Plotting Exercises, Part 1

In [ ]:
```
import pandas as pd
import numpy as np

import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
```

```
c:\Users\kbagh\miniconda3\Lib\site-packages\numpy\_distributor_init.py:30: UserWarni
ng: loaded more than 1 DLL from .libs:
c:\Users\kbagh\miniconda3\Lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_1
0_3_0.dll
c:\Users\kbagh\miniconda3\Lib\site-packages\numpy\.libs\libopenblas64__v0.3.23-246-g
3d31191b-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

Exercise 1

Create a pandas dataframe from the "Datasaurus.txt" file using the code:

Note that the file being downloaded is *not* actually a CSV file. It is tab-delimited, meaning that within each row, columns are separated by tabs rather than commas. We communicate this to pandas with the `delimiter="\t"` option (`"\t"` is how we write a tab, as we will discuss in future lessons).

In [ ]:
```
pd.set_option("mode.copy_on_write", True)

df = pd.read_csv(
    "https://raw.githubusercontent.com/nickeubank/practicaldatascience"
    "/master/Example_Data/Datasaurus.txt",
    delimiter="\t",
)
```

Exercise 2

This dataset actually contains 13 separate example datasets, each with two variables named `example[number]_x` and `example[number]_y`.

In order to get a better sense of what these datasets look like, write a loop that iterates over each example dataset (numbered 1 to 13) and print out the mean and standard deviation for `example[number]_x` and `example[number]_y` for each dataset.

For example, the first iteration of this loop might return something like:

```
Example Dataset 1:
Mean x: 23.12321978429576,
```

```
Mean y: 98.23980921730972,
Std Dev x: 21.2389710287,
Std Dev y: 32.2389081209832,
Correlation: 0.73892819281
```

(Though you shouldn't get those specific values)

In [ ]:
```python
for i in range(1, 14):
    x_tag = f"example{i}_x"
    y_tag = f"example{i}_y"
    temp_df = df[[x_tag, y_tag]].describe().loc[["mean", "std"]]
    print(f"Dataset {i}:")
    print(f"Mean x: {temp_df.iloc[0,0]}")
    print(f"Std dev x: {temp_df.iloc[1,0]}")
    print(f"Mean y: {temp_df.iloc[0,1]}")
    print(f"Std dev y: {temp_df.iloc[1,1]}\n\n")
```

```
Dataset 1:
Mean x: 54.266099784295776
Std dev x: 16.769824954043756
Mean y: 47.834720624943664
Std dev y: 26.9397434192671


Dataset 2:
Mean x: 54.268730022394365
Std dev x: 16.769239493454403
Mean y: 47.83082315530282
Std dev y: 26.935726689918784


Dataset 3:
Mean x: 54.26731970598592
Std dev x: 16.76001265980608
Mean y: 47.83771726725352
Std dev y: 26.930036087838204


Dataset 4:
Mean x: 54.26327323943662
Std dev x: 16.76514203911679
Mean y: 47.832252816901416
Std dev y: 26.935403486939116


Dataset 5:
Mean x: 54.26030345169014
Std dev x: 16.767735488473807
Mean y: 47.839829209014084
Std dev y: 26.93019151853346


Dataset 6:
Mean x: 54.26144178316902
Std dev x: 16.765897903899337
Mean y: 47.83025191366197
Std dev y: 26.93987622043797


Dataset 7:
Mean x: 54.26880527950703
Std dev x: 16.766704015934764
Mean y: 47.83545020401409
Std dev y: 26.939997961411027


Dataset 8:
Mean x: 54.26784882366197
Std dev x: 16.76675894771805
Mean y: 47.83589633112676
Std dev y: 26.936104931679978
```

```
Dataset 9:
Mean x: 54.26588178542254
Std dev x: 16.768852670828494
Mean y: 47.831495652323945
Std dev y: 26.93860807087184


Dataset 10:
Mean x: 54.26734110478873
Std dev x: 16.76895921619445
Mean y: 47.83954522535211
Std dev y: 26.93027468808843


Dataset 11:
Mean x: 54.26992723091549
Std dev x: 16.769958611325382
Mean y: 47.836987988408445
Std dev y: 26.937683806980512


Dataset 12:
Mean x: 54.266916301197185
Std dev x: 16.769999617573024
Mean y: 47.83160198797184
Std dev y: 26.937901927731797


Dataset 13:
Mean x: 54.26015033415493
Std dev x: 16.76995769550748
Mean y: 47.839717279450696
Std dev y: 26.93000168716234
```

Exercise 3

Based only on these results, discuss what might you conclude about these example datasets with your partner. Write down your thoughts.

From only these results, it seems as if the datasets all have a similar distriubtion. I believe this because each has a relatively similar mean and standard deviation for both the x and y variables. This means that each dataset occupies generally the same range and has generally the same distribution.

Execise 4

Write a loop that iterates over these example datasets, and using Altair library, plot a simple scatter plot of each dataset with the `x` variable on the x-axis and the `y` variable on the y-axis.

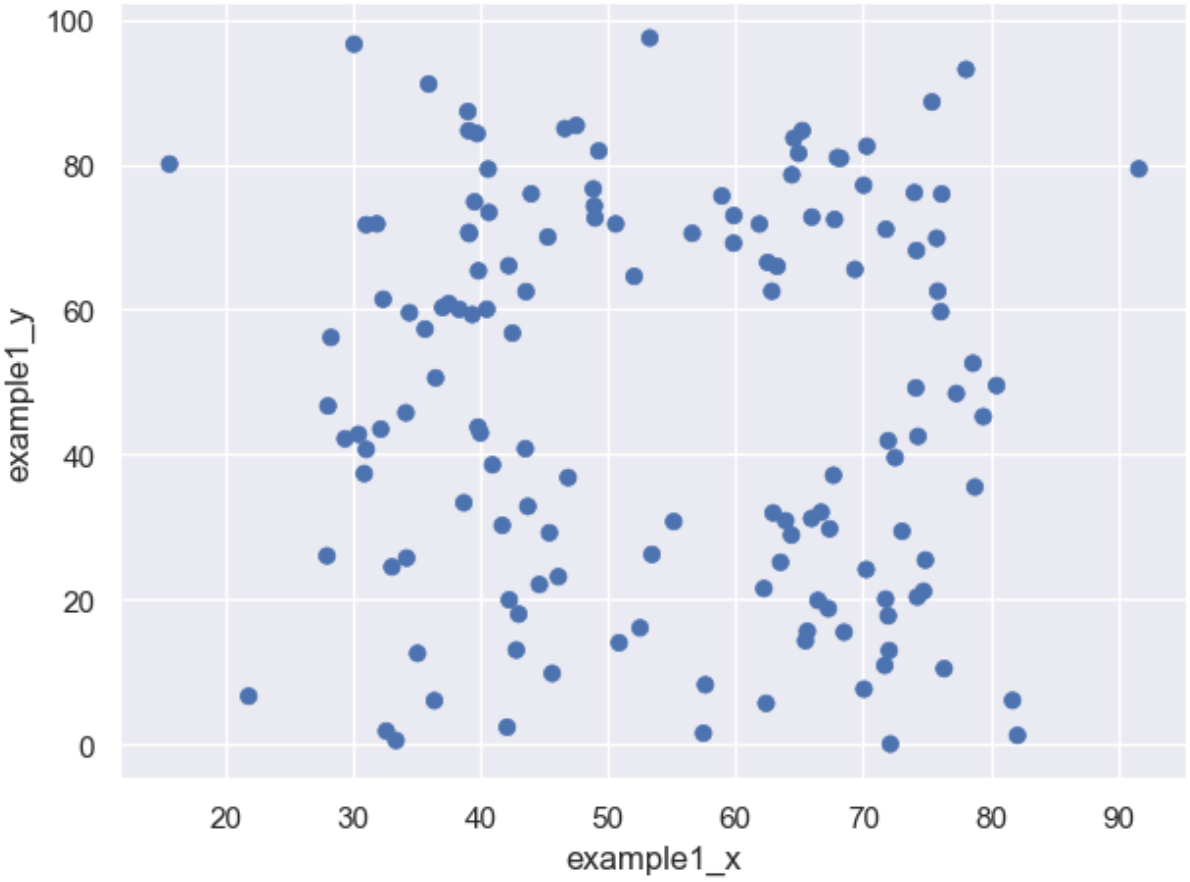
Hint: When writing this type of code, it is often best to start by writing code to do what you want for the first iteration of the loop. Once you have code that works for the first example dataset, then write the full loop around it.

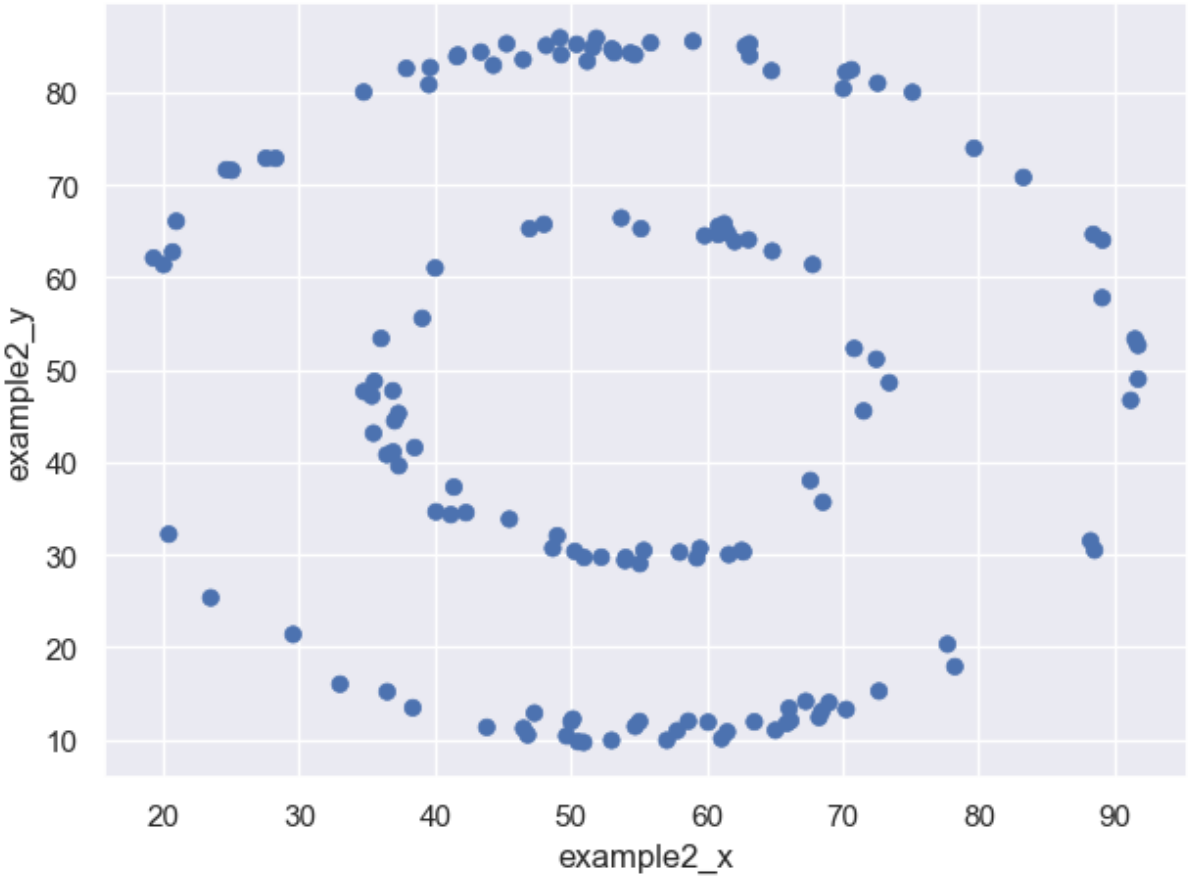
Hint 2: To force Jupyter to display your charts when they're generated within a loop, use the method `.show()` (e.g. `my_chart.show()`).

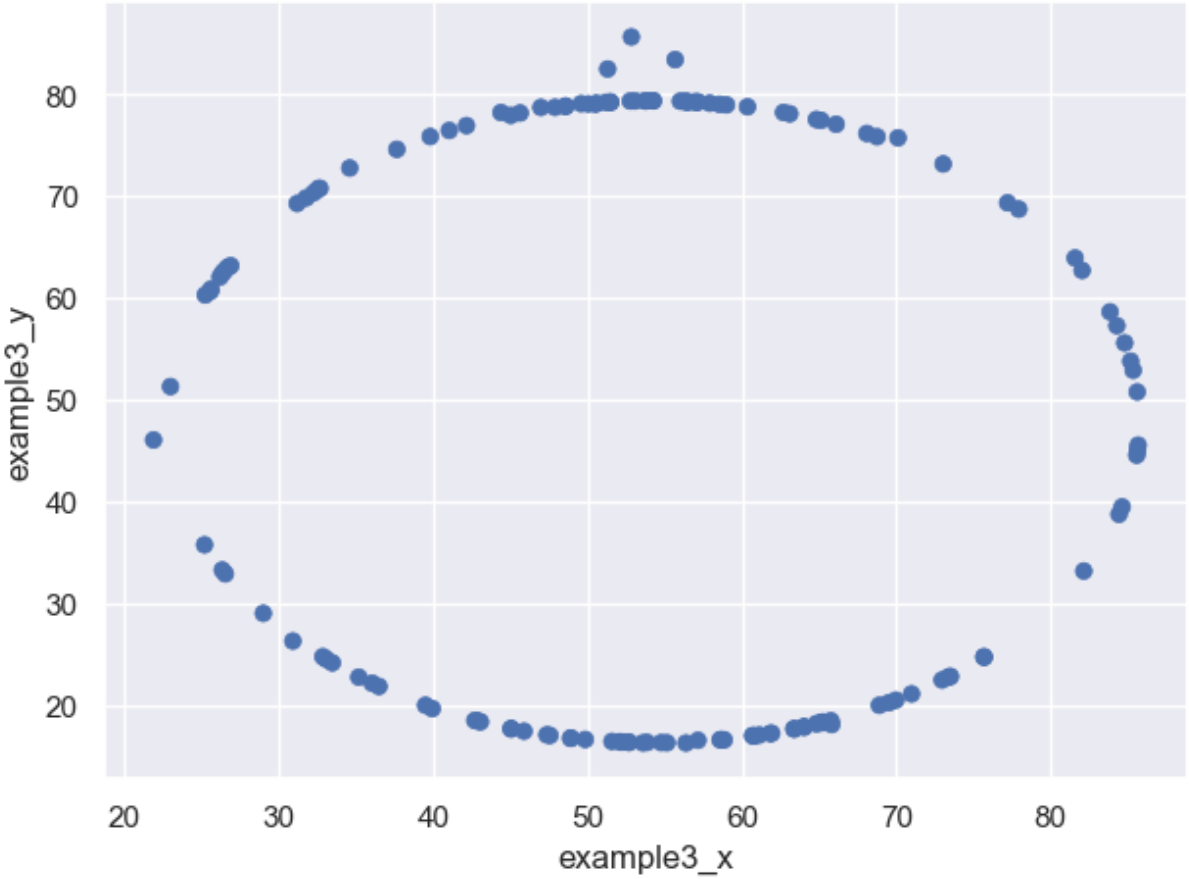
Hint 3: You will need to change the range of the axes to make the plots look good!

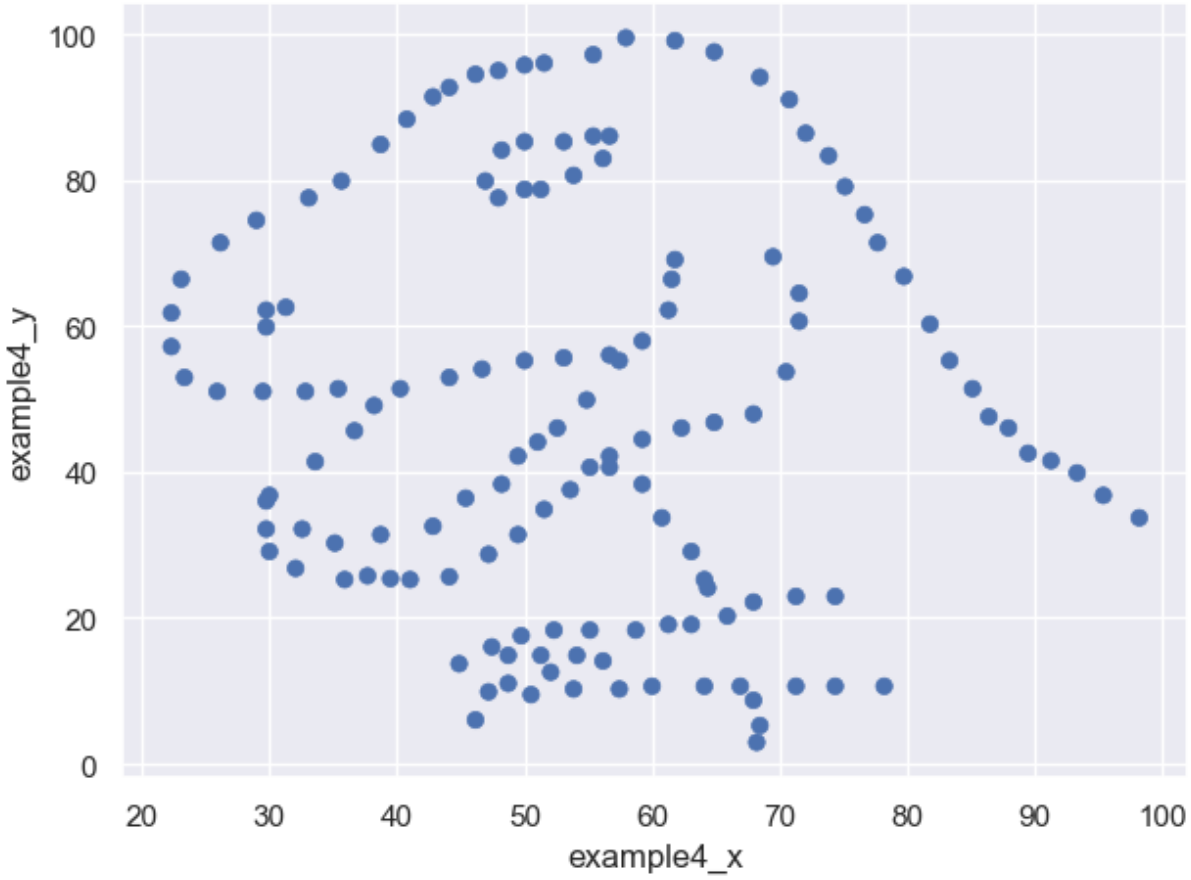
In [ ]:
```
import seaborn.objects as so
```

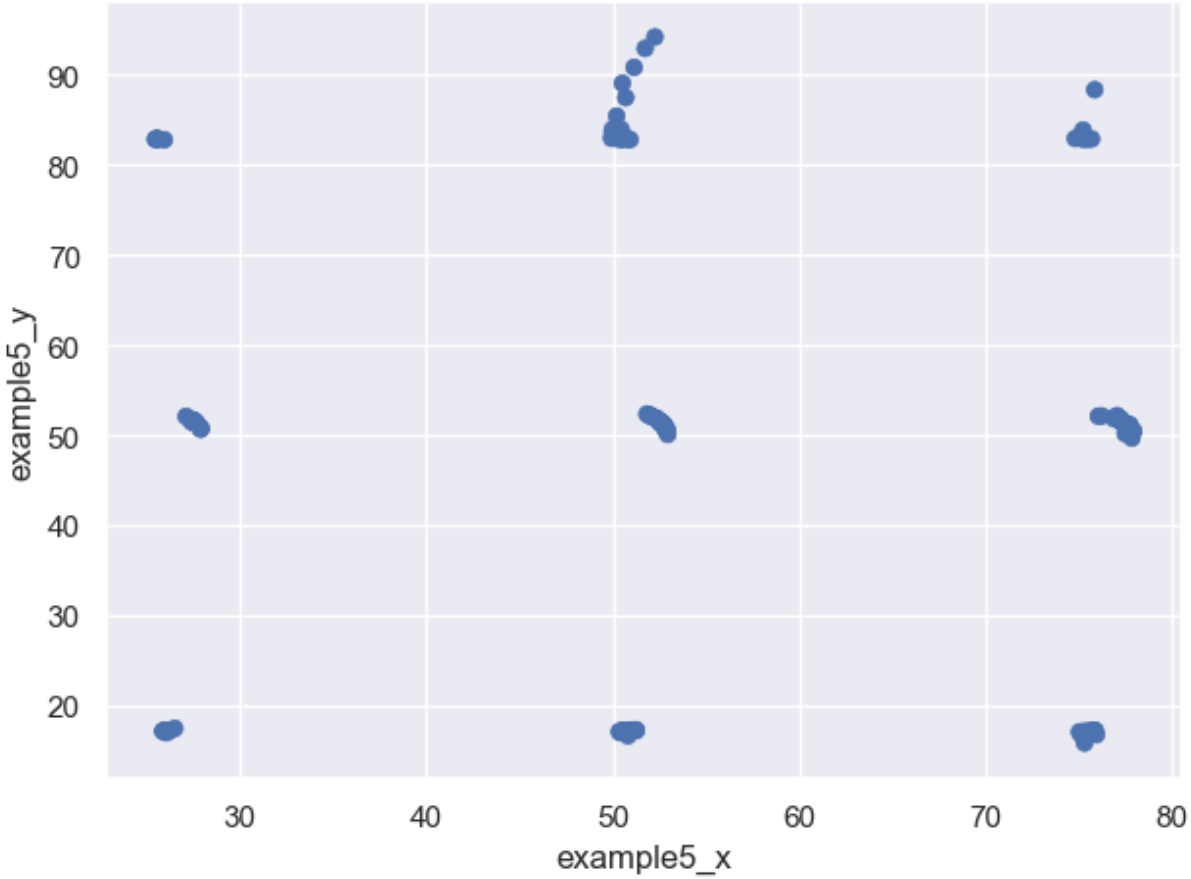
In [ ]:
```
for i in range(1, 14):
    x_tag = f"example{i}_x"
    y_tag = f"example{i}_y"
    my_chart = so.Plot(df, x=x_tag, y=y_tag).add(so.Dot())
    my_chart.show()
```

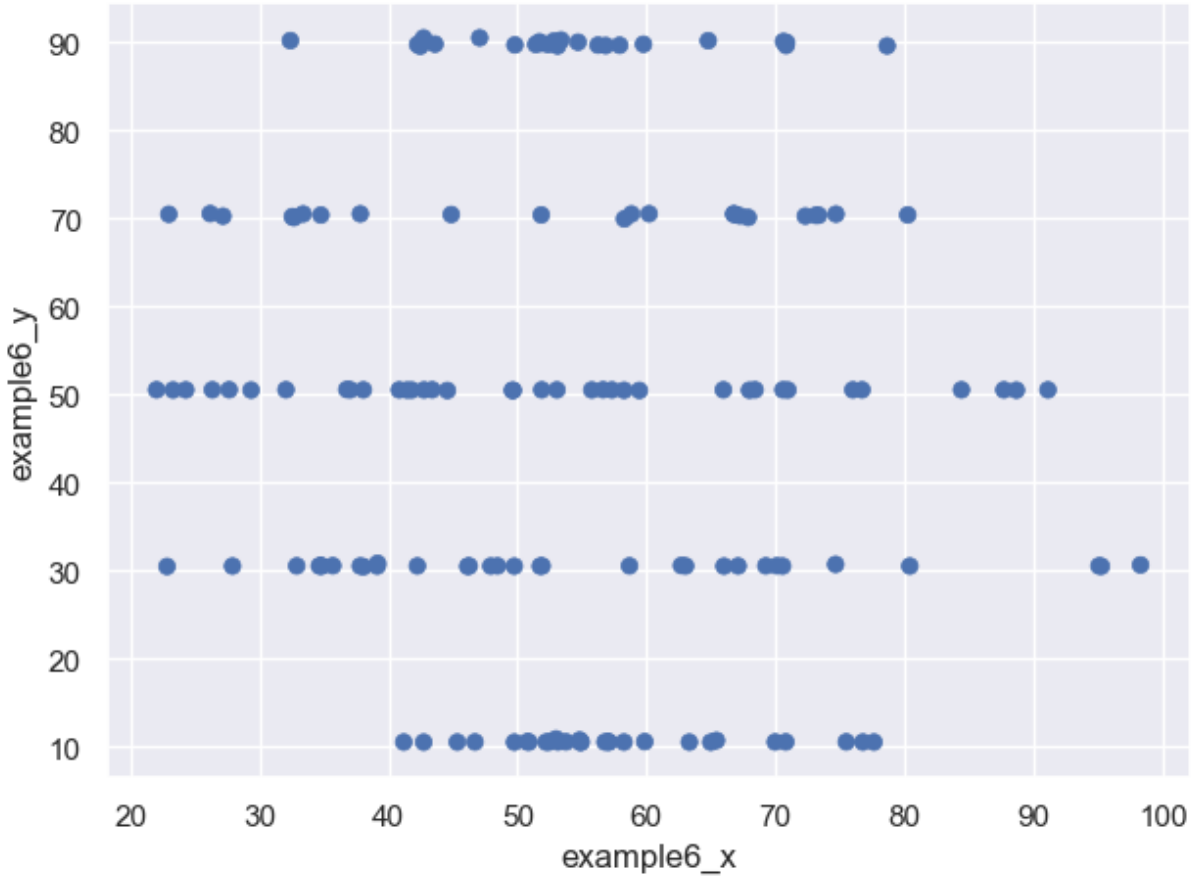


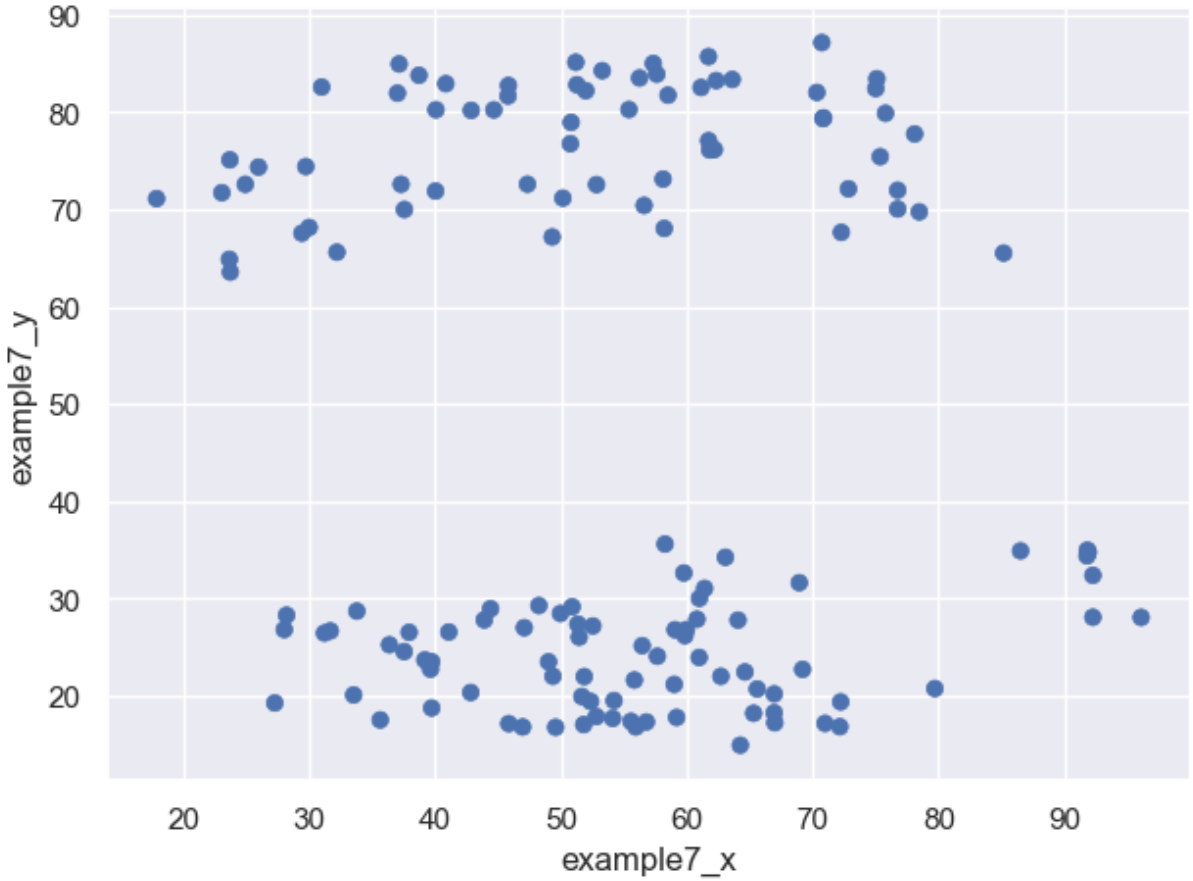

example2_y
example2_x
10
20
30
40
50
60
70
80
20
30
40
50
60
70
80
90

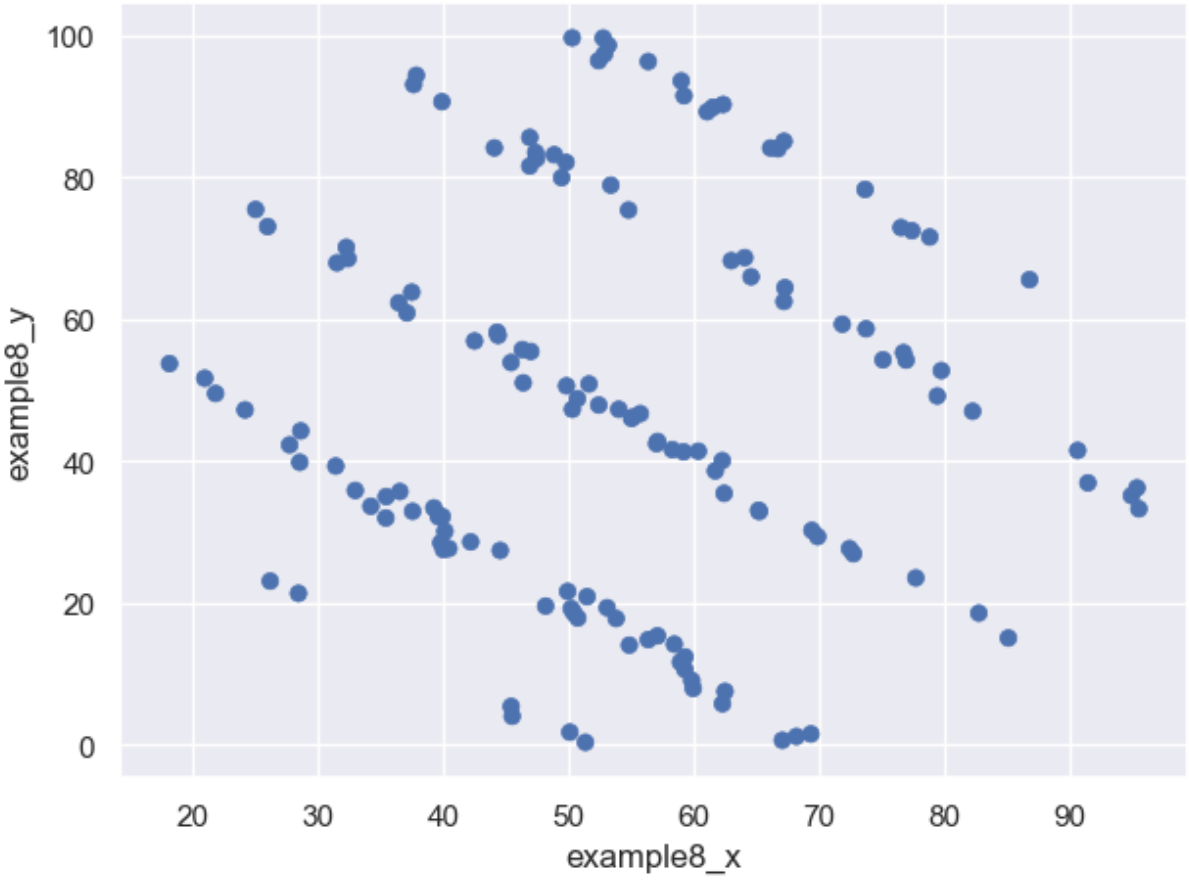

example3_y
example3_x
20
30
40
50
60
70
80
20
30
40
50
60
70
80

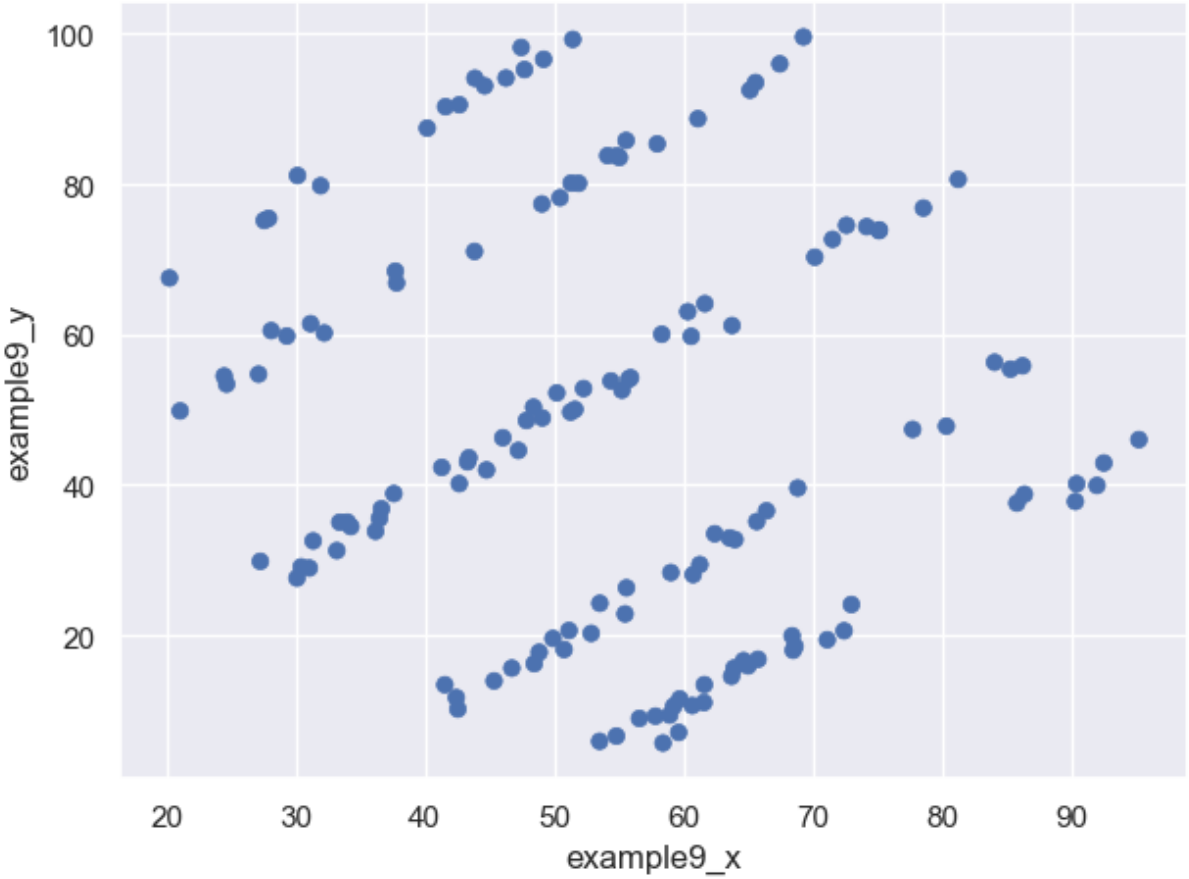

example4_y
example4_x
0
20
40
60
80
100
20
30
40
50
60
70
80
90
100

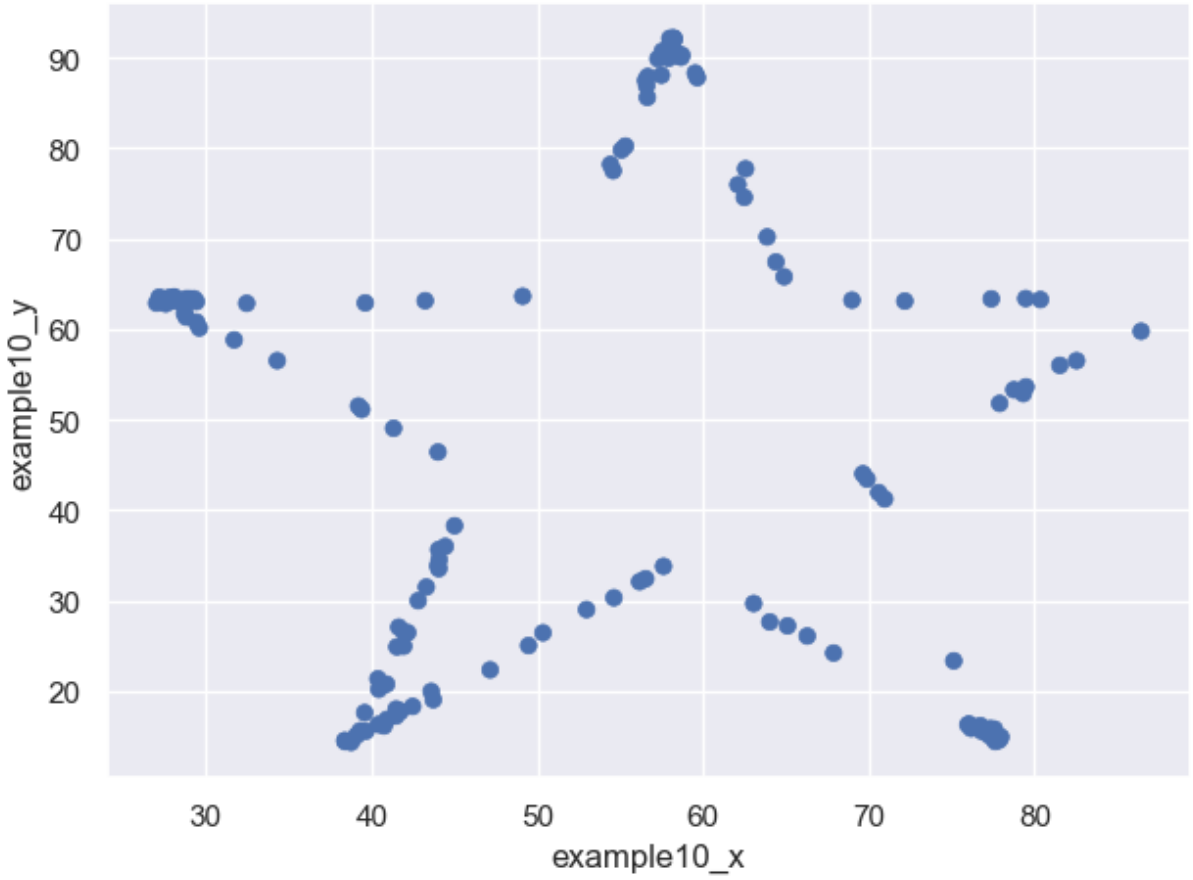

example5_y
example5_x
20
30
40
50
60
70
80
90
30
40
50
60
70
80

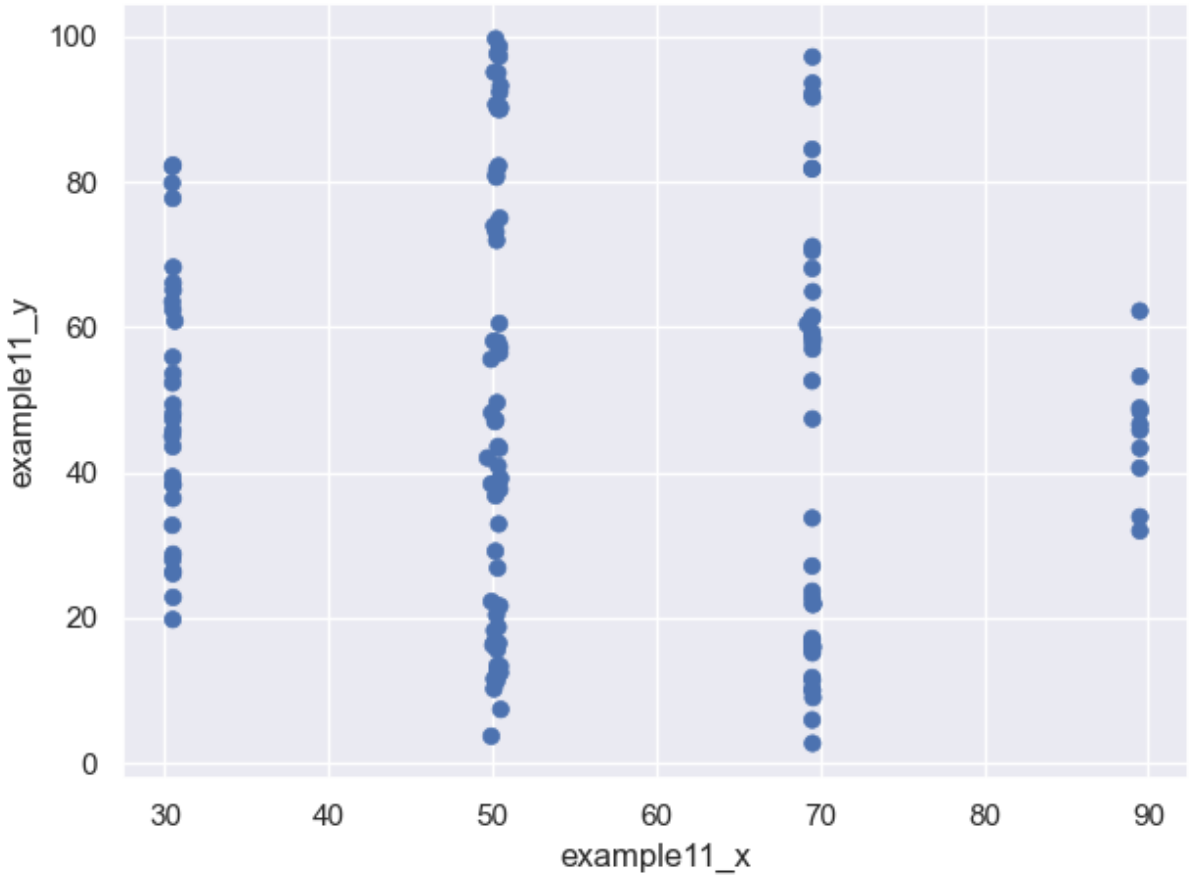

example6_y
example6_x
90
80
70
60
50
40
30
20
10
20
30
40
50
60
70
80
90
100

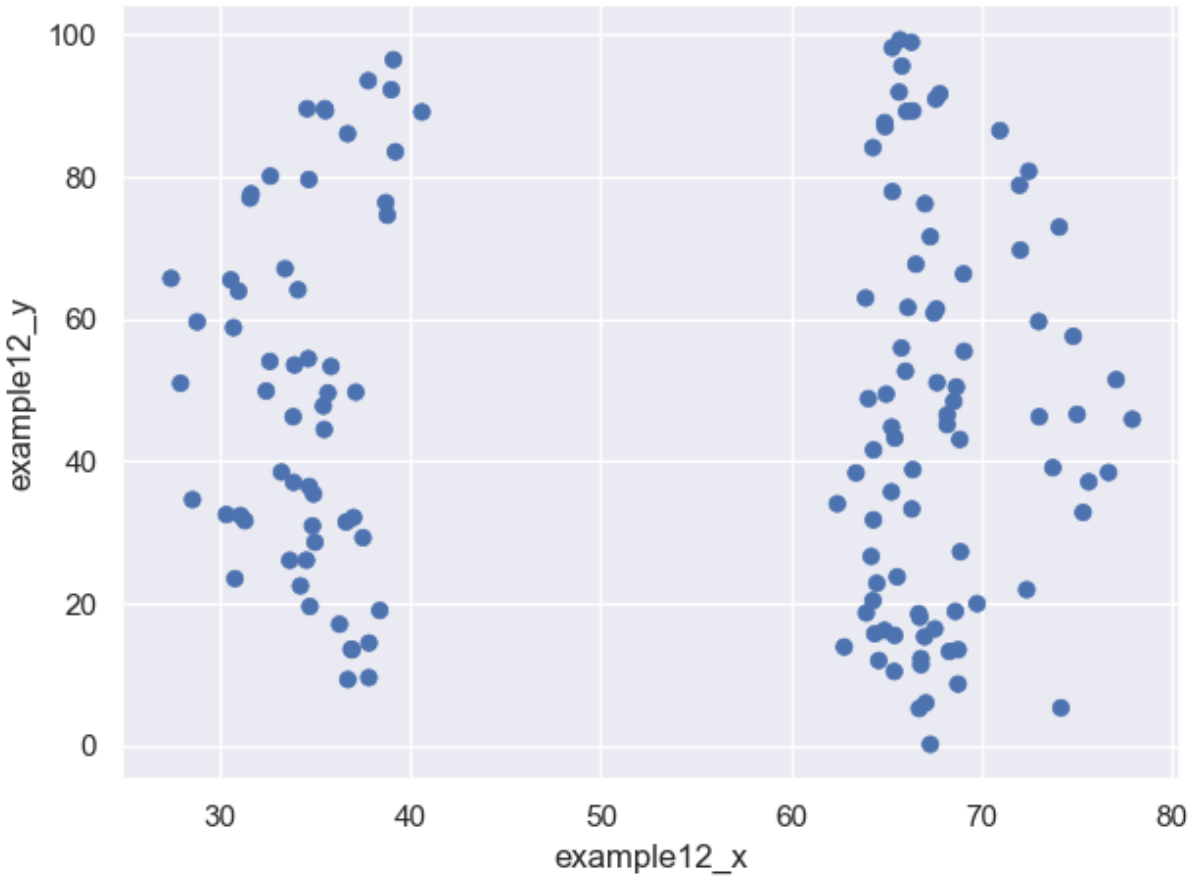

example7_y
example7_x
90
80
70
60
50
40
30
20
20
30
40
50
60
70
80
90

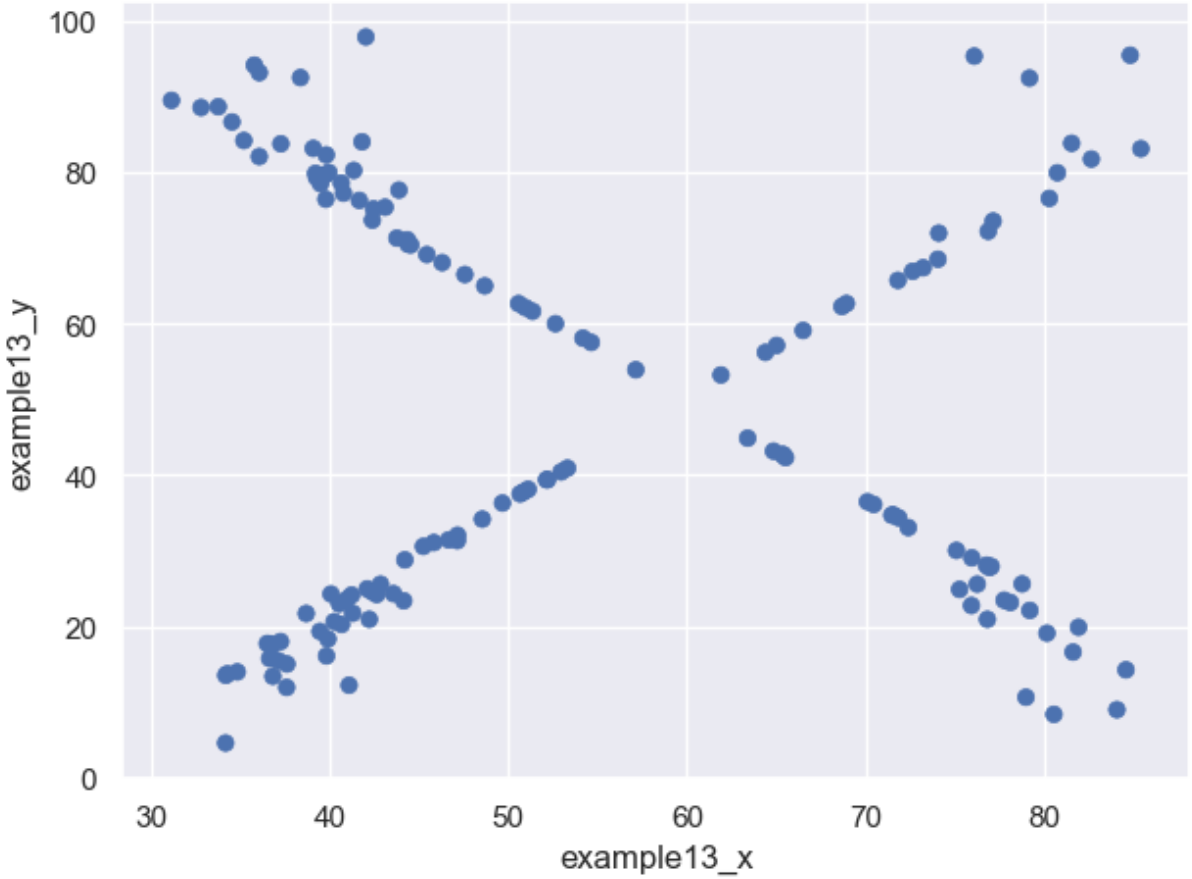

100
80
60
40
20
0
example8_y
20
30
40
50
60
70
80
90
example8_x


100
80
60
40
20
example9_y
20
30
40
50
60
70
80
90
example9_x


example10_y
example10_x
90
80
70
60
50
40
30
20
30
40
50
60
70
80


example11_y
example11_x
100
80
60
40
20
0
30
40
50
60
70
80
90


100
80
60
40
20
0
example12_y
30
40
50
60
70
80
example12_x


100
80
60
40
20
0
example13_y
30
40
50
60
70
80
example13_x

Exercise 5

Review you plots. How does your impression of how these datasets differ from what you wrote down in Exercise 3?

It is clear that my impression of these datasets drastically differ compared to what I wrote in exercise 3. In order to truly have a grasp of your data, you should both look at the descriptive statistics for said data, and the graph. The descriptive statistics can be incredibly misleading alone, as can the graphs. You need both together.

Economic Development and... Your Choice!

Exercise 6

Load the World Development Indicator data used in the plotting reading. Rather than picking a single year, pick a single country and look at how GDP per capita and one of the other variables in that dataset have evolved together over time.

Make any adjustments to the functional forms of your variables and/or axes needed to make the figure legible.

In [ ]:
```
wdi_data = (
    "https://raw.githubusercontent.com/nickeubank/"
    "practicaldatascience/master/Example_Data/wdi_plotting.csv"
)
world = pd.read_csv(wdi_data)
print("The following below is a sample of the data")
world.sample(5)
```

```
The following below is a sample of the data
```

Out[ ]:

	Year	Country Name	Country Code	GDP per capita (constant 2010 US$)	Population, total	CO2 emissions (metric tons per capita)	Mortality rate attributed to household and ambient air pollution, age-standardized (per 100,000 population)	PM2 poll popul exp to exce guic val of
5619	1996	Togo	TGO	556.747236	4348808.0	0.312729	NaN	
7694	2006	Jordan	JOR	3633.192887	5991547.0	3.366409	NaN	
5054	1994	Eswatini	SWZ	2862.963764	907622.0	1.145851	NaN	
345	1972	Micronesia, Fed. Sts.	FSM	NaN	62275.0	NaN	NaN	
3601	1987	Moldova	MDA	NaN	2918487.0	NaN	NaN	

In [ ]:
```
print(
    "First, we are going to write our code so that it specifically looks at the cou
)
iran = world[world["Country Name"] == "Iran, Islamic Rep."]
```

```
First, we are going to write our code so that it specifically looks at the country I
ran, before we graph our results
```

In [ ]:
```
print("The following below is a graph of Iran's GDP through the years")
so.Plot(iran, x="Year", y="GDP per capita (constant 2010 US$)").add(so.Line()).labe
    title="Iran's GDP Through 1970"
)
```

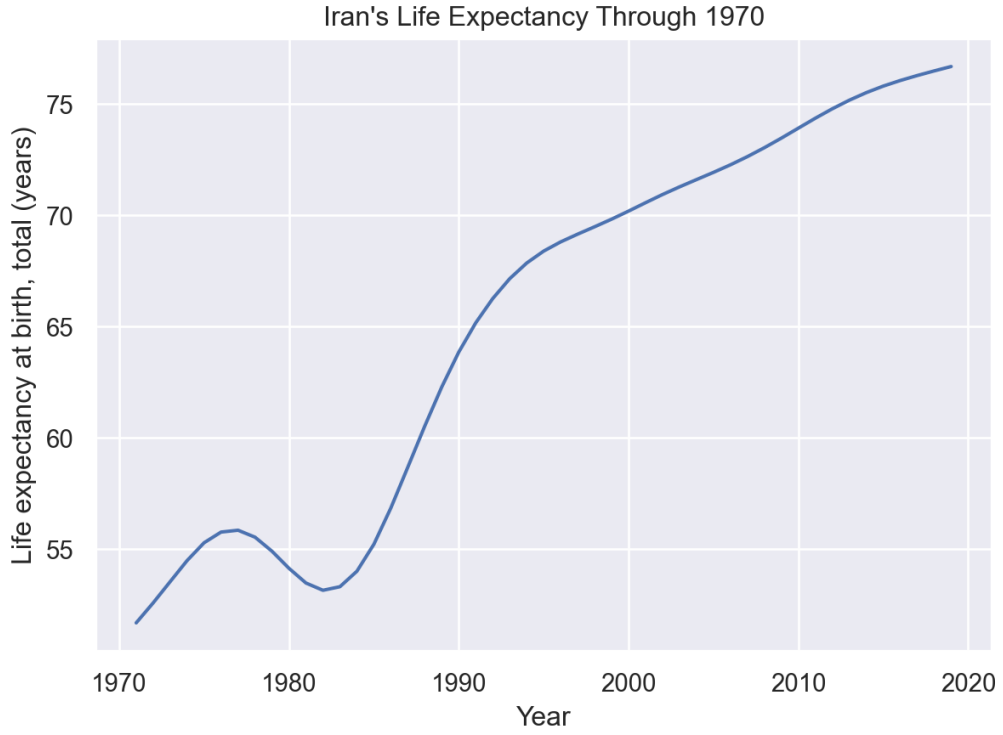
```
The following below is a graph of Iran's GDP through the years
```

Out[ ]:

In [ ]:
```
print("The following below is a graph of Iran's Life Expectancy through the years")
so.Plot(iran, x="Year", y="Life expectancy at birth, total (years)").add(
    so.Line()
).label(title="Iran's Life Expectancy Through 1970")
```

```
The following below is a graph of Iran's Life Expectancy through the years
```

Out[ ]:



Exercise 7

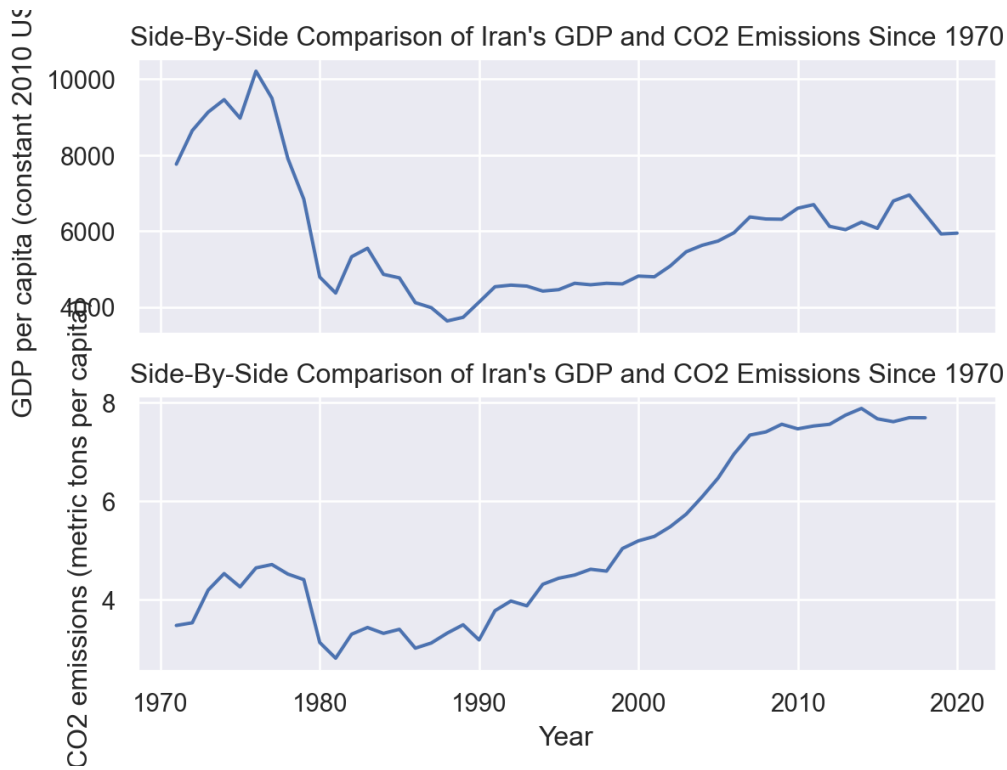
Now add a second series. Facet your plot so that the two subplots are positioned so that they are effectively sharing the same time axes (e.g., if you draw a line up from 2010 on one plot, you get to 2010 on the other).

Rather than telling you exactly how to do it, however, I'll point you to the seaborn tutorial. It has examples that don't do exactly what you want, but should be close enough you can guess-and-check to the solution you want!

Use your detective skills (and some guess and check work) to figure out how to get it to work!

In [ ]:
```
so.Plot(iran, x="Year").pair(
    y=[
        "GDP per capita (constant 2010 US$)",
        "CO2 emissions (metric tons per capita)",
    ]
).add(so.Line()).label(
    title="Side-By-Side Comparison of Iran's GDP and CO2 Emissions Since 1970"
)
```

Out[ ]:



In [ ]: