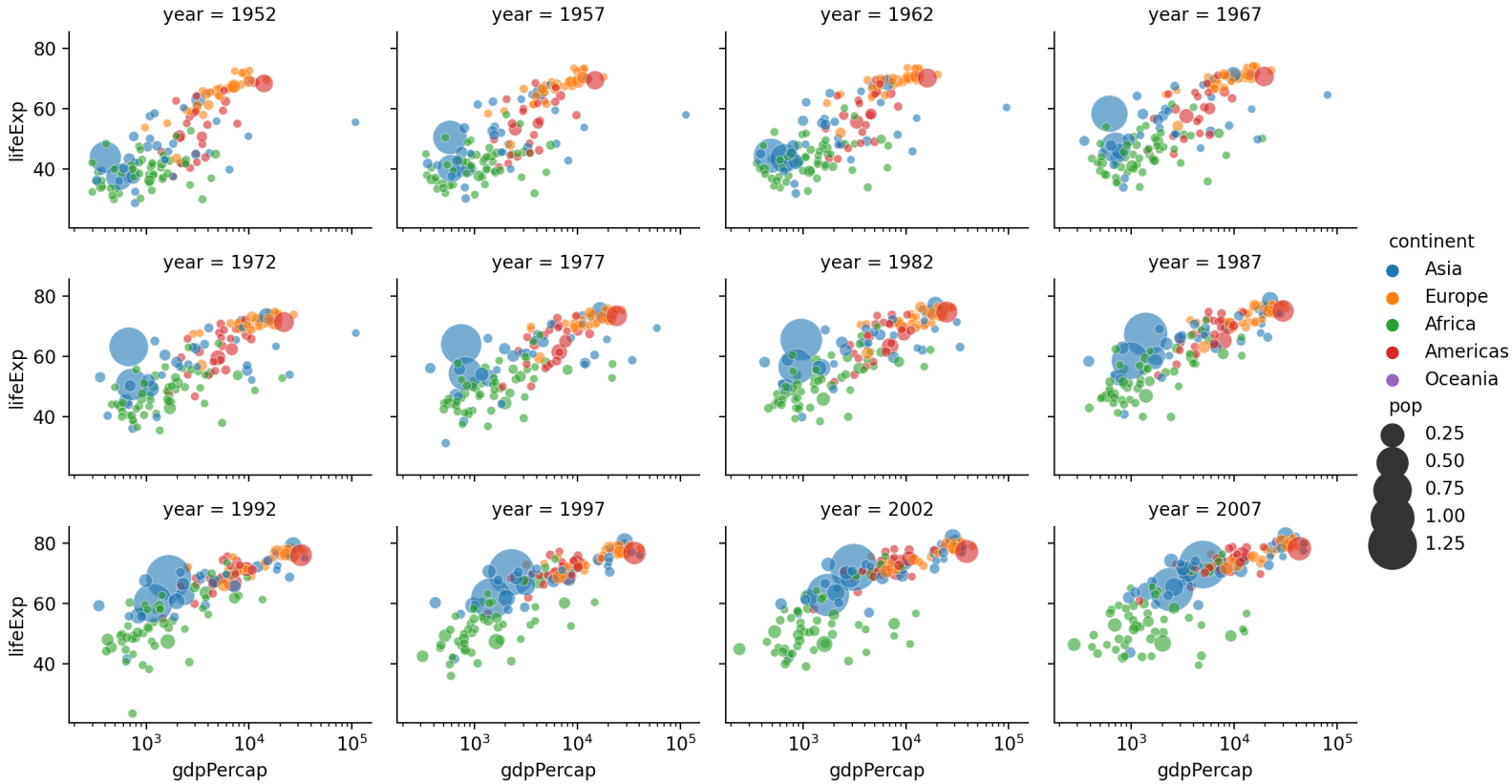
The result of the above code is that `fgrid` is now a `FacetGrid` object (we'll see why it's called that in a minute). However, under the hood, we are still dealing with Matplotlib. To retrieve the individual plotting axis, we can access it from the `fgrid.axes` attribute, which is a 2d Numpy array containing the axes. For example, to make the horizontal axis logarithmic and labels, we can do this:

In [4]:
```
fgrid = sns.relplot(x = "gdpPercap",
                    y = "lifeExp",
                    hue = "continent",
                    data = gapminder)

fgrid.axes[0][0].semilogx()

labels = fgrid.axes[0][0].set(xlabel = "GDP per capita",
                              ylabel = "Life Expectancy",
                              title  = "Impact of Economic Output on Life Expectancy")
```

Not bad! Note that seaborn has effectively looped over the different continents, without us needing to use `for` -loops or `apply` .

Part A

Create the following plot using Seaborn:



In this figure, each point represents a country, and the size of the point corresponds to the population.

In this part and the next, it is not necessary that your plot *precisely* match mine -- feel free to modify the sizes of labels, change the color scheme, etc.

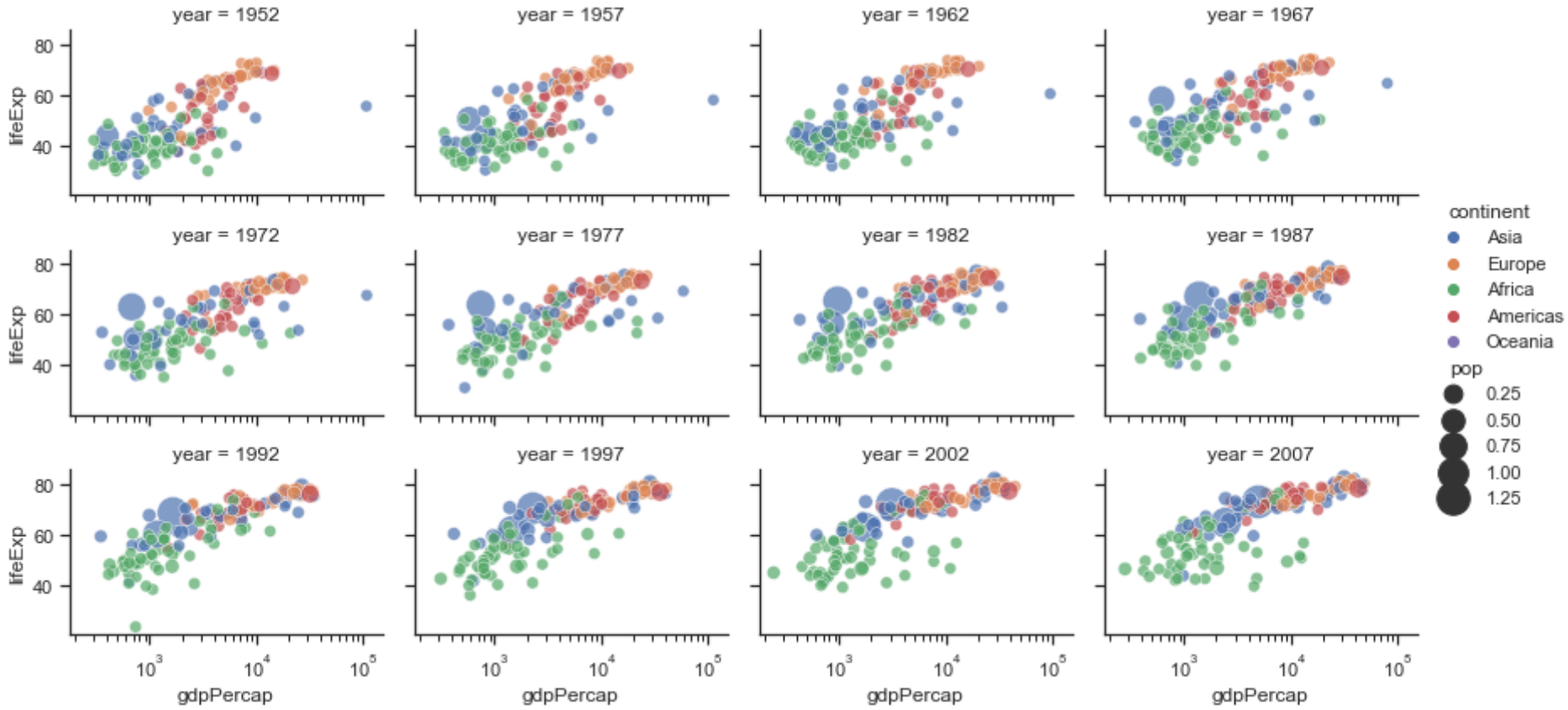
Hints

- You should use `sns.relplot` to create the figure.
- Look in the Seaborn Gallery for examples illustrating how to create a multi-panel figure without using `plt.subplots()`.
- The size of the figure is controlled by the `height` and `aspect` ratio arguments to `relplot`. I used `height = 2` and `aspect = 1.3`, but other choices are also fine.

- When using column wrapping to create the `3 x 4` axis array, the resulting `fgrid.axes` array is no longer 2-dimensional, but 1-dimensional.

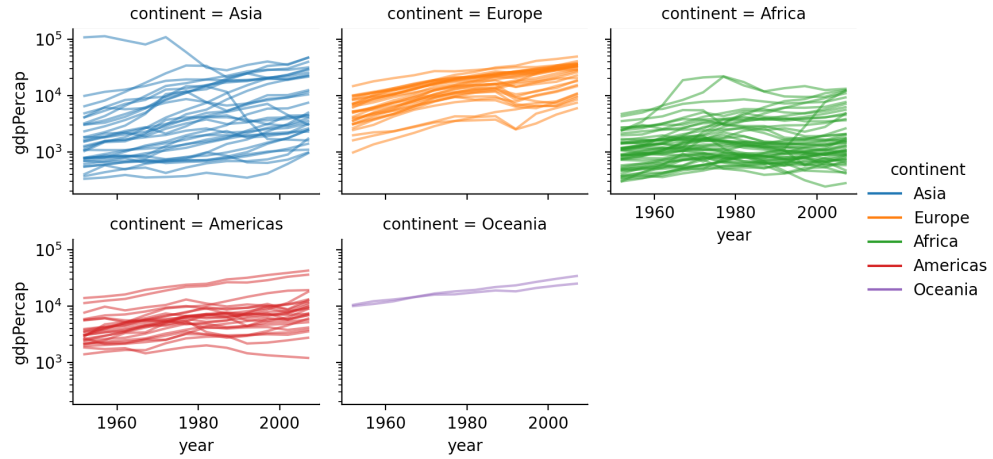
In [28]:
```
sns.set_theme(style="ticks") #ticks theme

fgrid = sns.relplot(data=gapminder,
            x="gdpPercap",
            y="lifeExp",
           hue = "continent",
           col="year",
            size = "pop",
            sizes = (50,400),
            alpha = .7,
           col_wrap=4,
           height=2,
           aspect=1.5)
fgrid.set(xscale="log")
```

Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x19b1589d910>
```



Part B

Create the following plot using Seaborn:

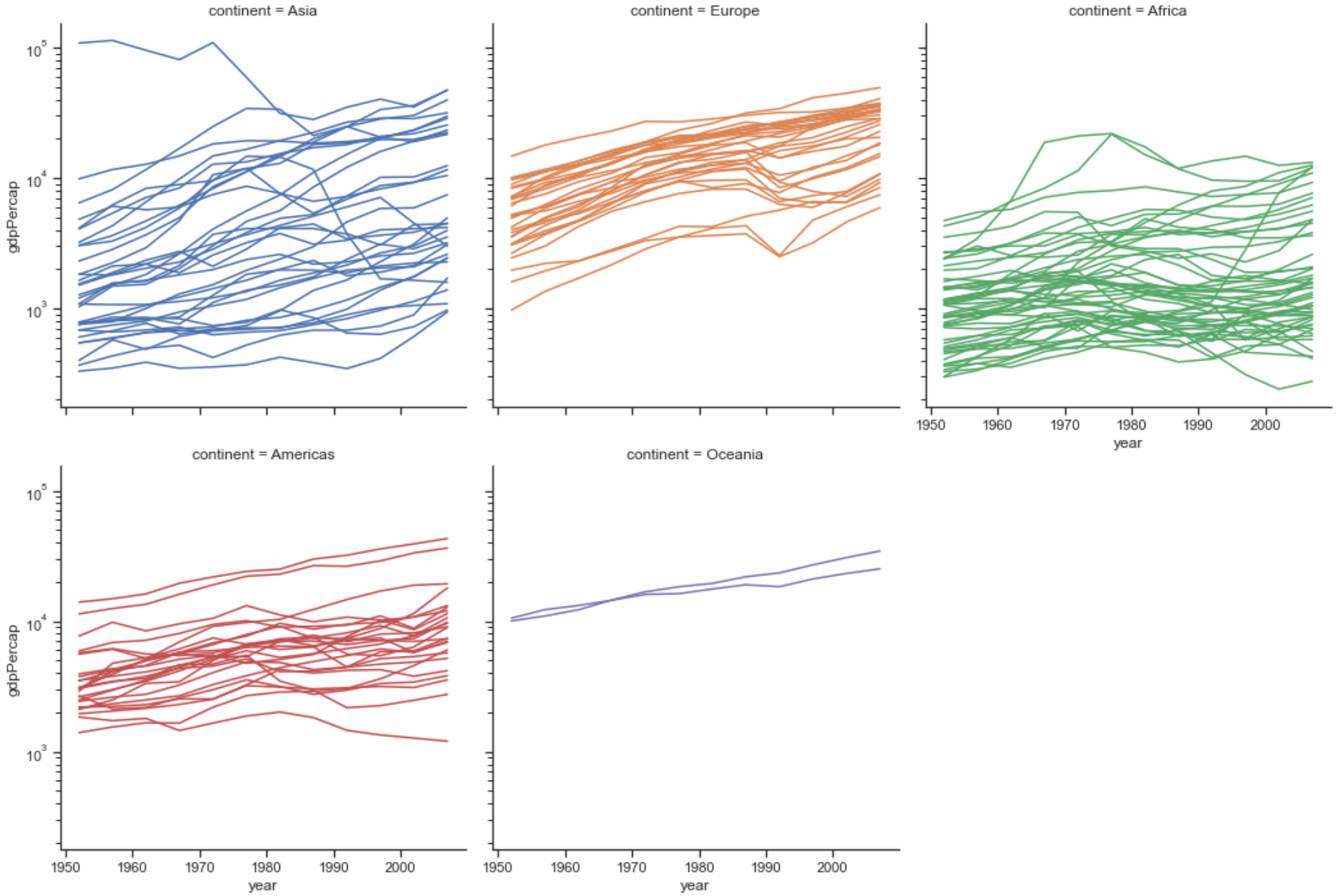


Hints

- The easiest way is to again use `relplot` . This function defaults to producing scatterplots, but if you check the gallery you'll find examples of how to specify keywords that create line charts instead.

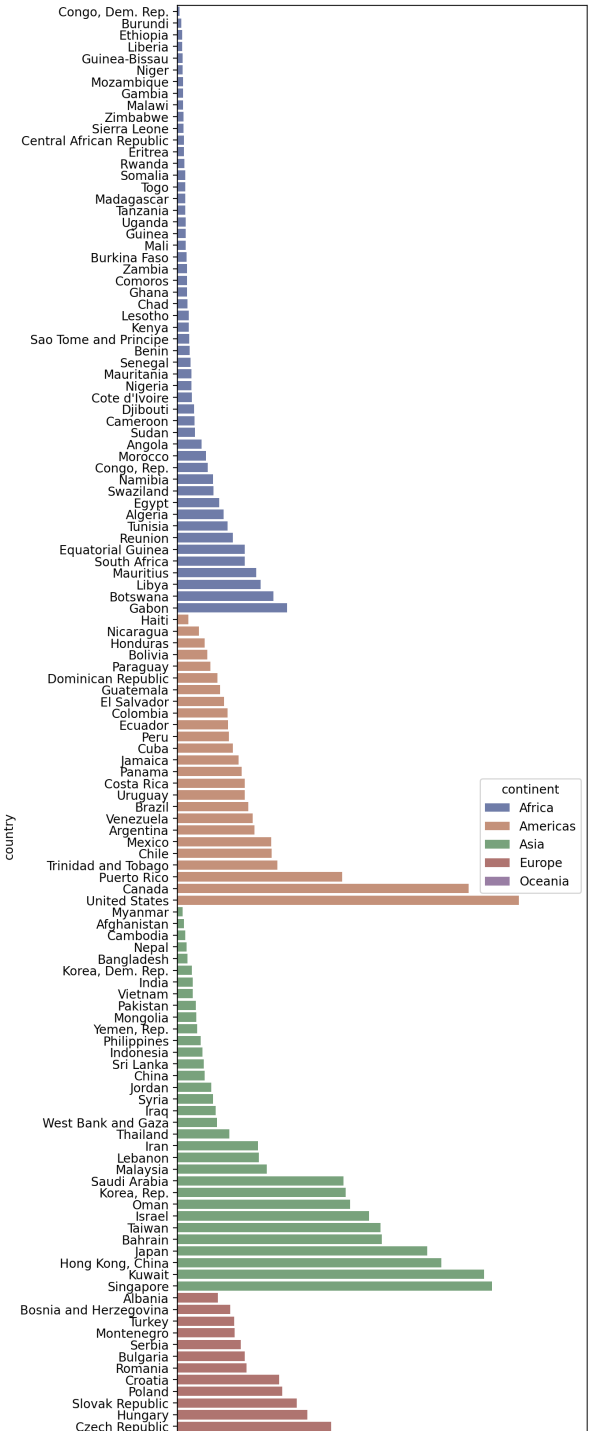
In [23]:
```
sns.set_theme(style="ticks") #ticks theme

fgrid = sns.relplot(data=gapminder,
            x="year",
            y="gdpPercap",
            hue = "continent",
            col = "continent",
            col_wrap = 3,
            kind = "line",
            aspect = 1,
            style="country",
            dashes=False,
            legend=False
        )
fgrid.set(yscale="log")
```

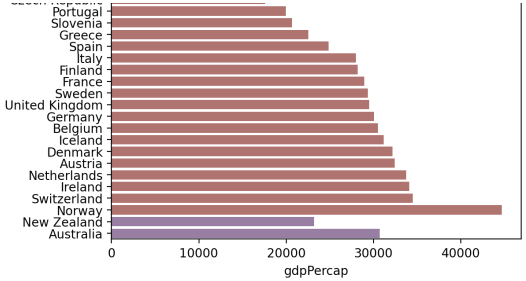
Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x21bced832b0>
```



Part C

Create the following (very long) plot using Seaborn:

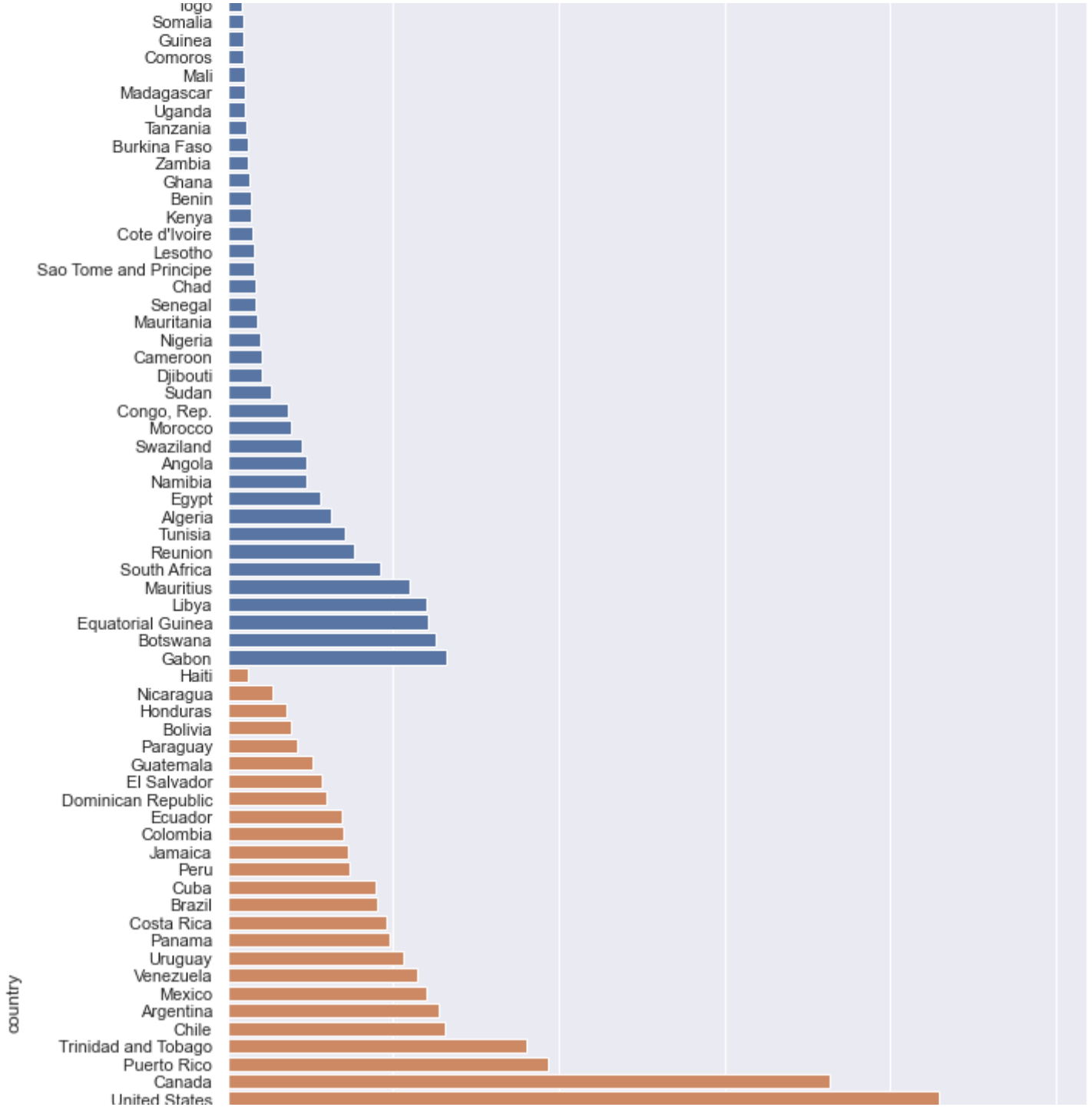

Congo, Dem. Rep.
Burundi
Ethiopia
Liberia
Guinea-Bissau
Niger
Mozambique
Gambia
Malawi
Zimbabwe
Sierra Leone
Central African Republic
Eritrea
Rwanda
Somalia
Togo
Madagascar
Tanzania
Uganda
Guinea
Mali
Burkina Faso
Zambia
Comoros
Ghana
Chad
Lesotho
Kenya
Sao Tome and Principe
Benin
Senegal
Mauritania
Nigeria
Cote d'Ivoire
Djibouti
Cameroon
Sudan
Angola
Morocco
Congo, Rep.
Namibia
Swaziland
Egypt
Algeria
Tunisia
Reunion
Equatorial Guinea
South Africa
Mauritius
Libya
Botswana
Gabon
Haiti
Nicaragua
Honduras
Bolivia
Paraguay
Dominican Republic
Guatemala
El Salvador
Colombia
Ecuador
Peru
Cuba
Jamaica
Panama
Costa Rica
Uruguay
Brazil
Venezuela
Argentina
Mexico
Chile
Trinidad and Tobago
Puerto Rico
Canada
United States
Myanmar
Afghanistan
Cambodia
Nepal
Bangladesh
Korea, Dem. Rep.
India
Vietnam
Pakistan
Mongolia
Yemen, Rep.
Philippines
Indonesia
Sri Lanka
China
Jordan
Syria
Iraq
West Bank and Gaza
Thailand
Iran
Lebanon
Malaysia
Saudi Arabia
Korea, Rep.
Oman
Israel
Taiwan
Bahrain
Japan
Hong Kong, China
Kuwait
Singapore
Albania
Bosnia and Herzegovina
Turkey
Montenegro
Serbia
Bulgaria
Romania
Croatia
Poland
Slovak Republic
Hungary
Czech Republic
country
continent
Africa
Americas
Asia
Europe
Oceania

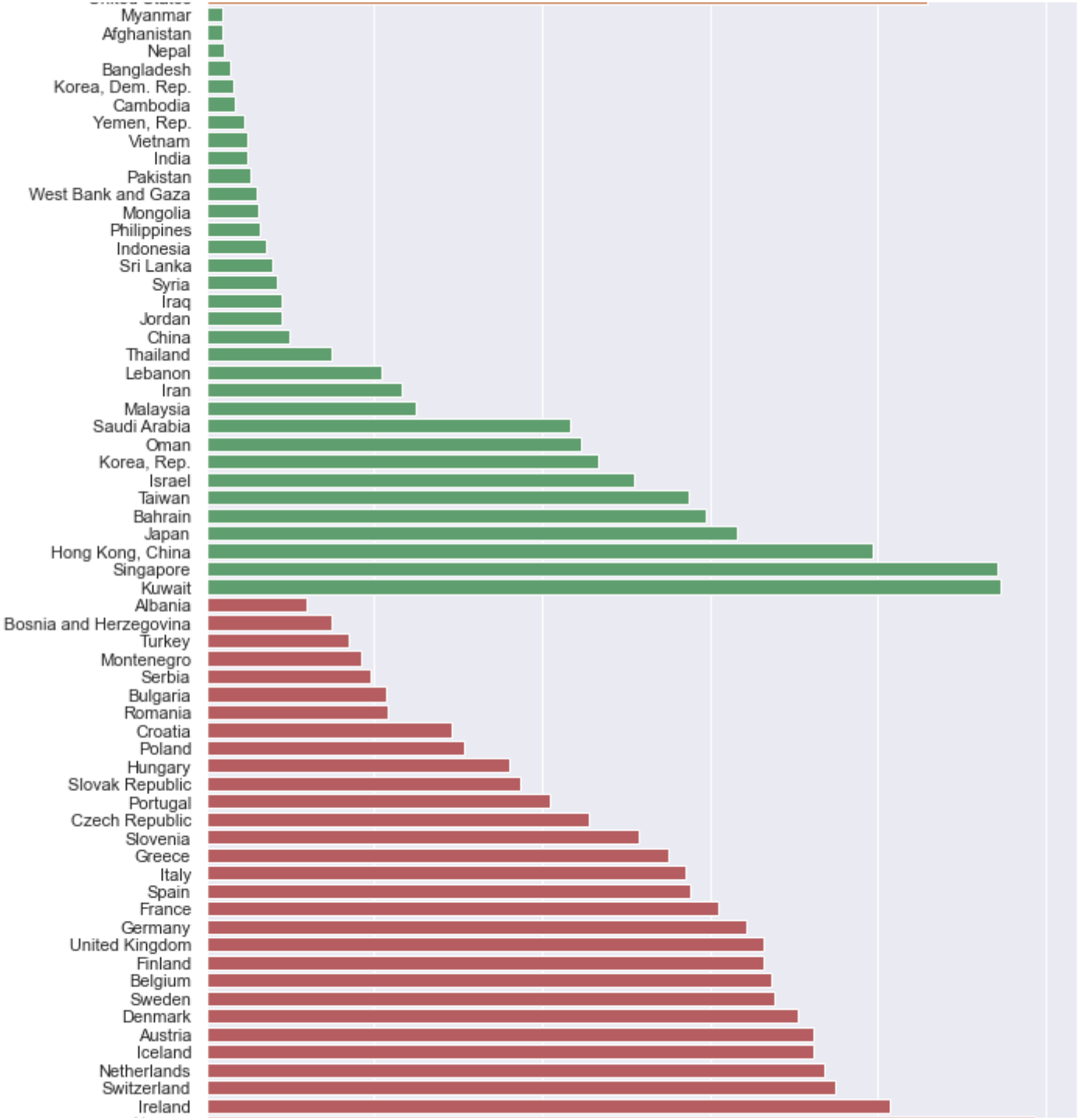


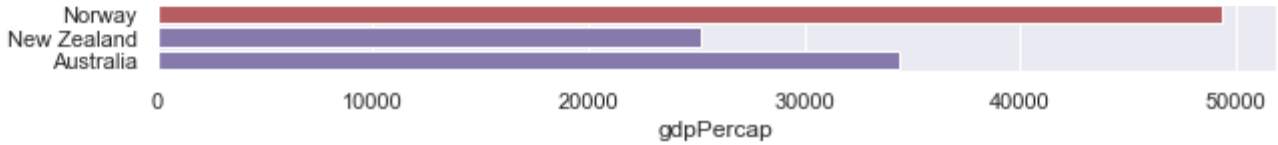
Hints

- Use `sns.barplot` .
- Pick a single year of the `gapminder` data frame to show. In the example, I showed data from 2002 only.
- To sort the bars in ascending order by continent, you should sort the data frame that you pass to `sns.barplot` .
- The argument `dodge = False` is a helpful keyword that will keep your bars centered.
- I used a different color palette just to shake things up a little.

In [79]:
```
mask = gapminder["year"]==2007
df2007 = gapminder[mask]
df2007 = df2007.sort_values(by=['continent','gdpPercap'])
fgrid = sns.barplot(data=df2007,
                    x='gdpPercap',
                    y="country",
                     hue = "continent",
                     dodge = False,
                    )
sns.set(rc = {'figure.figsize':(10,30)})
```

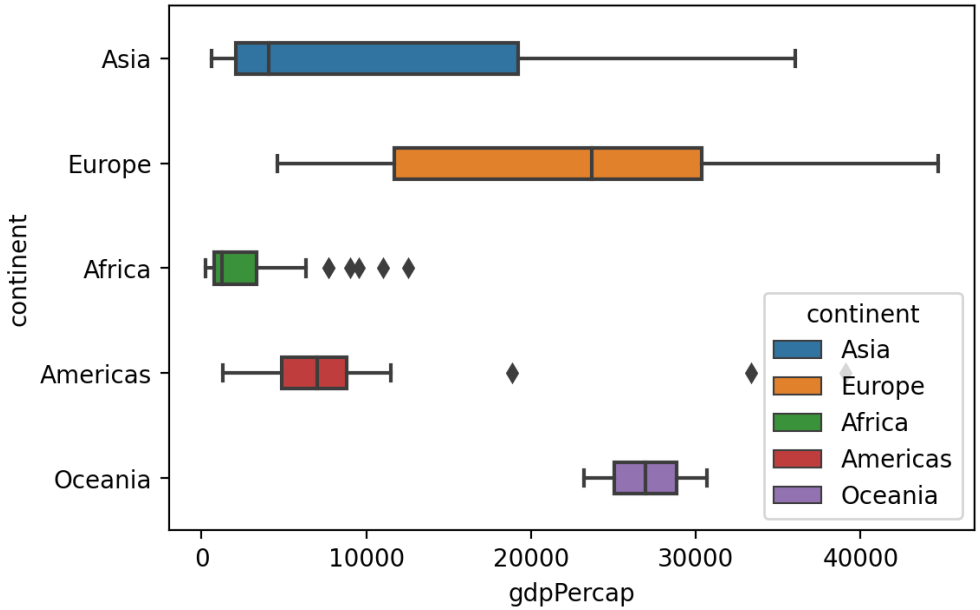

Togo
Somalia
Guinea
Comoros
Mali
Madagascar
Uganda
Tanzania
Burkina Faso
Zambia
Ghana
Benin
Kenya
Cote d'Ivoire
Lesotho
Sao Tome and Principe
Chad
Senegal
Mauritania
Nigeria
Cameroon
Djibouti
Sudan
Congo, Rep.
Morocco
Swaziland
Angola
Namibia
Egypt
Algeria
Tunisia
Reunion
South Africa
Mauritius
Libya
Equatorial Guinea
Botswana
Gabon
Haiti
Nicaragua
Honduras
Bolivia
Paraguay
Guatemala
El Salvador
Dominican Republic
Ecuador
Colombia
Jamaica
Peru
Cuba
Brazil
Costa Rica
Panama
Uruguay
Venezuela
Mexico
Argentina
Chile
Trinidad and Tobago
Puerto Rico
Canada
United States
country


Myanmar
Afghanistan
Nepal
Bangladesh
Korea, Dem. Rep.
Cambodia
Yemen, Rep.
Vietnam
India
Pakistan
West Bank and Gaza
Mongolia
Philippines
Indonesia
Sri Lanka
Syria
Iraq
Jordan
China
Thailand
Lebanon
Iran
Malaysia
Saudi Arabia
Oman
Korea, Rep.
Israel
Taiwan
Bahrain
Japan
Hong Kong, China
Singapore
Kuwait
Albania
Bosnia and Herzegovina
Turkey
Montenegro
Serbia
Bulgaria
Romania
Croatia
Poland
Hungary
Slovak Republic
Portugal
Czech Republic
Slovenia
Greece
Italy
Spain
France
Germany
United Kingdom
Finland
Belgium
Sweden
Denmark
Austria
Iceland
Netherlands
Switzerland
Ireland



Part D

Create the one last plot using Seaborn:

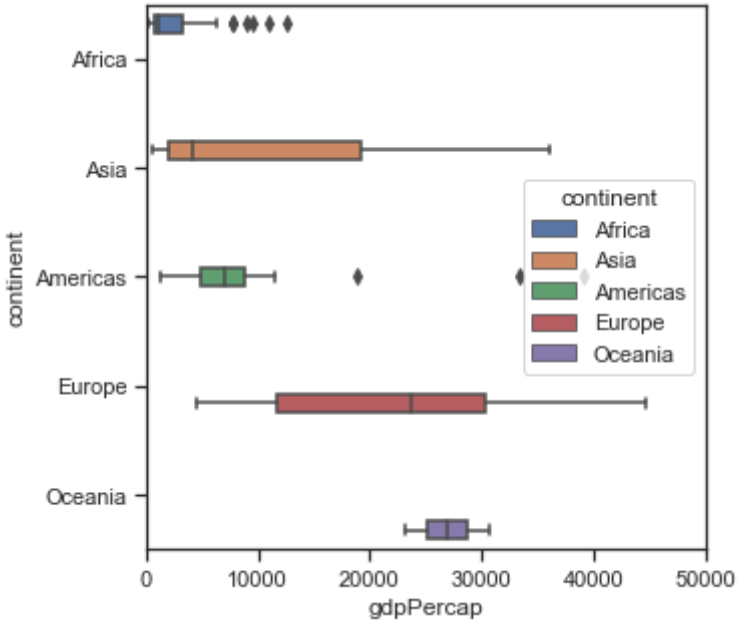


Hints

- This is a boxplot.
- You should again pick a single year of the `gapminder` data frame to show. In the example, I show data from 2002 only.

In [78]:
```
sns.set_theme(style="ticks") #ticks theme
mask = gapminder["year"]==2002
df2002 = gapminder[mask]
df2002 = df2002.sort_values(by=['gdpPercap'])
```

```
df2002
fgrid = sns.boxplot(data = df2002,
                    x="gdpPercap",
                    y="continent",
                     hue="continent",
                    )
fgrid.set_xlim(0, 50000)
sns.set(rc = {'figure.figsize':(8,8)})
```



Problem 3

In our first lecture on machine learning, we did linear regression "by hand." In this problem, we will similarly perform logistic regression "by hand." This homework problem is closely parallel to the lecture, and so you might want to have the notes handy.

Although logistic regression is a relatively simple model, it is a foundation for many modern deep neural nets. Additioanlly, logistic regression itself is still often used in applied contexts even now.

> "When we raise money it's AI, when we hire it's machine learning, and when we do the work it's logistic regression."

> (I'm not sure who came up with this but it's a gem 💎)
>
> — Dr. Daniela Witten (@daniela_witten) September 26, 2019

Whereas linear regression is often used to measure arbitrary quantities like GDP or happiness scores, logistic regression is primarily used to estimate *probabilities*. For example, we might use logistic regression to estimate the probability of a passenger surviving the Titanic crash, a borrower defaulting on a loan, or an email being spam.

For concreteness, let's say that we are considering the latter case. Suppose that we wish to model the probability that an email is spam as a function of the proportion of flag words (like "investment", "capital", "bank", "account", etc.) in the email's body text. Call this proportion x. x is then a variable between 0 and 1.

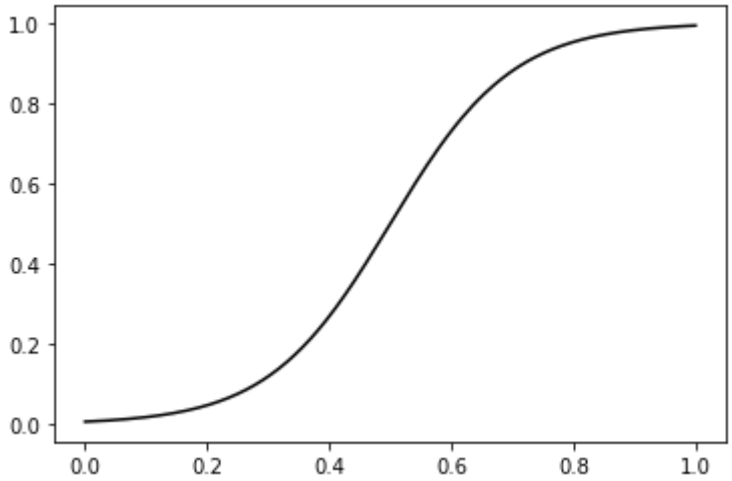
In logistic regression, we suppose that the probability p that an email is spam has the form

$$p = \frac{1}{1 + e^{-ax-b}} \ ,$$

where a and b are again parameters. Let's see how this looks.

In [3]:
```
# run this block

import numpy as np
from matplotlib import pyplot as plt

n_points = 1000

a = 10
b = -5

x = np.sort(np.random.rand(n_points))
p = 1/(1+np.exp(-a*x - b))

fig, ax = plt.subplots(1)
ax.plot(x, p, color = "black")
#print(x)
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x28e82e5be20>]
```



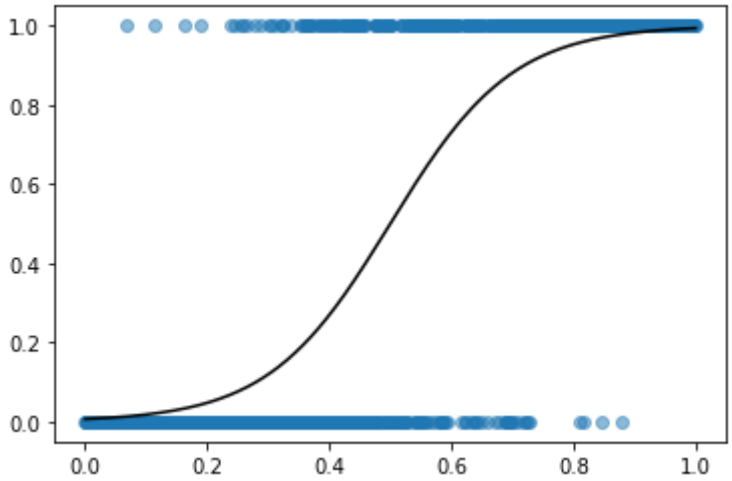
As usual, in practice we don't have access to the true function telling us the probability that an email is spam. Instead, we have access to data telling us whether or not the email really IS spam. We can model this situation by flipping a biased coin for each email, with the probability of heads determined by the modeled probability.

In [4]:
```
# run this block
y = 1.0*(np.random.rand(n_points) < p)
#print(y)
```

A value of 1 indicates that the email is indeed spam, while a value of 0 indicates that the email is not spam.

In [5]:
```
# run this block
ax.scatter(x, y,  alpha = 0.5)
fig
```

Out[5]:

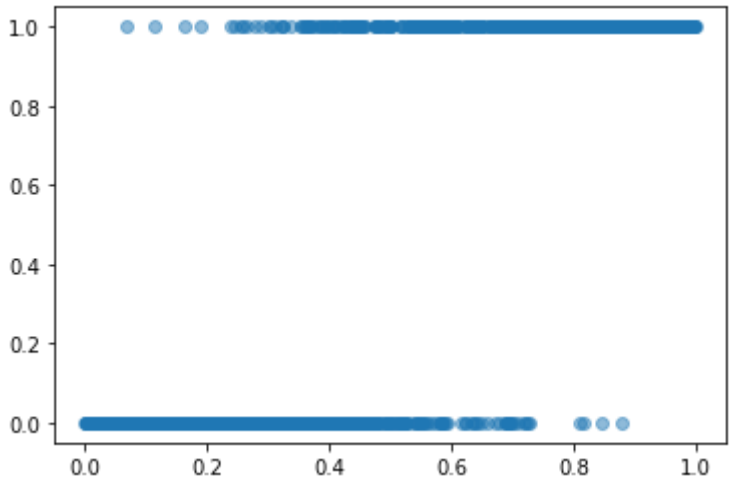


Notice that there are more spam emails where the model gives a high probability, and fewer where the model gives a lower probability. However, there may be some non-spam emails with even high probability -- sometimes we get legitimate emails about bank accounts, investments, etc.

Of course, we don't have access to the true model, so our practical situation looks more like this:

In [6]:
```
# run this block
fig, ax = plt.subplots(1)
ax.scatter(x, y, alpha = 0.5)
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x28e83601d90>
```



We would like to use logistic regression to try to recover something close to the true model.

Part A

Write the model function `f`. The arguments of `f` should be the predictor variables `x` and the parameters `a` and `b`. The output of `f` should be the spam probabilities under the logistic model (see equation above) for these data. Use `numpy` tools, without `for`-loops. If you scan the above code carefully, you'll observe that most of this code is already written for you.

This is a simple function, but **please add a short docstring indicating** what kinds of input it accepts and how to interpret the output.

Comments are necessary only if your function body exceeds one line.

In [7]:
```
# your solution
def f(x,a,b):
    """
    Function that models the likelihood of an email being spam from the predictor variable x and the parameters a and b
    Param x: predictor variable that is a numpy array of random numbers between 0 and 1
    Param a: parameter int/float that is varied in value in order to optomize the accuracy
        of the model for the predictor and target data
    Param b: parameter int/float that is varied in value in order to optomize the accuracy
        of the model for the predictor and target data
    Returns: numpy array of predictions based on logarithmic distribution
    """
    return 1/(1+np.exp(-a*x - b))
```

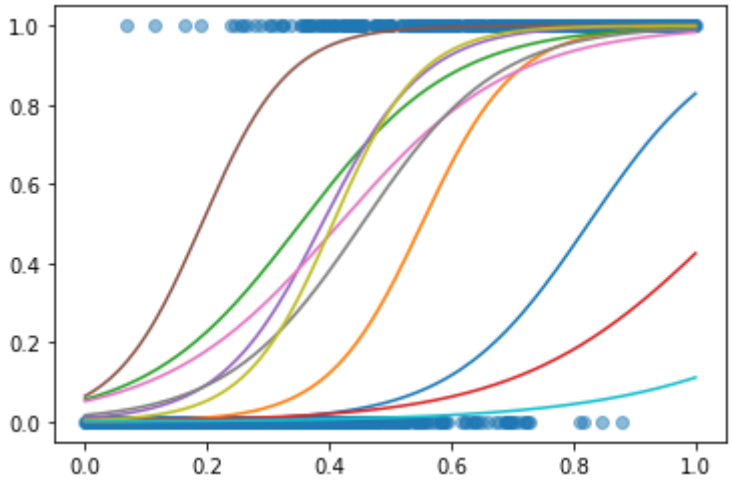
Part B

Plot 10 candidate models against the data, using randomly chosen values of `a` between 5 and 15 and randomly chosen values of `b` between -2.5 and -7.5. Your plot should resemble in certain respects the third plot in these lecture notes.

Comments are not necessary in this part.

In [8]:
```
# your solution here
fig,ax=plt.subplots(1)
ax.scatter(x, y, alpha = 0.5)

for i in range(10):
```

```
    a = np.random.randint(5,15)
    b = np.random.uniform(-7.5,-2.5)
    ax.plot(x,f(x,a,b))
```



Part C

The *loss function* most commonly used in logistic regression is the *negative cross-entropy*. The negative cross-entropy of the `i` th observation is

$$-\left[y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)\right]$$

where $y_i \in \{0,1\}$ is the `i` th entry of the target data and $\hat{p}_i$ is the model's estimated probability that $y_i = 1$. The negative cross-entropy of the entire data set is the sum of the negative cross-entropies for each individual observation.

Write a function that computes the negative cross entropy as a function of `x`, `y`, `a`, and `b`. This can be done in no more than two lines using `numpy`, without `for`-loops. Don't forget which logarithm is #BestLogarithm.

As in Part B, please write a short docstring describing what your function does and what inputs it accepts. Comments are necessary only if your function body exceeds two lines.

In [9]:

```
# your solution here
def logNCE(x,y,a,b):
    """
    Function that computes the negative cross-entropy of the ith observation. Basically returns the
```

```
    accuracy of the model on the ith observation
Param x: predictor variable that is a numpy array containing the data of random numbers btwn 0 and 1
Param y: a numpy array containing the model's target which is whether an email was not-spam or spam (0 or 1)
Param a: int/float parameter that model changes to optomize for the data
Param b: int/float parameter that model changes to optomize for the data
Returns: a float representing negative cross-entropy value for model with given a and b parameters
"""
arrayNegCrossEnt = -1*(y*(np.log(f(x,a,b)))+(1-y)*np.log(1-f(x,a,b)))
return arrayNegCrossEnt.sum()
```

Part D

On a single axis, plot 100 distinct models (using the code that you wrote in) in Part B. Highlight the one with the lowest negative cross entropy in a different color -- say, red. Compare the best values of `a` and `b` that you found to the true values, which were `a = 10` and `b = -5`. Are you close?

The plot you produce should resemble, in some respects, the fifth plot in these lecture notes.

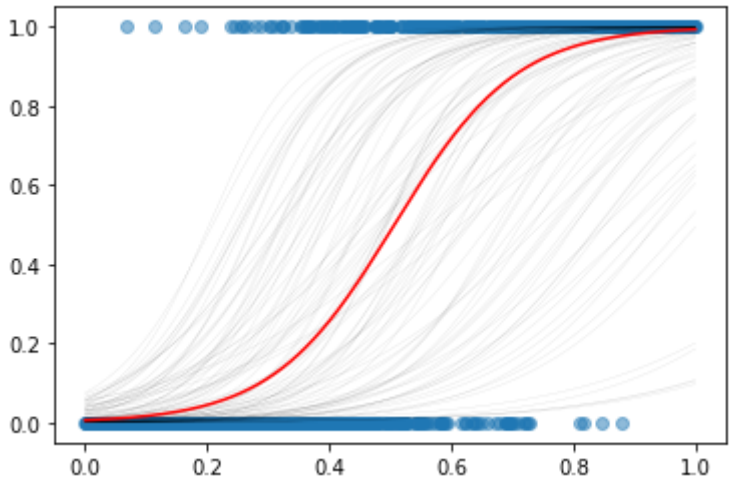
It is not necessary to write comments in this part.

In [10]:

```
# your solution here
fig,ax=plt.subplots(1)
ax.scatter(x, y, alpha = 0.5)

bestNCE = np.inf

for i in range(100):
    a = np.random.randint(5,15)
    b = np.random.uniform(-7.5,-2.5)
    prediction = f(x,a,b)
    NCE = logNCE(x,y,a,b)

    ax.plot(x,prediction,color="black",alpha=.25,linewidth=.15)

    if(NCE < bestNCE):
        bestNCE = NCE
        best_a = a
        best_b = b
```

```
bestRun = f(x,best_a,best_b)
ax.plot(x,bestRun,color="red")
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x28e8381a580>]
```



In [ ]:

It is not required to use `scipy.optimize` to more accurately estimate `a` and `b` for this homework assignment, but you are free to do so if you wish. You may then use the optimal estimates in the following part.

Part E

In classification tasks, we evaluate not just the standard loss function, but also the *accuracy* -- how often does the model correctly classify the data? Let's say that the model classifies an email as spam according to the following rule:

1. If $\hat{p}_i$ (the model probability plotted above) is larger than c, classify the email as spam.
2. If $\hat{p}_i$ is less than or equal to c, classify the email as not-spam.

Write a function called `positive_rates` which accepts the following arguments:

1. The data, `x` and `y`.
2. The best parameters `best_a` and `best_b`.

3. A threshold `c` between 0 and 1.

This function should output two numbers. The first of these is *false positive rate*: the proportion of non-spam emails that the model incorrectly labels as spam. The second is the *true positive rate*: the proportion of spam emails that the model correctly labels as spam.

For example:

```
positive_rates(x, y, best_a, best_b, c = 0.5)
```

```
(0.05058365758754864, 0.7469135802469136)
```

Note: due to randomization, your numerical output may be different.

Please write a descriptive docstring for your function. Comments are necessary only if your function body exceeds five lines.

In [11]:

```
# your solution here
def positive_rates(x,y,best_a,best_b,c):
    """
    Function that computes the accuracy of log regression model by computing the false positive and the
    true positive rates of email classification
    Param x: numpy array of random numbers between 0 and 1 representing the predictor data
    Param y: a numpy array containing the model's target prediction (w vals either 0 or 1 representing not-spam/spam)
    Param best_a: an integer representing the best tested for parameter to input in the model for lowest negative cross entropy
    Param best_b: a float representing the best tested for parameter to input in the model for lowest negative cross entropy
    Param c: probability threshold between 0 and 1 to classify the email as spam or not spam
    Returns: two doubles. First represents false positive rate and second represents true positive rate
    """
    probability = f(x,best_a,best_b) #create variable for numpy array of predictions
    incorrectSpam = 0 #counter variable for false positives
    correctSpam = 0 #counter variable for true positives
    for i in range(len(probability)): #iterate through each each guess
        classification = (probability[i] > c) #check whether model classified as not-spam (false) or spam (true)
        if(classification==1 and y[i]==0): #classifies as spam when not spam
            incorrectSpam+=1
        elif(classification==1 and y[i]==1): #classifies as spam and is spam
            correctSpam+=1
    return (incorrectSpam/(np.count_nonzero(y))),(correctSpam/(len(y)-np.count_nonzero(y)))
    #count zeros counts the instances of nonspam emails. The numb of values in y minus nonspam emails gives spam
```

In [39]:

```
# demonstrate your function here
positive_rates(x,y,best_a,best_b,c=.5)
```

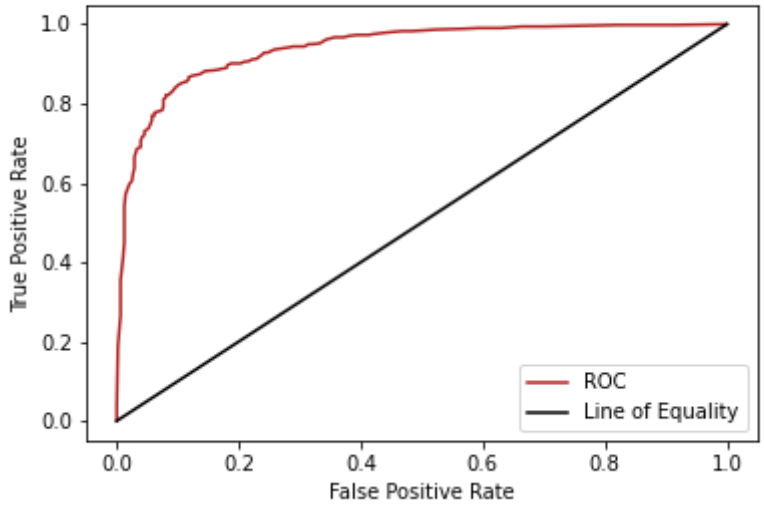
Out[39]: (0.11706349206349206, 0.8709677419354839)

Part F

Plot the *receiver operating characteristic* (ROC) curve for the logistic model with parameters `best_a` and `best_b`. The ROC curve is the plot of the `false_positive` rate (on the horizontal axis) against the `true_positive` rate (on the vertical axis) as the threshold `c` is allowed to vary. Additionally, plot a diagonal line ("the line of equality") between the points (0,0) and (1,1). Your ROC curve should lie noticeably above the line of equality. Plot your curves in different colors and add a legend to help your reader understand the plot.

It is ok to use `for`-loops and list comprehensions in this part.

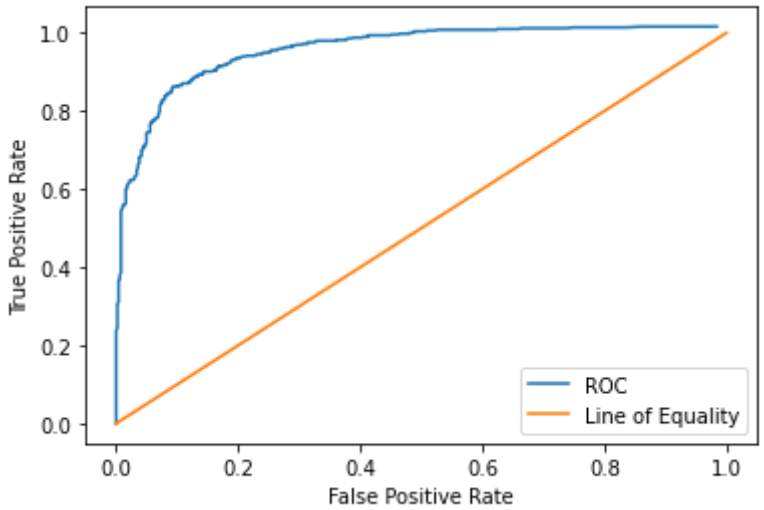
Your result should look something like this, although it won't be exactly the same due to randomness.



In [51]:
```
# plot code here
#create figure
fig,ax=plt.subplots(1)
```

```
#Generating/Extracting/Plotting ROC values
ROCvals = [positive_rates(x,y,best_a,best_b,c) for c in np.linspace(0,1,n_points)] #trying n_points values of c btwn 0 and 1
xROC = [ROCvals[i][0] for i in range(n_points)] #extracting x values from tuples in list ROCvals
yROC = [ROCvals[i][1] for i in range(n_points)] #extracting y values from tuples
line, = ax.plot(xROC,yROC,label="ROC") #plotting

#plotting Line of Equality
x1=np.linspace(0,1,100)
y1=np.linspace(0,1,100)
line2, = ax.plot(x1,y1,label="Line of Equality")

#Annotations
ax.set(xlabel="False Positive Rate",
       ylabel="True Positive Rate",
       )
ax.legend(handles=[line,line2])
```

Out[51]: <matplotlib.legend.Legend at 0x28e86099e80>



Generally speaking, a "good" classifier is one that can reach the closest to the top-left corner of the ROC diagram. This is a classifier that can achieve a high rate of true positives, while keeping a low rate of false positives. There are various ways to measure how "good" an ROC curve is, which are beyond our present scope.