exercise_plotting_part1

October 8, 2025

1 Plotting Exercises, Part 1

1.0.1 Exercise 1

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

[18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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",
)
```

[19]:
```
df
```

[19]:
```
     example1_x  example1_y  example2_x  example2_y  example3_x  example3_y  \
0     32.331110   61.411101   51.203891   83.339777   55.993030   79.277264
1     53.421463   26.186880   58.974470   85.499818   50.032254   79.013071
2     63.920202   30.832194   51.872073   85.829738   51.288459   82.435940
3     70.289506   82.533649   48.179931   85.045117   51.170537   79.165294
4     34.118830   45.734551   41.683200   84.017941   44.377915   78.164628
..          …           …           …           …          …           …
137   59.851838   72.958391   50.967748   29.679774   39.921363   19.701850
138   48.960460   72.629526   91.191054   46.674343   84.794278   55.568650
139   46.844855   36.791714   55.863768   85.336487   55.662959   83.356480
140   39.963022   42.944915   49.280595   84.048823   50.492248   78.997532
141   66.704944   32.015095   43.368502   84.332177   51.467101   79.201845

     example4_x  example4_y  example5_x  example5_y  …  example9_x  \
0       55.3846     97.1795   51.147917   90.867412  …   47.695201
1       51.5385     96.0256   50.517126   89.102395  …   44.609976
2       46.1538     94.4872   50.207480   85.460047  …   43.856381
3       42.8205     91.4103   50.069482   83.057670  …   41.578929
```

```
4       40.7692     88.3333   50.562846   82.937822  …   49.177419
..          …          …         …          …   …        …
137     39.4872     25.3846   50.533635   17.019581  …   31.333244
138     91.2821     41.5385   77.500907   50.166986  …   86.401550
139     50.0000     95.7692   50.691124   87.513960  …   47.442112
140     47.9487     95.0000   49.990395   83.997357  …   46.264741
141     44.1026     92.6923   50.127182   82.990750  …   40.163816

     example9_y  example10_x  example10_y  example11_x  example11_y  \
0     95.241187    58.213608    91.881892    50.481508    93.222701
1     93.075835    58.196054    92.214989    50.282406    97.609984
2     94.085872    58.718231    90.310532    50.186703    99.694680
3     90.303567    57.278373    89.907607    50.326911    90.022053
4     96.610532    58.082020    92.008145    50.456207    89.987410
..          …            …            …            …           …
137   32.538569    43.722551    19.077328    30.487392    19.779470
138   38.746933    79.326078    52.900391    89.500180    31.978917
139   98.184302    56.663974    87.940125    50.410272    98.628369
140   94.116192    57.821789    90.693167    50.325924    94.994631
141   87.448672    58.243172    92.104328    50.104031    95.088538

     example12_x  example12_y  example13_x  example13_y
0      65.815540    95.588374    38.337757    92.472719
1      65.672265    91.933402    35.751871    94.116768
2      39.002716    92.261838    32.767218    88.518295
3      37.795303    93.532455    33.729607    88.622266
4      35.513901    89.599190    37.238249    83.724928
..           …            …            …           …
137    33.674442    26.090490    34.794594    13.969683
138    75.627255    37.128752    79.221764    22.094591
139    40.610125    89.136240    36.030880    93.121733
140    39.114366    96.481751    34.499558    86.609985
141    34.583829    89.588902    31.106867    89.461635

[142 rows x 26 columns]
```

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).

1.0.2 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. You might get values that are quite similar across datasets.)

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. Or, as Drew and Genevieve would say, WORK ONE CASE BY HAND! Once you have code that works for the first example dataset, then write the full loop around it.

[20]:
```python
for num in range(len(df.columns) // 2):
    mean_x = df[f"example{num + 1}_x"].mean()
    mean_y = df[f"example{num + 1}_y"].mean()
    std_x = df[f"example{num+1}_x"].std()
    std_y = df[f"example{num+1}_y"].std()
    corr = np.corrcoef(df[f"example{num + 1}_x"], df[f"example{num + 1}_y"])[0,␣
↪1]
    print(
        f"Example Dataset {num+1}:\n mean x: {mean_x}\n mean y: {mean_y}\n␣
↪standard deviation x: {std_x}\n standard deviation y: {std_y}\n Correlation:␣
↪{corr}"
    )
```

```
Example Dataset 1:
 mean x: 54.266099784295776
 mean y: 47.834720624943664
 standard deviation x: 16.769824954043756
 standard deviation y: 26.9397434192671
 Correlation: -0.0641283521673984
Example Dataset 2:
 mean x: 54.268730022394365
 mean y: 47.83082315530282
 standard deviation x: 16.769239493454403
 standard deviation y: 26.935726689918784
 Correlation: -0.06858639424107654
Example Dataset 3:
 mean x: 54.26731970598592
 mean y: 47.83771726725352
 standard deviation x: 16.76001265980608
 standard deviation y: 26.930036087838204
 Correlation: -0.06834335648025565
Example Dataset 4:
```

```
 mean x: 54.26327323943662
 mean y: 47.832252816901416
 standard deviation x: 16.76514203911679
 standard deviation y: 26.935403486939116
 Correlation: -0.06447185270095167
Example Dataset 5:
 mean x: 54.26030345169014
 mean y: 47.839829209014084
 standard deviation x: 16.767735488473807
 standard deviation y: 26.93019151853346
 Correlation: -0.06034144199921764
Example Dataset 6:
 mean x: 54.26144178316902
 mean y: 47.83025191366197
 standard deviation x: 16.765897903899337
 standard deviation y: 26.93987622043797
 Correlation: -0.06171483797263011
Example Dataset 7:
 mean x: 54.26880527950703
 mean y: 47.83545020401409
 standard deviation x: 16.766704015934764
 standard deviation y: 26.939997961411027
 Correlation: -0.06850422049412316
Example Dataset 8:
 mean x: 54.26784882366197
 mean y: 47.83589633112676
 standard deviation x: 16.76675894771805
 standard deviation y: 26.936104931679978
 Correlation: -0.068979735359512
Example Dataset 9:
 mean x: 54.26588178542254
 mean y: 47.831495652323945
 standard deviation x: 16.768852670828494
 standard deviation y: 26.93860807087184
 Correlation: -0.06860920641825635
Example Dataset 10:
 mean x: 54.26734110478873
 mean y: 47.83954522535211
 standard deviation x: 16.76895921619445
 standard deviation y: 26.93027468808843
 Correlation: -0.0629611002206542
Example Dataset 11:
 mean x: 54.26992723091549
 mean y: 47.836987988408445
 standard deviation x: 16.769958611325382
 standard deviation y: 26.937683806980512
 Correlation: -0.06944556959350369
Example Dataset 12:
```

```
 mean x: 54.266916301197185
 mean y: 47.83160198797184
 standard deviation x: 16.769999617573024
 standard deviation y: 26.937901927731797
 Correlation: -0.06657523020460904
Example Dataset 13:
 mean x: 54.26015033415493
 mean y: 47.839717279450696
 standard deviation x: 16.76995769550748
 standard deviation y: 26.93000168716234
 Correlation: -0.06558333729297582
```

1.0.3 Exercise 3

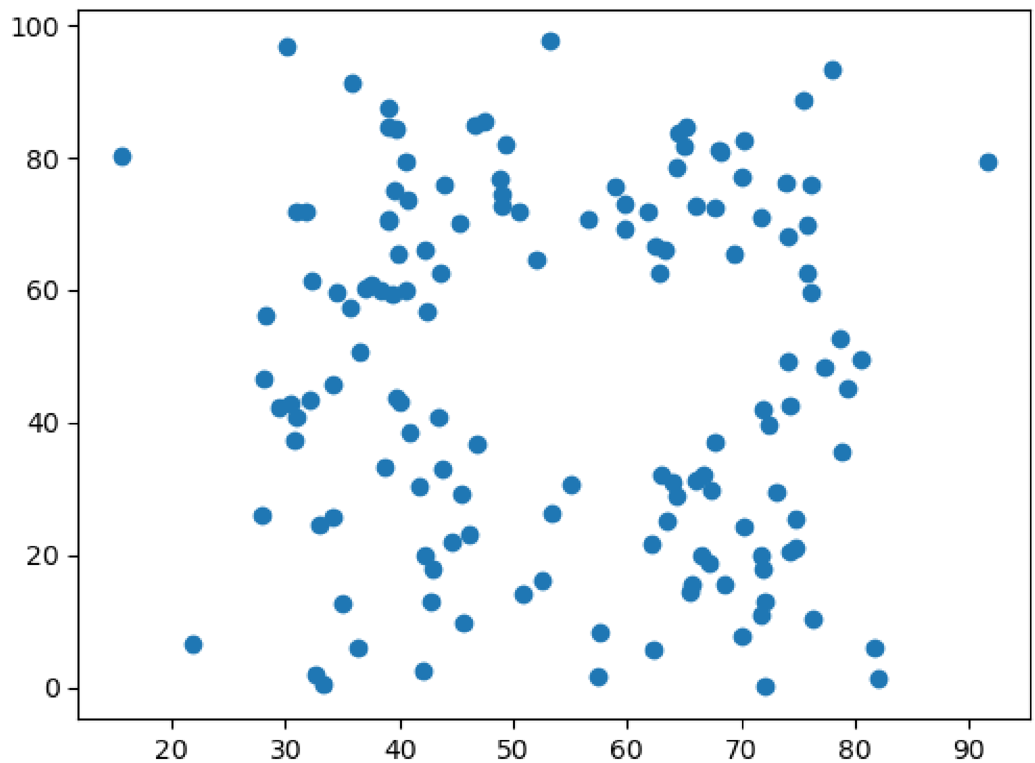
Based only on these results, discuss what might you conclude about these example datasets with your partner. Write down your thoughts.

Based on this data, the distribution of values between column x and y for each pair of columns is very similar, seeing as the mean and standard deviation values along with the correlation coefficient are approximately the same.

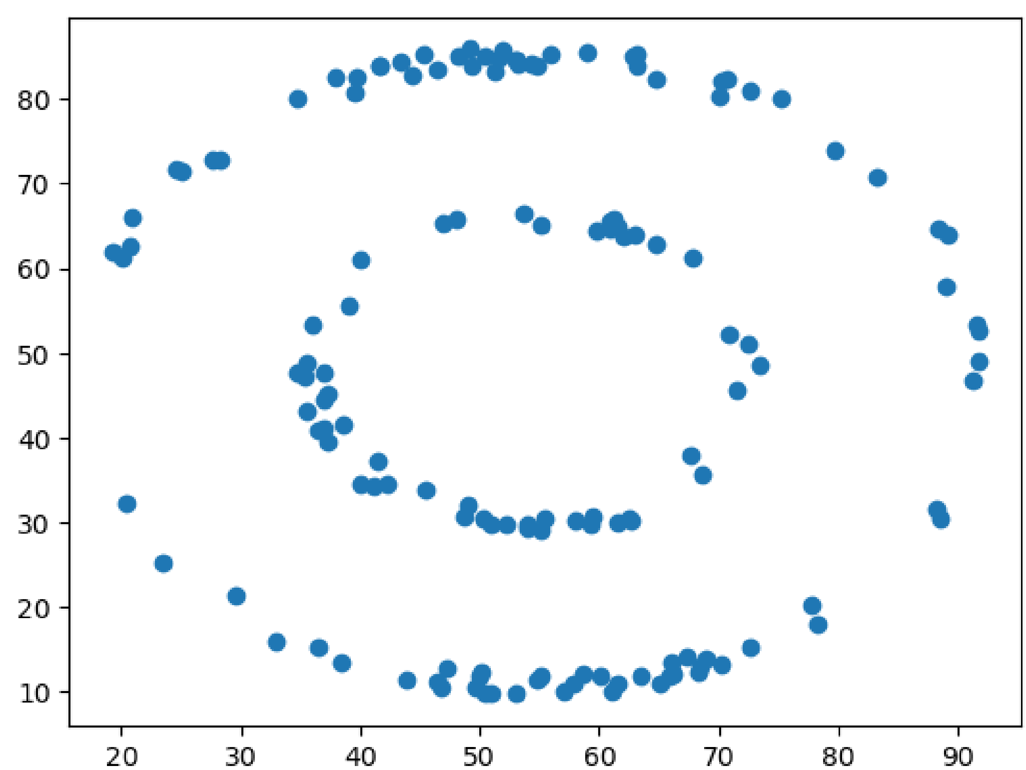
1.0.4 Execise 4

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

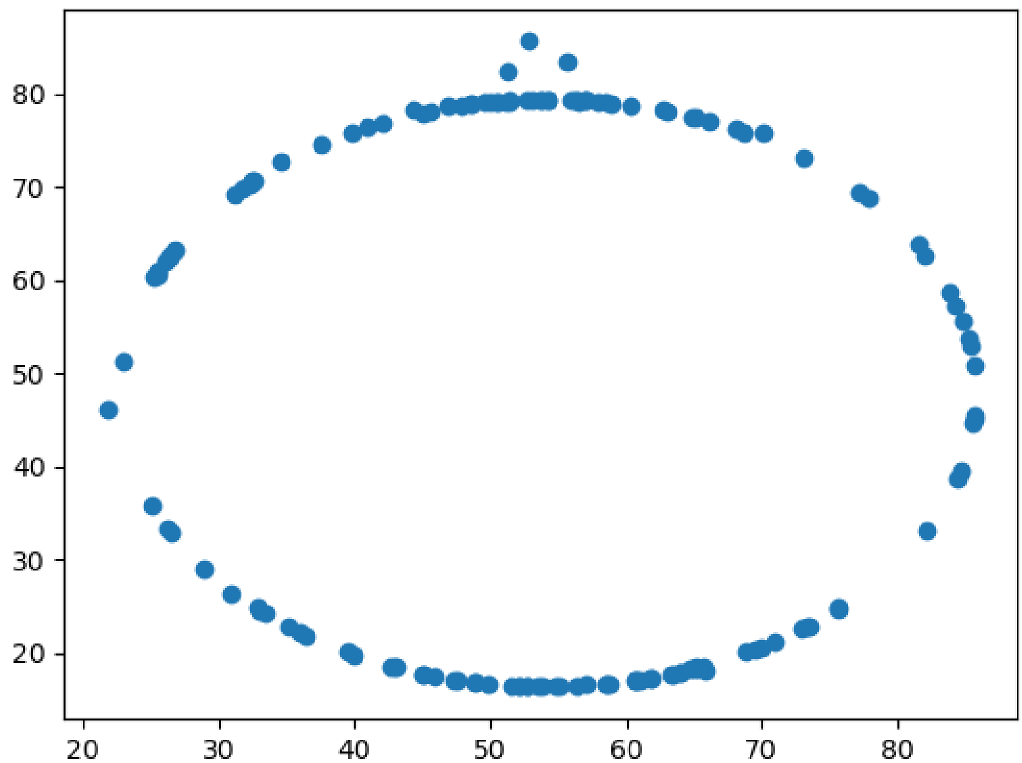
[21]:
```
for num in range(len(df.columns) // 2):
    x = df[f"example{num + 1}_x"]
    y = df[f"example{num + 1}_y"]
    fig, ax = plt.subplots()
    ax.scatter(x, y)
    plt.show()
```

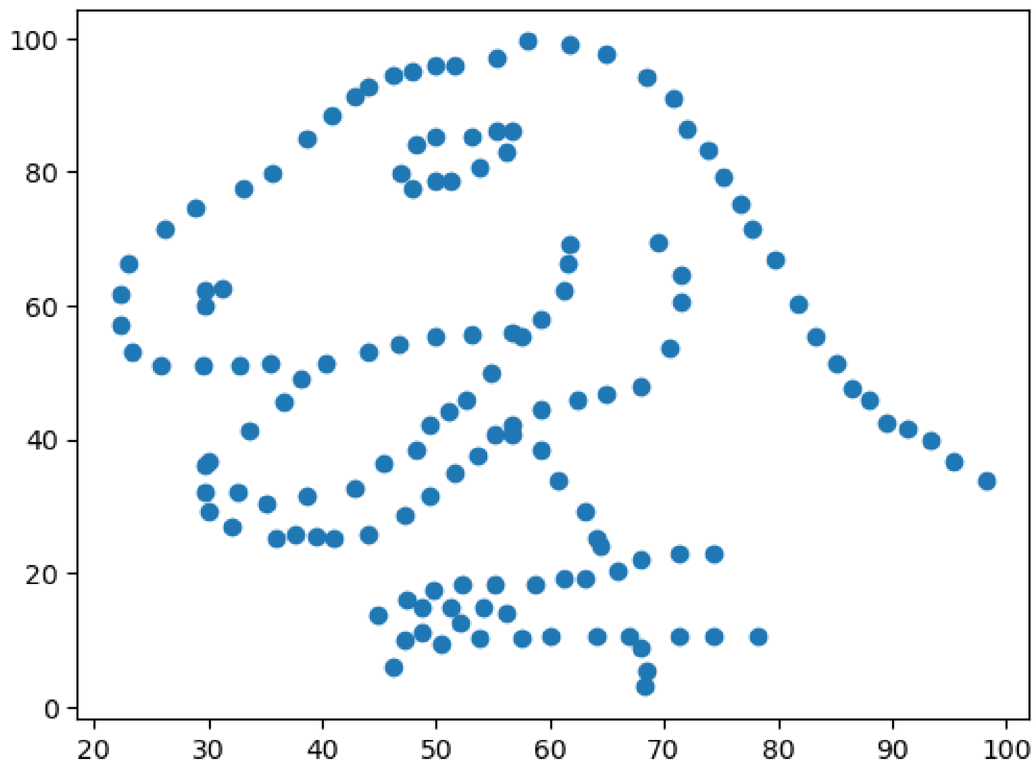


100
80
60
40
20
0
20
30
40
50
60
70
80
90

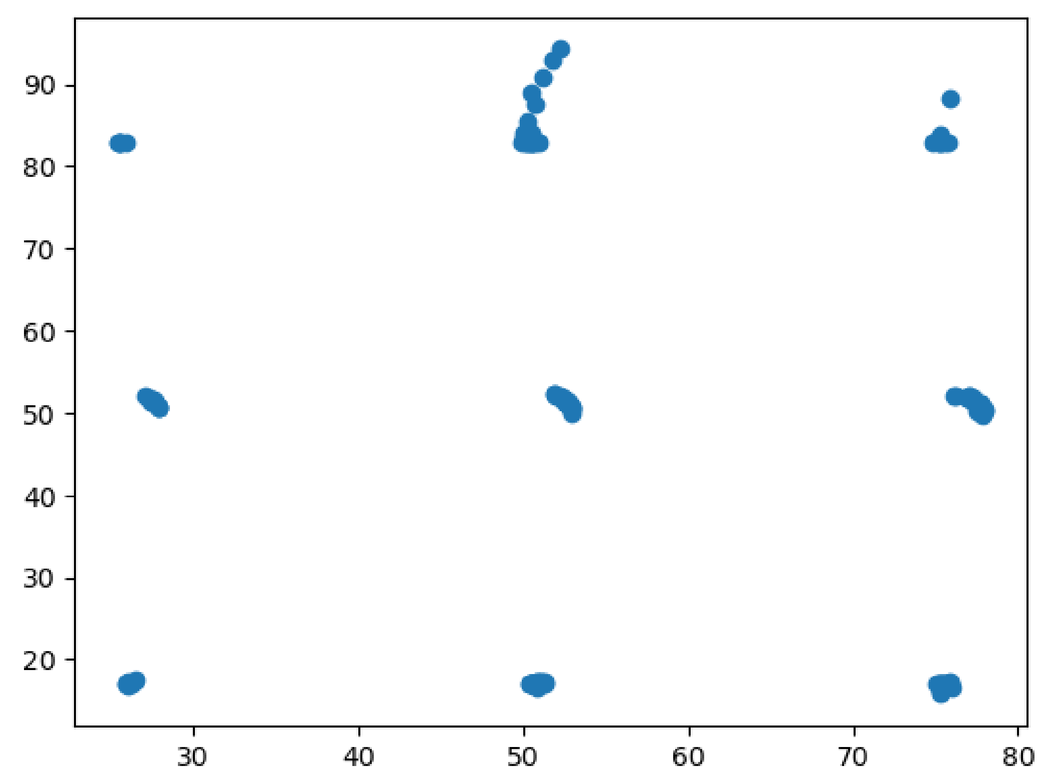


80
70
60
50
40
30
20
10
20
30
40
50
60
70
80
90

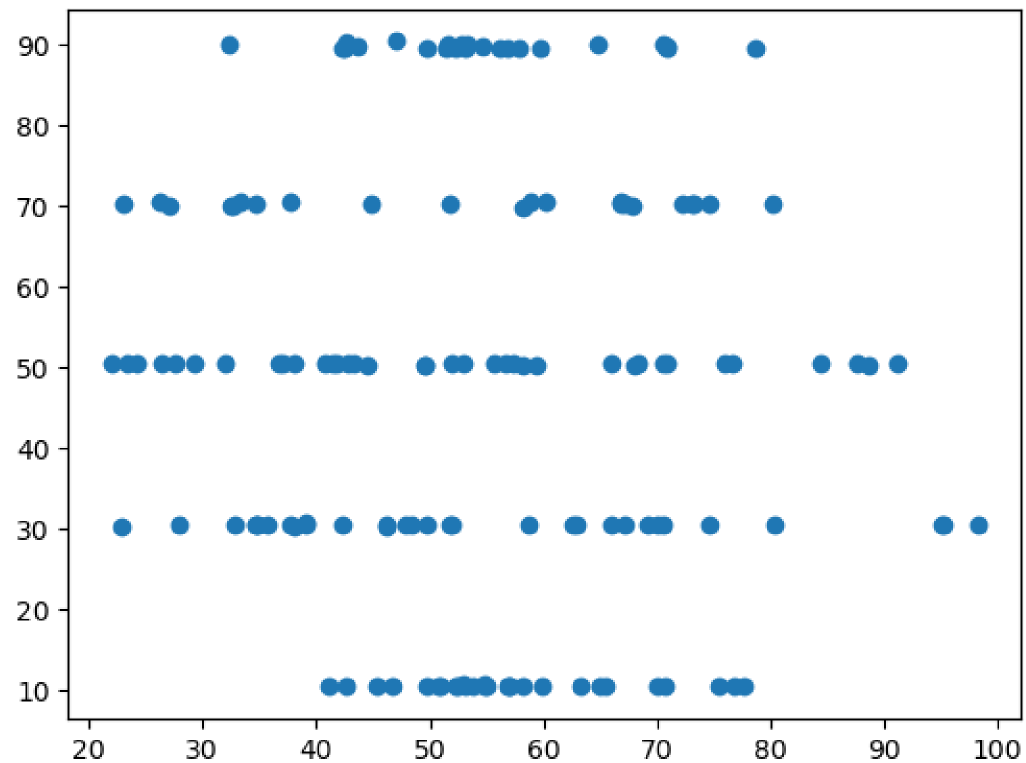

80
70
60
50
40
30
20
20
30
40
50
60
70
80

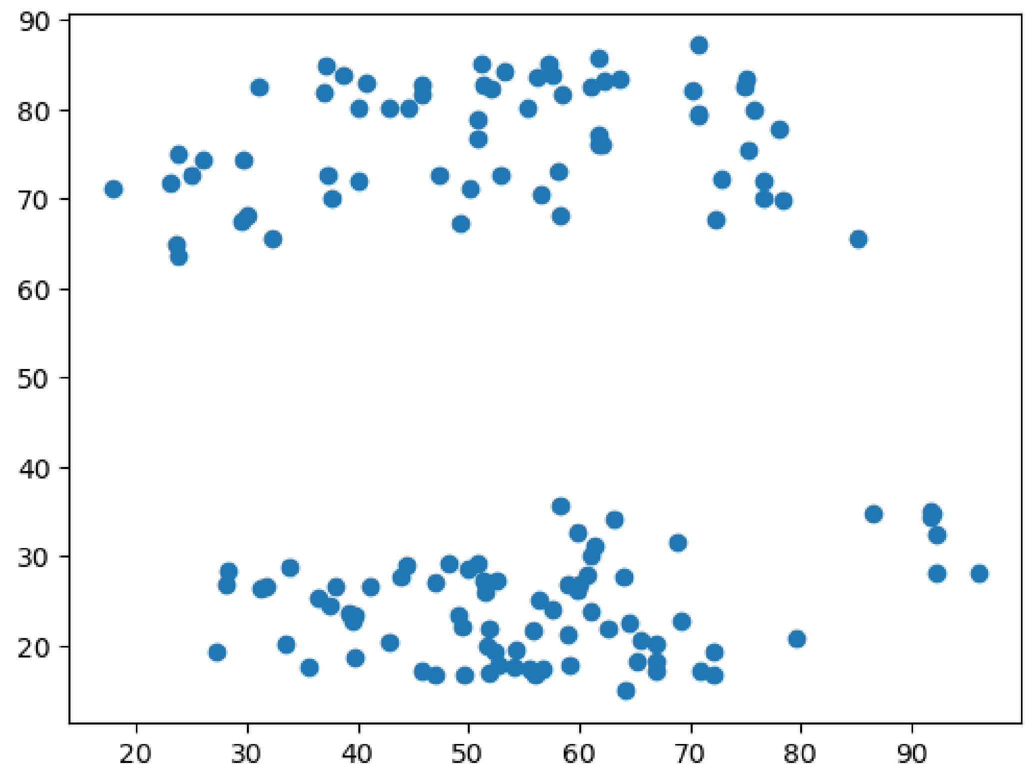


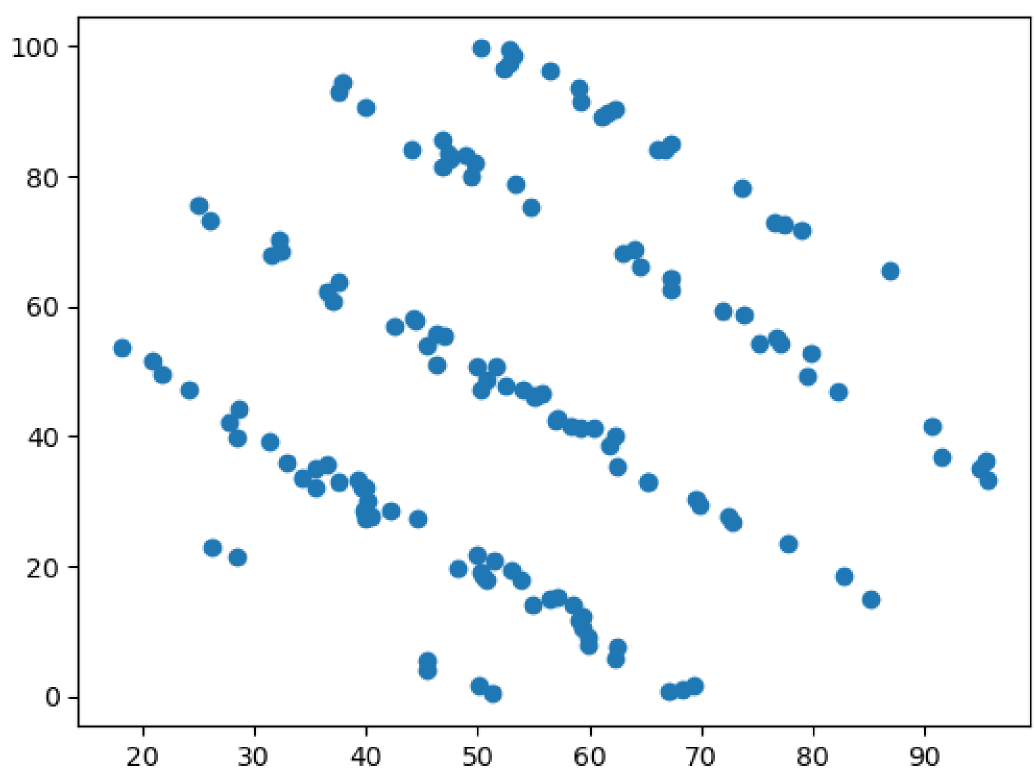
100
80
60
40
20
0
20
30
40
50
60
70
80
90
100

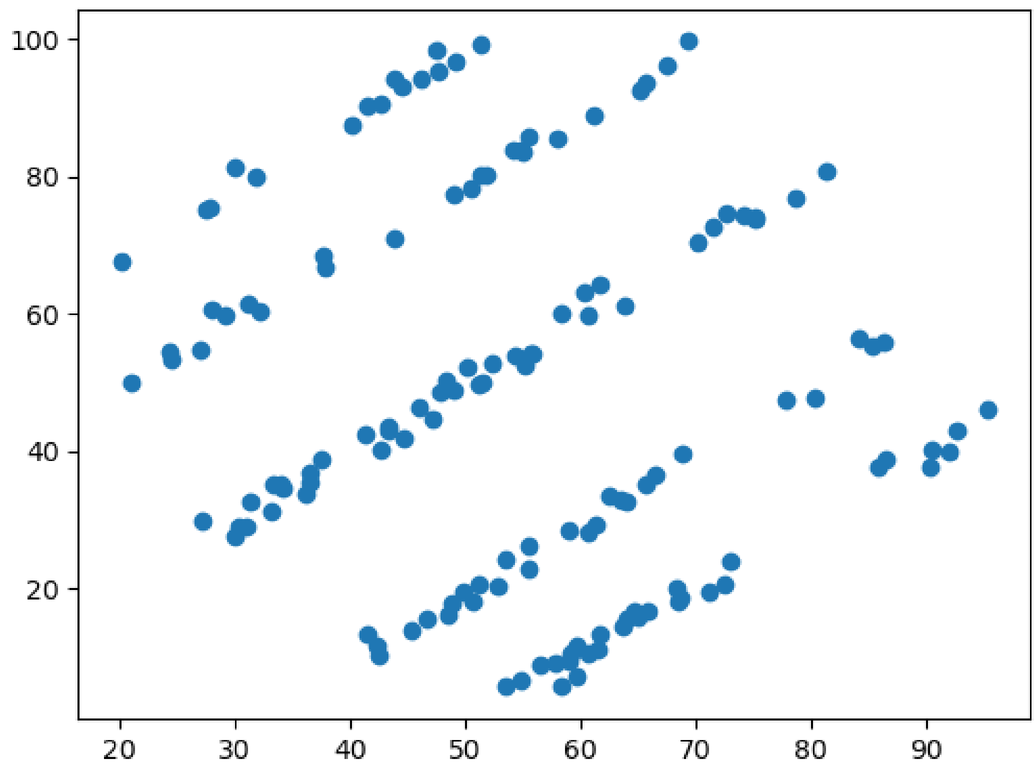


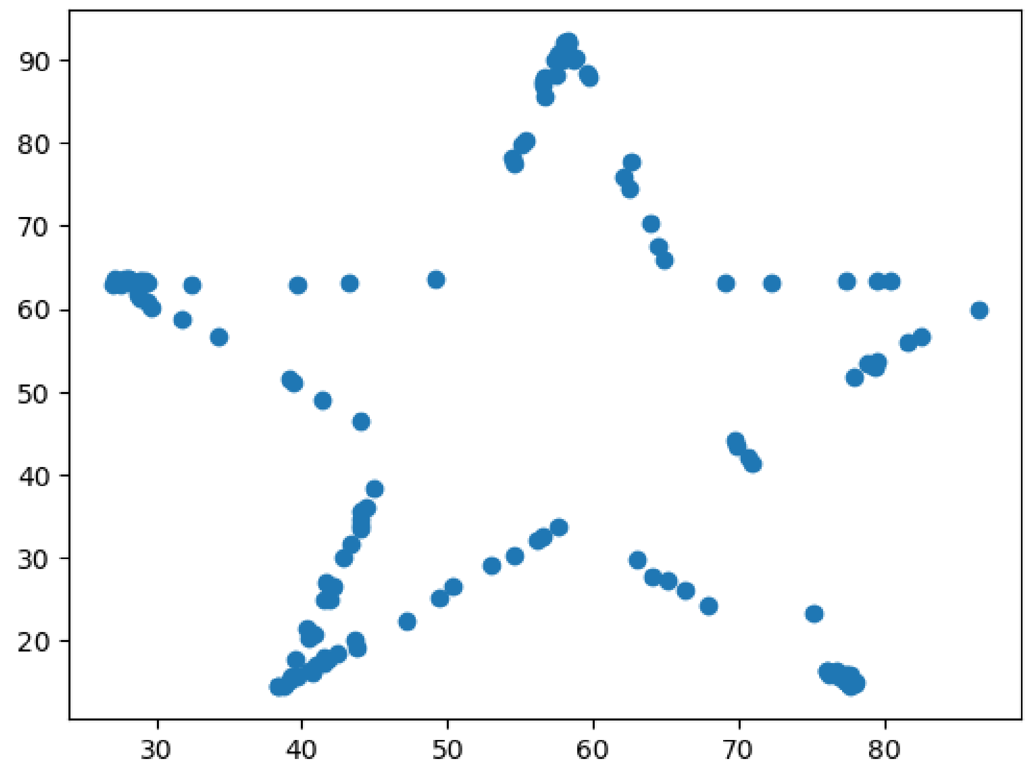
90
80
70
60
50
40
30
20
30
40
50
60
70
80

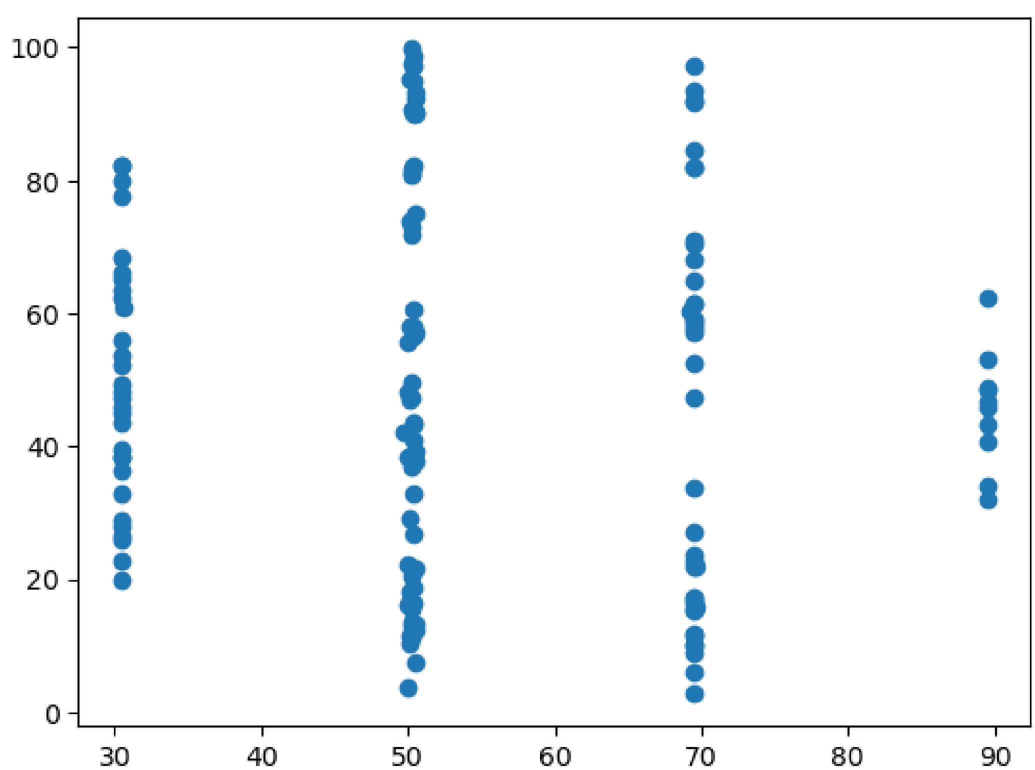

90
80
70
60
50
40
30
20
10
20
30
40
50
60
70
80
90
100

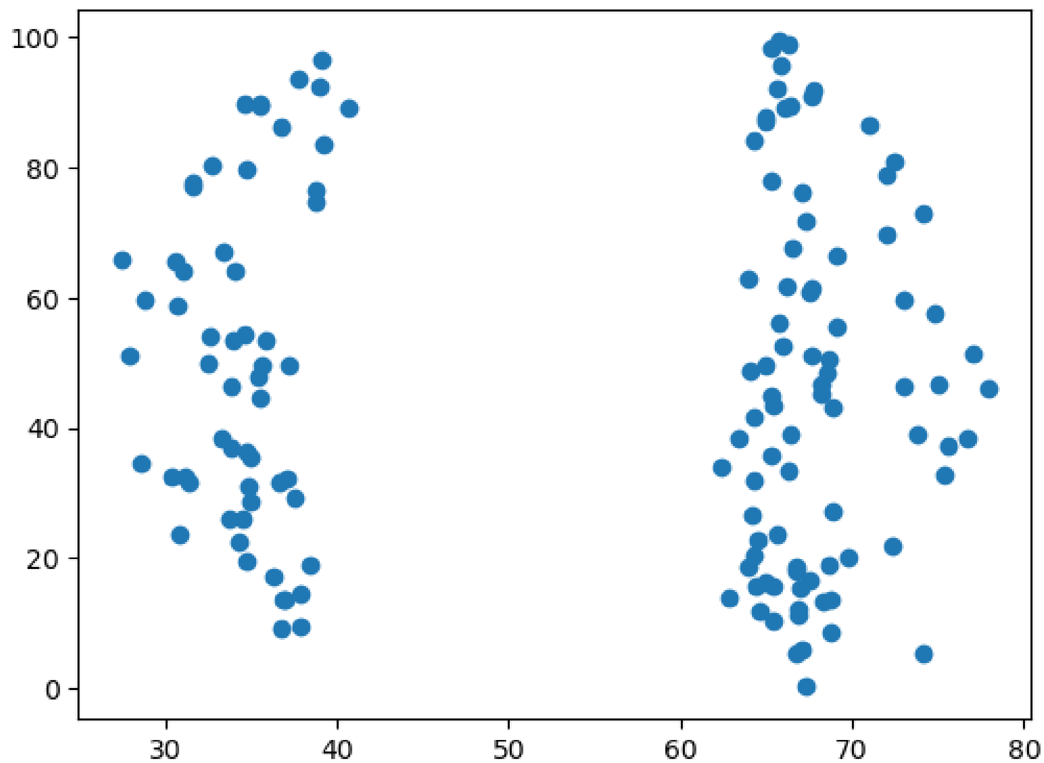

90
80
70
60
50
40
30
20
20
30
40
50
60
70
80
90


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

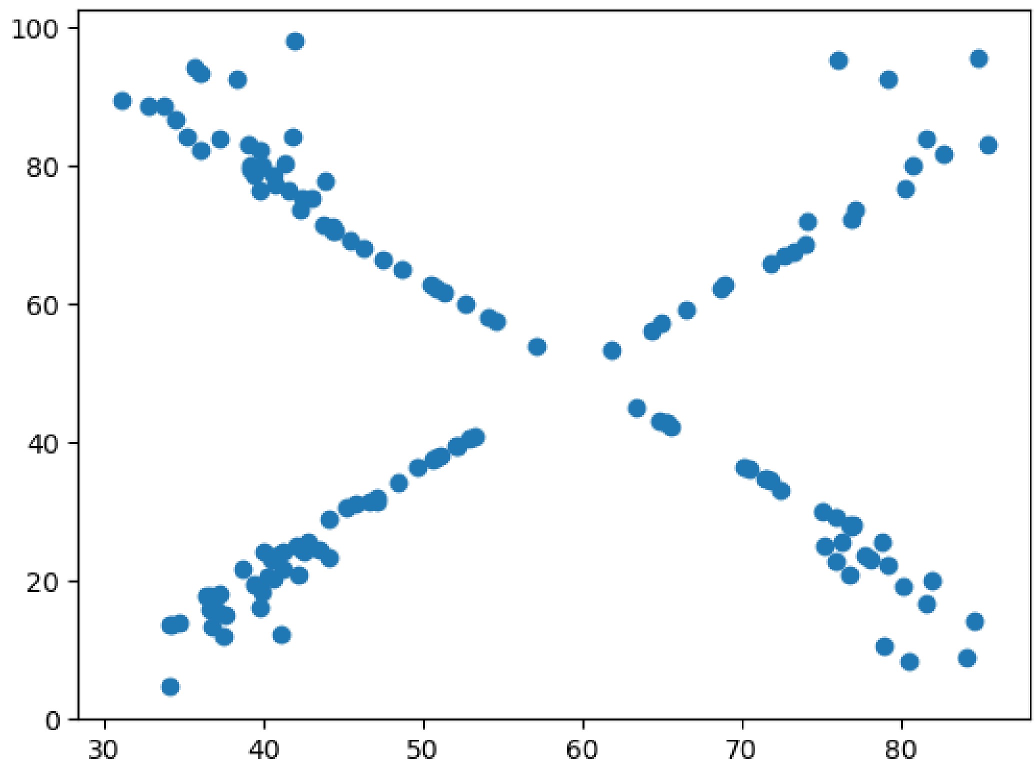

100
80
60
40
20
20
30
40
50
60
70
80
90


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


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



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



1.0.5 Exercise 5

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

We are thoroughly surprised by how differently scattered these points are given the summary statistics were approximately the same. We initially expected the graphs to look similar due to this. However, after giving it some thought, it seems reasonable that different distributions can give similar summary statistics since they just need to have a similar center and spread of data, even if the data points might be differently scattered.

1.1 Economic Development and... Your Choice!

1.1.1 Exercise 6

Load the World Development Indicator data here

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.

[25]:
```
world_df = pd.read_csv(
    "http://raw.githubusercontent.com/nickeubank/practicaldatascience/master/
↪Example_Data/wdi_plotting.csv"
)
preferred_country = "Argentina"
country_df = world_df[world_df["Country Name"] == preferred_country].copy()
```

[ ]:
```
country_df = country_df.sort_values("Year")
gdp_col = "GDP per capita (constant 2010 US$)"
life_col = "Life expectancy at birth, total (years)"
country_df = country_df[["Year", gdp_col, life_col]].dropna()
country_df["log_GDP"] = np.log10(
    country_df[gdp_col]
)  # log transform for easier understanding
```

1.1.2 Exercise 7

Now add a second series. Create a pair of plots 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).

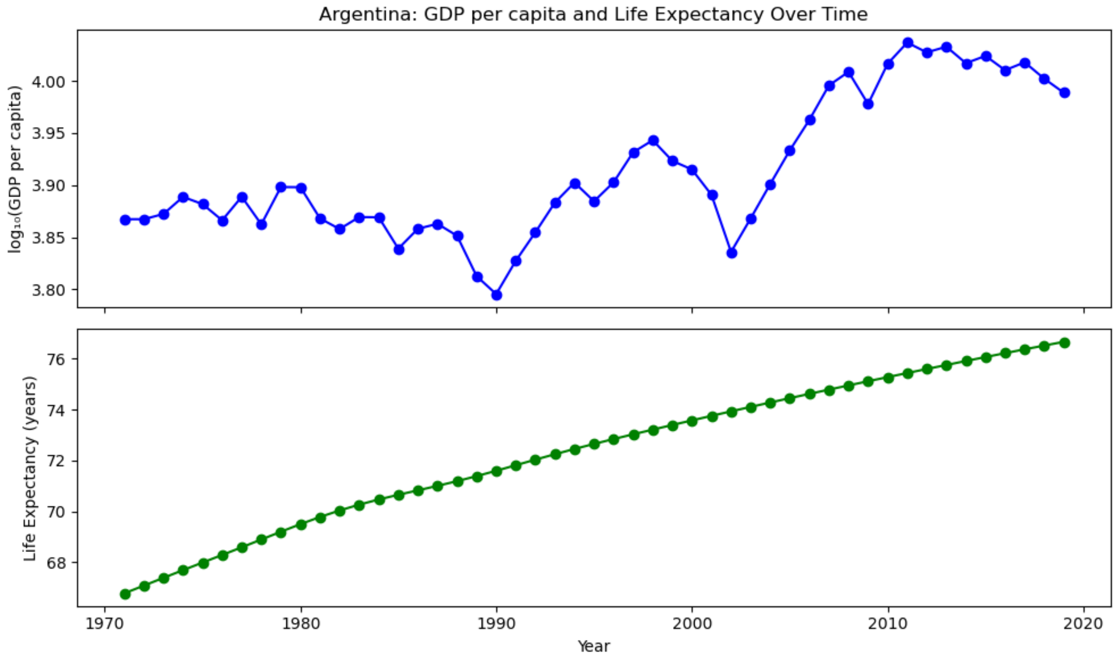
Use your detective skills (and some guess and check work) to figure out how to get it to work!

[ ]:
```
# gdp plot - top
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), sharex=True)
ax1.plot(country_df["Year"], country_df["log_GDP"], marker="o", color="blue")
ax1.set_ylabel("log (GDP per capita)")
ax1.set_title(f"{preferred_country}: GDP per capita and Life Expectancy Over␣
↪Time")

# life expectancy plot - bottom
ax2.plot(country_df["Year"], country_df[life_col], marker="o", color="green")
ax2.set_ylabel("Life Expectancy (years)")
ax2.set_xlabel("Year")

plt.tight_layout()
plt.show()

corr = np.corrcoef(country_df["log_GDP"], country_df[life_col])[0, 1]
print(f"Correlation between log(GDP per capita) and life expectancy: {corr:.
↪3f}")
```



`Correlation between log(GDP per capita) and life expectancy: 0.737`

- From 1970 to 2020, Argentina's GDP per capita goes up and down quite a bit instead of following a smooth upward path. There are clear dips in the late 1980s, early 2000s, and again around 2018–2020, which line up with known economic crises in the country. Even with these ups and downs, GDP per capita still trends slightly upward overall across the 50 years.
- On the other hand, life expectancy (the bottom graph) shows a steady and consistent increase from about 67 years in 1970 to over 76 years in 2020. This means that people in Argentina have been living longer over time, likely because of improvements in healthcare, living conditions, and technology, even when the economy was struggling.

-Overall, there seems to be a positive relationship between GDP per capita and life expectancy in the long run — as the economy grows, people tend to live longer. But the steady rise in life expectancy even when GDP falls suggests that factors other than income, like medical advancements and public health programs, also play a big role in improving overall well-being.