**#CODE#**

import matplotlib.pyplot as plt

"""

year=[1950, 1960, 1970, 1980, 1990, 2000, 2010, 2015]

populationInBil=[12,23,45,65,76,77,78,79]

plt.plot(year, populationInBil)

plt.show()

plt.scatter(year,populationInBil)

plt.show()

"""

"""

year=[1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100]

print(year)

print("printing length of list")

print(677 len(year))

print("printing last data point in list")

print(year[150])

print(year[-1])

population=[2.53, 2.57, 2.62, 2.67, 2.71, 2.76, 2.81, 2.86, 2.92, 2.97, 3.03, 3.08, 3.14, 3.2, 3.26, 3.33, 3.4, 3.47, 3.54, 3.62, 3.69, 3.77, 3.84, 3.92, 4.0, 4.07, 4.15, 4.22, 4.3, 4.37, 4.45, 4.53, 4.61, 4.69, 4.78, 4.86, 4.95, 5.05, 5.14, 5.23, 5.32, 5.41, 5.49, 5.58, 5.66, 5.74, 5.82, 5.9, 5.98, 6.05, 6.13, 6.2, 6.28, 6.36, 6.44, 6.51, 6.59, 6.67, 6.75, 6.83, 6.92, 7.0, 7.08, 7.16, 7.24, 7.32, 7.4, 7.48, 7.56, 7.64, 7.72, 7.79, 7.87, 7.94, 8.01, 8.08, 8.15, 8.22, 8.29, 8.36, 8.42, 8.49, 8.56, 8.62, 8.68, 8.74, 8.8, 8.86, 8.92, 8.98, 9.04, 9.09, 9.15, 9.2, 9.26, 9.31, 9.36, 9.41, 9.46, 9.5, 9.55, 9.6, 9.64, 9.68, 9.73, 9.77, 9.81, 9.85, 9.88, 9.92, 9.96, 9.99, 10.03, 10.06, 10.09, 10.13, 10.16, 10.19, 10.22, 10.25, 10.28, 10.31, 10.33, 10.36, 10.38, 10.41, 10.43, 10.46, 10.48, 10.5, 10.52, 10.55, 10.57, 10.59, 10.61, 10.63, 10.65, 10.66, 10.68, 10.7, 10.72, 10.73, 10.75, 10.77, 10.78, 10.79, 10.81, 10.82, 10.83, 10.84, 10.85]

print("printing len of population")

print(len(population))

print("printing last element of population", str(population[-1]))

plt.scatter(year,population)

plt.show()

"""

gdp\_cap=[974.58033839999996, 5937.0295259999984, 6223.3674650000003, 4797.2312670000001, 12779.379639999999, 34435.367439999995, 36126.492700000003, 29796.048340000001, 1391.253792, 33692.605080000001, 1441.2848730000001, 3822.137084, 7446.2988029999997, 12569.851769999999, 9065.8008250000003, 10680.792820000001, 1217.0329939999999, 430.07069159999998, 1713.7786860000001, 2042.0952400000001, 36319.235009999997, 706.01653699999997, 1704.0637240000001, 13171.638849999999, 4959.1148540000004, 7006.5804189999999, 986.14787920000003, 277.55185870000003, 3632.5577979999998, 9645.06142, 1544.7501119999999, 14619.222719999998, 8948.1029230000004, 22833.308509999999, 35278.418740000001, 2082.4815670000007, 6025.3747520000015, 6873.2623260000009, 5581.1809979999998, 5728.3535140000004, 12154.089749999999, 641.36952360000021, 690.80557590000001, 33207.0844, 30470.0167, 13206.48452, 752.74972649999995, 32170.37442, 1327.6089099999999, 27538.41188, 5186.0500030000003, 942.6542111, 579.23174299999982, 1201.637154, 3548.3308460000007, 39724.978669999997, 18008.944439999999, 36180.789190000003, 2452.210407, 3540.6515639999998, 11605.71449, 4471.0619059999999, 40675.996350000001, 25523.277099999999, 28569.719700000001, 7320.8802620000015, 31656.068060000001, 4519.4611709999999, 1463.249282, 1593.06548, 23348.139730000006, 47306.989780000004, 10461.05868, 1569.3314419999999, 414.5073415, 12057.49928, 1044.7701259999999, 759.34991009999999, 12451.6558, 1042.581557, 1803.151496, 10956.991120000001, 11977.57496, 3095.7722710000007, 9253.896111, 3820.1752299999998, 823.68562050000003, 944.0, 4811.0604290000001, 1091.359778, 36797.933319999996, 25185.009109999999, 2749.3209649999999, 619.67689239999982, 2013.9773049999999, 49357.190170000002, 22316.192869999999, 2605.94758, 9809.1856360000002, 4172.8384640000004, 7408.9055609999996, 3190.4810160000002, 15389.924680000002, 20509.64777, 19328.709009999999, 7670.122558, 10808.47561, 863.08846390000019, 1598.4350890000001, 21654.83194, 1712.4721360000001, 9786.5347139999994, 862.54075610000018, 47143.179640000002, 18678.314350000001, 25768.257590000001, 926.14106830000003, 9269.6578079999999, 28821.063699999999, 3970.0954069999998, 2602.3949950000001, 4513.4806429999999, 33859.748350000002, 37506.419070000004, 4184.5480889999999, 28718.276839999999, 1107.482182, 7458.3963269999977, 882.9699437999999, 18008.509239999999, 7092.9230250000001, 8458.2763840000007, 1056.3801209999999, 33203.261279999999, 42951.65309, 10611.46299, 11415.805689999999, 2441.5764039999999, 3025.3497980000002, 2280.769906, 1271.211593, 469.70929810000007]

life\_exp=[43.828000000000003, 76.423000000000002, 72.301000000000002, 42.731000000000002, 75.319999999999993, 81.234999999999999, 79.828999999999994, 75.635000000000005, 64.061999999999998, 79.441000000000003, 56.728000000000002, 65.554000000000002, 74.852000000000004, 50.728000000000002, 72.390000000000001, 73.004999999999995, 52.295000000000002, 49.579999999999998, 59.722999999999999, 50.43, 80.653000000000006, 44.741000000000007, 50.651000000000003, 78.552999999999997, 72.960999999999999, 72.888999999999996, 65.152000000000001, 46.462000000000003, 55.322000000000003, 78.781999999999996, 48.328000000000003, 75.748000000000005, 78.272999999999996, 76.486000000000004, 78.331999999999994, 54.790999999999997, 72.234999999999999, 74.994, 71.338000000000022, 71.878, 51.578999999999994, 58.039999999999999, 52.947000000000003, 79.313000000000002, 80.656999999999996, 56.734999999999999, 59.448, 79.406000000000006, 60.021999999999998, 79.483000000000004, 70.259, 56.006999999999998, 46.388000000000012, 60.915999999999997, 70.198000000000008, 82.207999999999998, 73.338000000000022, 81.757000000000005, 64.698000000000008, 70.650000000000006, 70.963999999999999, 59.545000000000002, 78.885000000000005, 80.745000000000005, 80.546000000000006, 72.566999999999993, 82.602999999999994, 72.534999999999997, 54.109999999999999, 67.296999999999997, 78.623000000000005, 77.588000000000022, 71.992999999999995, 42.591999999999999, 45.677999999999997, 73.951999999999998, 59.443000000000012, 48.302999999999997, 74.241, 54.466999999999999, 64.164000000000001, 72.801000000000002, 76.194999999999993, 66.802999999999997, 74.543000000000006, 71.164000000000001, 42.082000000000001, 62.069000000000003, 52.906000000000013, 63.784999999999997, 79.762, 80.203999999999994, 72.899000000000001, 56.866999999999997, 46.859000000000002, 80.195999999999998, 75.640000000000001, 65.483000000000004, 75.536999999999978, 71.751999999999995, 71.421000000000006, 71.688000000000002, 75.563000000000002, 78.097999999999999, 78.746000000000024, 76.441999999999993, 72.475999999999999, 46.241999999999997, 65.528000000000006, 72.777000000000001, 63.061999999999998, 74.001999999999995, 42.568000000000012, 79.971999999999994, 74.662999999999997, 77.926000000000002, 48.158999999999999, 49.338999999999999, 80.941000000000003, 72.396000000000001, 58.555999999999997, 39.613, 80.884, 81.701000000000022, 74.143000000000001, 78.400000000000006, 52.517000000000003, 70.616, 58.420000000000002, 69.819000000000003, 73.923000000000002, 71.777000000000001, 51.542000000000002, 79.424999999999997, 78.242000000000004, 76.384, 73.747, 74.248999999999995, 73.421999999999997, 62.698, 42.383999999999993, 43.487000000000002]

#priniting length of these lists

print("len(gdp\_cap)", str(len(gdp\_cap)))

print("len(life\_exp)", str(len(life\_exp)))

# Make a line plot, gdp\_cap on the x-axis, life\_exp on the y-axis

plt.plot(gdp\_cap, life\_exp)

# Display the plot

plt.show()

#LINE GRAPH IS MESSY ..MAY NOT BE THAT READABLE OR MEANIGFUL FOR ANALAYSIS

# BETTER OPTION IS TO USE SCATTER PLOT..

#IF SCATTER PLOT IS ALSO DENSE AND NOT READABLE

#BEST OPTION IS USE LOG SCALE FOR EITHER X AXIS OR Y AXIS PARAMETER

# Make a line plot, gdp\_cap on the x-axis, life\_exp on the y-axis

plt.scatter(gdp\_cap, life\_exp)

# Display the plot

#plt.show()

#cainging x scale ot log

plt.xscale('log')

plt.show()

"""

"""

# Import package

import matplotlib.pyplot as plt

# Build Scatter plot

#pop on x and life\_exp on y axis

#print(len(population))

#plt.scatter(population,life\_exp)

# Show plot

#plt.show()

# this is giving an error of mismatch in len of two lists being plotted against each other

"""

"""

# get another lists contents

population=[31.889923, 3.6005229999999999, 33.333216, 12.420476000000001, 40.301926999999999, 20.434176000000001, 8.199783, 0.70857300000000001, 150.448339, 10.392226000000001, 8.0783140000000007, 9.1191519999999997, 4.5521979999999997, 1.6391309999999999, 190.01064700000001, 7.3228580000000001, 14.326203, 8.3905049999999992, 14.131857999999999, 17.696293000000001, 33.390141, 4.3690379999999998, 10.238807, 16.284741, 1318.683096, 44.227550000000001, 0.71096000000000004, 64.606758999999997, 3.8006099999999998, 4.1338840000000001, 18.013408999999999, 4.4933120000000004, 11.416987000000001, 10.228744000000001, 5.4681199999999999, 0.49637399999999998, 9.3196220000000007, 13.75568, 80.264543000000003, 6.9396880000000003, 0.55120100000000005, 4.9065849999999998, 76.511887000000002, 5.2384599999999999, 61.083916000000002, 1.4548669999999999, 1.6883589999999999, 82.400996000000006, 22.873338, 10.706289999999999, 12.572927999999999, 9.9478139999999993, 1.4720409999999999, 8.5028140000000008, 7.4837629999999997, 6.9804120000000003, 9.9561080000000004, 0.301931, 1110.3963309999999, 223.547, 69.453569999999999, 27.499638000000001, 4.1090859999999996, 6.426679, 58.147733000000002, 2.780132, 127.467972, 6.0531930000000003, 35.610177, 23.301725000000001, 49.044789999999999, 2.5055589999999999, 3.921278, 2.0126490000000001, 3.1939419999999998, 6.0369140000000003, 19.167653999999999, 13.327078999999999, 24.821286000000001, 12.031795000000001, 3.2700650000000002, 1.250882, 108.700891, 2.8741270000000001, 0.68473600000000001, 33.757174999999997, 19.951656, 47.761980000000001, 2.0550799999999998, 28.901789999999998, 16.570613000000002, 4.1157709999999996, 5.6753559999999998, 12.894864999999999, 135.03116399999999, 4.6279260000000004, 3.2048969999999999, 169.27061699999999, 3.2421730000000002, 6.6671469999999999, 28.674757, 91.077286999999998, 38.518241000000003, 10.642836000000001, 3.942491, 0.79809399999999997, 22.276056000000001, 8.8605879999999999, 0.19957900000000001, 27.601037999999999, 12.267493, 10.150264999999999, 6.1445619999999996, 4.5530090000000003, 5.4475020000000001, 2.0092449999999999, 9.1187729999999991, 43.997827999999998, 40.448191000000001, 20.378239000000001, 42.292929000000001, 1.1330659999999999, 9.0310880000000004, 7.5546610000000003, 19.314747000000001, 23.174294, 38.13964, 65.068149000000005, 5.7015789999999997, 1.056608, 10.276158000000001, 71.158647000000002, 29.170397999999999, 60.776237999999999, 301.13994700000001, 3.4474960000000001, 26.084662000000002, 85.262355999999997, 4.018332, 22.211742999999998, 11.746034999999999, 12.311143]

print("prinitng len of population", str(len(population)))

life\_exp=[43.828000000000003, 76.423000000000002, 72.301000000000002, 42.731000000000002, 75.319999999999993, 81.234999999999999, 79.828999999999994, 75.635000000000005, 64.061999999999998, 79.441000000000003, 56.728000000000002, 65.554000000000002, 74.852000000000004, 50.728000000000002, 72.390000000000001, 73.004999999999995, 52.295000000000002, 49.579999999999998, 59.722999999999999, 50.43, 80.653000000000006, 44.741000000000007, 50.651000000000003, 78.552999999999997, 72.960999999999999, 72.888999999999996, 65.152000000000001, 46.462000000000003, 55.322000000000003, 78.781999999999996, 48.328000000000003, 75.748000000000005, 78.272999999999996, 76.486000000000004, 78.331999999999994, 54.790999999999997, 72.234999999999999, 74.994, 71.338000000000022, 71.878, 51.578999999999994, 58.039999999999999, 52.947000000000003, 79.313000000000002, 80.656999999999996, 56.734999999999999, 59.448, 79.406000000000006, 60.021999999999998, 79.483000000000004, 70.259, 56.006999999999998, 46.388000000000012, 60.915999999999997, 70.198000000000008, 82.207999999999998, 73.338000000000022, 81.757000000000005, 64.698000000000008, 70.650000000000006, 70.963999999999999, 59.545000000000002, 78.885000000000005, 80.745000000000005, 80.546000000000006, 72.566999999999993, 82.602999999999994, 72.534999999999997, 54.109999999999999, 67.296999999999997, 78.623000000000005, 77.588000000000022, 71.992999999999995, 42.591999999999999, 45.677999999999997, 73.951999999999998, 59.443000000000012, 48.302999999999997, 74.241, 54.466999999999999, 64.164000000000001, 72.801000000000002, 76.194999999999993, 66.802999999999997, 74.543000000000006, 71.164000000000001, 42.082000000000001, 62.069000000000003, 52.906000000000013, 63.784999999999997, 79.762, 80.203999999999994, 72.899000000000001, 56.866999999999997, 46.859000000000002, 80.195999999999998, 75.640000000000001, 65.483000000000004, 75.536999999999978, 71.751999999999995, 71.421000000000006, 71.688000000000002, 75.563000000000002, 78.097999999999999, 78.746000000000024, 76.441999999999993, 72.475999999999999, 46.241999999999997, 65.528000000000006, 72.777000000000001, 63.061999999999998, 74.001999999999995, 42.568000000000012, 79.971999999999994, 74.662999999999997, 77.926000000000002, 48.158999999999999, 49.338999999999999, 80.941000000000003, 72.396000000000001, 58.555999999999997, 39.613, 80.884, 81.701000000000022, 74.143000000000001, 78.400000000000006, 52.517000000000003, 70.616, 58.420000000000002, 69.819000000000003, 73.923000000000002, 71.777000000000001, 51.542000000000002, 79.424999999999997, 78.242000000000004, 76.384, 73.747, 74.248999999999995, 73.421999999999997, 62.698, 42.383999999999993, 43.487000000000002]

print("prinitng len of LIFE\_EXPENTANCY", str(len(life\_exp)))

#now plot scatter plot between population and life expentancy : to check is there anty correlation

# Build Scatter plot

#pop on x and life\_exp on y axis

plt.scatter(population,life\_exp)

# Show plot

plt.show()

# as increase in one parameter results in no increas or ddescre in other parameter .. these two parameters are not correlated

#plt.xscale('log')

#plt.show()

#EHISTOGRAMS ARE USEFULL TO GET IDEA ABOUT DATA DISTRIBUTIONS

import matplotlib.pyplot as plt

plt.hist(life\_exp)

# Display histogram

plt.show()

######################

# Build histogram with 5 bins

import matplotlib.pyplot as plt

# Build histogram with 5 bins

import matplotlib.pyplot as plt

plt.hist(life\_exp,bins=5)

# Show and clean up plot

plt.show()

plt.clf()

# Build histogram with 20 bins

plt.hist(life\_exp,bins=20)

help(plt.hist)

# Show and clean up again

plt.show()

plt.clf()

##############

"""

"""

"""

In the previous exercise, you didn't specify the number of bins. By default, Python sets the number

of bins to 10 in that case. The number of bins is pretty important. Too few bins will oversimplify

reality and won't show you the details. Too many bins will overcomplicate reality and won't show the bigger picture.

To control the number of bins to divide your data in, you can set the bins argument.

That's exactly what you'll do in this exercise. You'll be making two plots here. The code

in the script already includes plt.show() and plt.clf() calls; plt.show() displays a plot;

plt.clf() cleans it up again so you can start afresh.

As before, life\_exp is available and matplotlib.pyplot is imported as plt.

"""

#########################

# Histogram of life\_exp, 15 bins

plt.hist(life\_exp, bins=15)

print("just life\_exp 15 n=bins")

# Show and clear plot

plt.show()

plt.clf()

life\_exp1950=[28.8, 55.23, 43.08, 30.02, 62.48, 69.12, 66.8, 50.94, 37.48, 68.0, 38.22, 40.41, 53.82, 47.62, 50.92, 59.6, 31.98, 39.03, 39.42, 38.52, 68.75, 35.46, 38.09, 54.74, 44.0, 50.64, 40.72, 39.14, 42.11, 57.21, 40.48, 61.21, 59.42, 66.87, 70.78, 34.81, 45.93, 48.36, 41.89, 45.26, 34.48, 35.93, 34.08, 66.55, 67.41, 37.0, 30.0, 67.5, 43.15, 65.86, 42.02, 33.61, 32.5, 37.58, 41.91, 60.96, 64.03, 72.49, 37.37, 37.47, 44.87, 45.32, 66.91, 65.39, 65.94, 58.53, 63.03, 43.16, 42.27, 50.06, 47.45, 55.56, 55.93, 42.14, 38.48, 42.72, 36.68, 36.26, 48.46, 33.68, 40.54, 50.99, 50.79, 42.24, 59.16, 42.87, 31.29, 36.32, 41.72, 36.16, 72.13, 69.39, 42.31, 37.44, 36.32, 72.67, 37.58, 43.44, 55.19, 62.65, 43.9, 47.75, 61.31, 59.82, 64.28, 52.72, 61.05, 40.0, 46.47, 39.88, 37.28, 58.0, 30.33, 60.4, 64.36, 65.57, 32.98, 45.01, 64.94, 57.59, 38.64, 41.41, 71.86, 69.62, 45.88, 58.5, 41.22, 50.85, 38.6, 59.1, 44.6, 43.58, 39.98, 69.18, 68.44, 66.07, 55.09, 40.41, 43.16, 32.55, 42.04, 48.45]

plt.hist(life\_exp, bins=15)

# Show and clear plot

plt.show()

plt.clf()

# Histogram of life\_exp1950, 15 bins

plt.hist(life\_exp1950, bins=15)

# Show and clear plot

plt.show()

plt.clf()

# Show and clear plot again

plt.show()

plt.clf()

plt.show()

print("life exp 1950 15 bins")

# Histogram of life\_exp1950, 15 bins

plt.hist(life\_exp1950, bins=15)

# Show and clear plot

plt.show()

plt.clf()

# Show and clear plot again

plt.show()

plt.clf()

###################### GRAPH CUTOMIZATIONS #################

gdp\_cap=[974.58033839999996, 5937.0295259999984, 6223.3674650000003, 4797.2312670000001, 12779.379639999999, 34435.367439999995, 36126.492700000003, 29796.048340000001, 1391.253792, 33692.605080000001, 1441.2848730000001, 3822.137084, 7446.2988029999997, 12569.851769999999, 9065.8008250000003, 10680.792820000001, 1217.0329939999999, 430.07069159999998, 1713.7786860000001, 2042.0952400000001, 36319.235009999997, 706.01653699999997, 1704.0637240000001, 13171.638849999999, 4959.1148540000004, 7006.5804189999999, 986.14787920000003, 277.55185870000003, 3632.5577979999998, 9645.06142, 1544.7501119999999, 14619.222719999998, 8948.1029230000004, 22833.308509999999, 35278.418740000001, 2082.4815670000007, 6025.3747520000015, 6873.2623260000009, 5581.1809979999998, 5728.3535140000004, 12154.089749999999, 641.36952360000021, 690.80557590000001, 33207.0844, 30470.0167, 13206.48452, 752.74972649999995, 32170.37442, 1327.6089099999999, 27538.41188, 5186.0500030000003, 942.6542111, 579.23174299999982, 1201.637154, 3548.3308460000007, 39724.978669999997, 18008.944439999999, 36180.789190000003, 2452.210407, 3540.6515639999998, 11605.71449, 4471.0619059999999, 40675.996350000001, 25523.277099999999, 28569.719700000001, 7320.8802620000015, 31656.068060000001, 4519.4611709999999, 1463.249282, 1593.06548, 23348.139730000006, 47306.989780000004, 10461.05868, 1569.3314419999999, 414.5073415, 12057.49928, 1044.7701259999999, 759.34991009999999, 12451.6558, 1042.581557, 1803.151496, 10956.991120000001, 11977.57496, 3095.7722710000007, 9253.896111, 3820.1752299999998, 823.68562050000003, 944.0, 4811.0604290000001, 1091.359778, 36797.933319999996, 25185.009109999999, 2749.3209649999999, 619.67689239999982, 2013.9773049999999, 49357.190170000002, 22316.192869999999, 2605.94758, 9809.1856360000002, 4172.8384640000004, 7408.9055609999996, 3190.4810160000002, 15389.924680000002, 20509.64777, 19328.709009999999, 7670.122558, 10808.47561, 863.08846390000019, 1598.4350890000001, 21654.83194, 1712.4721360000001, 9786.5347139999994, 862.54075610000018, 47143.179640000002, 18678.314350000001, 25768.257590000001, 926.14106830000003, 9269.6578079999999, 28821.063699999999, 3970.0954069999998, 2602.3949950000001, 4513.4806429999999, 33859.748350000002, 37506.419070000004, 4184.5480889999999, 28718.276839999999, 1107.482182, 7458.3963269999977, 882.9699437999999, 18008.509239999999, 7092.9230250000001, 8458.2763840000007, 1056.3801209999999, 33203.261279999999, 42951.65309, 10611.46299, 11415.805689999999, 2441.5764039999999, 3025.3497980000002, 2280.769906, 1271.211593, 469.70929810000007]

life\_exp=[43.828000000000003, 76.423000000000002, 72.301000000000002, 42.731000000000002, 75.319999999999993, 81.234999999999999, 79.828999999999994, 75.635000000000005, 64.061999999999998, 79.441000000000003, 56.728000000000002, 65.554000000000002, 74.852000000000004, 50.728000000000002, 72.390000000000001, 73.004999999999995, 52.295000000000002, 49.579999999999998, 59.722999999999999, 50.43, 80.653000000000006, 44.741000000000007, 50.651000000000003, 78.552999999999997, 72.960999999999999, 72.888999999999996, 65.152000000000001, 46.462000000000003, 55.322000000000003, 78.781999999999996, 48.328000000000003, 75.748000000000005, 78.272999999999996, 76.486000000000004, 78.331999999999994, 54.790999999999997, 72.234999999999999, 74.994, 71.338000000000022, 71.878, 51.578999999999994, 58.039999999999999, 52.947000000000003, 79.313000000000002, 80.656999999999996, 56.734999999999999, 59.448, 79.406000000000006, 60.021999999999998, 79.483000000000004, 70.259, 56.006999999999998, 46.388000000000012, 60.915999999999997, 70.198000000000008, 82.207999999999998, 73.338000000000022, 81.757000000000005, 64.698000000000008, 70.650000000000006, 70.963999999999999, 59.545000000000002, 78.885000000000005, 80.745000000000005, 80.546000000000006, 72.566999999999993, 82.602999999999994, 72.534999999999997, 54.109999999999999, 67.296999999999997, 78.623000000000005, 77.588000000000022, 71.992999999999995, 42.591999999999999, 45.677999999999997, 73.951999999999998, 59.443000000000012, 48.302999999999997, 74.241, 54.466999999999999, 64.164000000000001, 72.801000000000002, 76.194999999999993, 66.802999999999997, 74.543000000000006, 71.164000000000001, 42.082000000000001, 62.069000000000003, 52.906000000000013, 63.784999999999997, 79.762, 80.203999999999994, 72.899000000000001, 56.866999999999997, 46.859000000000002, 80.195999999999998, 75.640000000000001, 65.483000000000004, 75.536999999999978, 71.751999999999995, 71.421000000000006, 71.688000000000002, 75.563000000000002, 78.097999999999999, 78.746000000000024, 76.441999999999993, 72.475999999999999, 46.241999999999997, 65.528000000000006, 72.777000000000001, 63.061999999999998, 74.001999999999995, 42.568000000000012, 79.971999999999994, 74.662999999999997, 77.926000000000002, 48.158999999999999, 49.338999999999999, 80.941000000000003, 72.396000000000001, 58.555999999999997, 39.613, 80.884, 81.701000000000022, 74.143000000000001, 78.400000000000006, 52.517000000000003, 70.616, 58.420000000000002, 69.819000000000003, 73.923000000000002, 71.777000000000001, 51.542000000000002, 79.424999999999997, 78.242000000000004, 76.384, 73.747, 74.248999999999995, 73.421999999999997, 62.698, 42.383999999999993, 43.487000000000002]

plt.clf()

print("priniting scatter plot with customization")

plt.scatter(gdp\_cap, life\_exp)

plt.xscale('log')

plt.xlabel('GDP CAPITAL INCOME')

plt.ylabel('LIFE EXPECTANCY')

plt.title('GDP CAPITAL VS LIFE EXPECTANCY')

plt.show()

**#OUTPUT#**

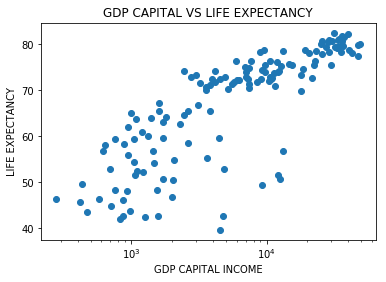
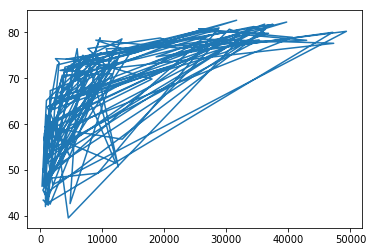
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]

Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

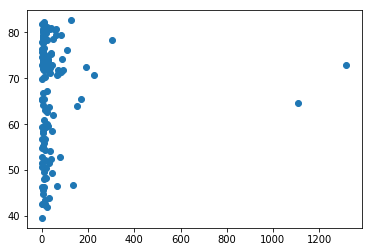
len(gdp\_cap) 142

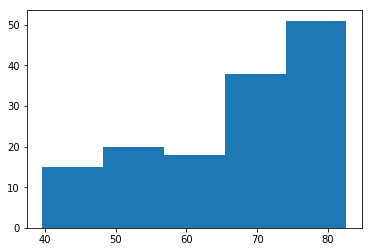
len(life\_exp) 142



prinitng len of population 142

prinitng len of LIFE\_EXPENTANCY 142





￼

Help on function hist in module matplotlib.pyplot:

hist(x, bins=None, range=None, density=None, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, normed=None, hold=None, data=None, \*\*kwargs)

Plot a histogram.

Compute and draw the histogram of \*x\*. The return value is a

tuple (\*n\*, \*bins\*, \*patches\*) or ([\*n0\*, \*n1\*, ...], \*bins\*,

[\*patches0\*, \*patches1\*,...]) if the input contains multiple

data.

Multiple data can be provided via \*x\* as a list of datasets

of potentially different length ([\*x0\*, \*x1\*, ...]), or as

a 2-D ndarray in which each column is a dataset. Note that

the ndarray form is transposed relative to the list form.

Masked arrays are not supported at present.

Parameters

----------

x : (n,) array or sequence of (n,) arrays

Input values, this takes either a single array or a sequency of

arrays which are not required to be of the same length

bins : integer or sequence or 'auto', optional

If an integer is given, ``bins + 1`` bin edges are calculated and

returned, consistent with :func:`numpy.histogram`.

If `bins` is a sequence, gives bin edges, including left edge of

first bin and right edge of last bin. In this case, `bins` is

returned unmodified.

All but the last (righthand-most) bin is half-open. In other

words, if `bins` is::

[1, 2, 3, 4]

then the first bin is ``[1, 2)`` (including 1, but excluding 2) and

the second ``[2, 3)``. The last bin, however, is ``[3, 4]``, which

\*includes\* 4.

Unequally spaced bins are supported if \*bins\* is a sequence.

If Numpy 1.11 is installed, may also be ``'auto'``.

Default is taken from the rcParam ``hist.bins``.

range : tuple or None, optional

The lower and upper range of the bins. Lower and upper outliers

are ignored. If not provided, \*range\* is ``(x.min(), x.max())``.

Range has no effect if \*bins\* is a sequence.

If \*bins\* is a sequence or \*range\* is specified, autoscaling

is based on the specified bin range instead of the

range of x.

Default is ``None``

Default is ``None`` for both \*normed\* and \*density\*. If either is

set, then that value will be used. If neither are set, then the

args will be treated as ``False``.

If both \*density\* and \*normed\* are set an error is raised.

weights : (n, ) array\_like or None, optional

An array of weights, of the same shape as \*x\*. Each value in \*x\*

only contributes its associated weight towards the bin count

(instead of 1). If \*normed\* or \*density\* is ``True``,

the weights are normalized, so that the integral of the density

over the range remains 1.

Default is ``None``

cumulative : boolean, optional

If ``True``, then a histogram is computed where each bin gives the

counts in that bin plus all bins for smaller values. The last bin

gives the total number of datapoints. If \*normed\* or \*density\*

is also ``True`` then the histogram is normalized such that the

last bin equals 1. If \*cumulative\* evaluates to less than 0

(e.g., -1), the direction of accumulation is reversed.

In this case, if \*normed\* and/or \*density\* is also ``True``, then

the histogram is normalized such that the first bin equals 1.

Default is ``False``

bottom : array\_like, scalar, or None

Location of the bottom baseline of each bin. If a scalar,

the base line for each bin is shifted by the same amount.

If an array, each bin is shifted independently and the length

of bottom must match the number of bins. If None, defaults to 0.

Default is ``None``

histtype : {'bar', 'barstacked', 'step', 'stepfilled'}, optional

The type of histogram to draw.

- 'bar' is a traditional bar-type histogram. If multiple data

are given the bars are aranged side by side.

- 'barstacked' is a bar-type histogram where multiple

data are stacked on top of each other.

- 'step' generates a lineplot that is by default

unfilled.

- 'stepfilled' generates a lineplot that is by default

filled.

Default is 'bar'

align : {'left', 'mid', 'right'}, optional

Controls how the histogram is plotted.

- 'left': bars are centered on the left bin edges.

- 'mid': bars are centered between the bin edges.

- 'right': bars are centered on the right bin edges.

Default is 'mid'

orientation : {'horizontal', 'vertical'}, optional

If 'horizontal', `~matplotlib.pyplot.barh` will be used for

bar-type histograms and the \*bottom\* kwarg will be the left edges.

rwidth : scalar or None, optional

The relative width of the bars as a fraction of the bin width. If

``None``, automatically compute the width.

Ignored if \*histtype\* is 'step' or 'stepfilled'.

Default is ``None``

log : boolean, optional

If ``True``, the histogram axis will be set to a log scale. If

\*log\* is ``True`` and \*x\* is a 1D array, empty bins will be

filtered out and only the non-empty ``(n, bins, patches)``

will be returned.

Default is ``False``

color : color or array\_like of colors or None, optional

Color spec or sequence of color specs, one per dataset. Default

(``None``) uses the standard line color sequence.

Default is ``None``

label : string or None, optional

String, or sequence of strings to match multiple datasets. Bar

charts yield multiple patches per dataset, but only the first gets

the label, so that the legend command will work as expected.

default is ``None``

stacked : boolean, optional

If ``True``, multiple data are stacked on top of each other If

``False`` multiple data are aranged side by side if histtype is

'bar' or on top of each other if histtype is 'step'

Default is ``False``

Returns

-------

n : array or list of arrays

The values of the histogram bins. See \*normed\* or \*density\*

and \*weights\* for a description of the possible semantics.

If input \*x\* is an array, then this is an array of length

\*nbins\*. If input is a sequence arrays

``[data1, data2,..]``, then this is a list of arrays with

the values of the histograms for each of the arrays in the

same order.

bins : array

The edges of the bins. Length nbins + 1 (nbins left edges and right

edge of last bin). Always a single array even when multiple data

sets are passed in.

patches : list or list of lists

Silent list of individual patches used to create the histogram

or list of such list if multiple input datasets.

Other Parameters

----------------

\*\*kwargs : `~matplotlib.patches.Patch` properties

See also

--------

hist2d : 2D histograms

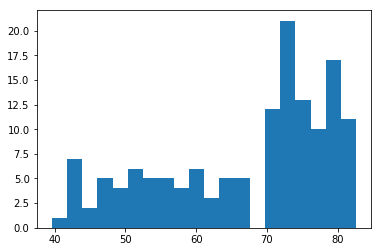
.. note::

In addition to the above described arguments, this function can take a

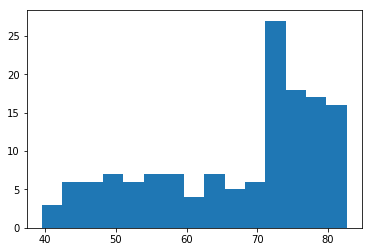
\*\*data\*\* keyword argument. If such a \*\*data\*\* argument is given, the

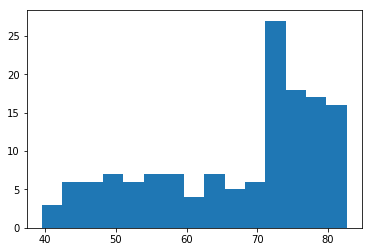
following arguments are replaced by \*\*data[<arg>]\*\*:

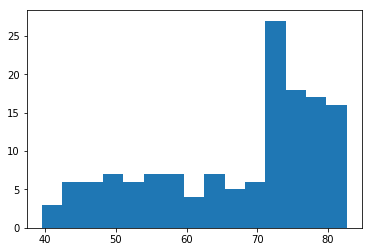
\* All arguments with the following names: 'weights', 'x'.

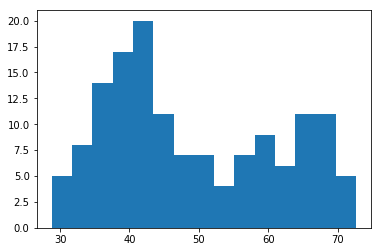


just life\_exp 15 n=bins





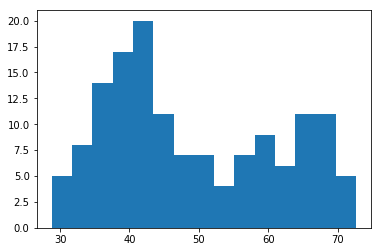




<matplotlib.figure.Figure at 0x11e56583358>

<matplotlib.figure.Figure at 0x11e5637a898>

life exp 1950 15 bins



<matplotlib.figure.Figure at 0x11e566475c0>

priniting scatter plot with customization

