Lab 05 - matplotlib I

Name: Dempsey Wade

Class: CSCI 349 - Intro to Data Mining

Semester: 2019SP

Instructor: Brian King

In [1]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as pat

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

1) [P] Create a dataframe named df_uniform that contains 1000 observations. It should have two variables, named x and y. For each observation, x should be generated from a uniform distribution between 10 and 90, and y should be generated from a uniform between 20 and 80.

In [2]:
```
np.randomseed = 1234
x = np.random.uniform(10, 90, 500)
y = np.random.uniform(20, 80, 500)
temp1 = pd.Series(x, index=x)
temp2 = pd.Series(y, index = y)
df_uniform = pd.DataFrame({'x': x, 'y':y})
#print(df_uniform)
```

2) [P] Generate a scatterplot of the data. Your plot must:

a. Have a title

b. Label both axes with "x" and "y" respectively

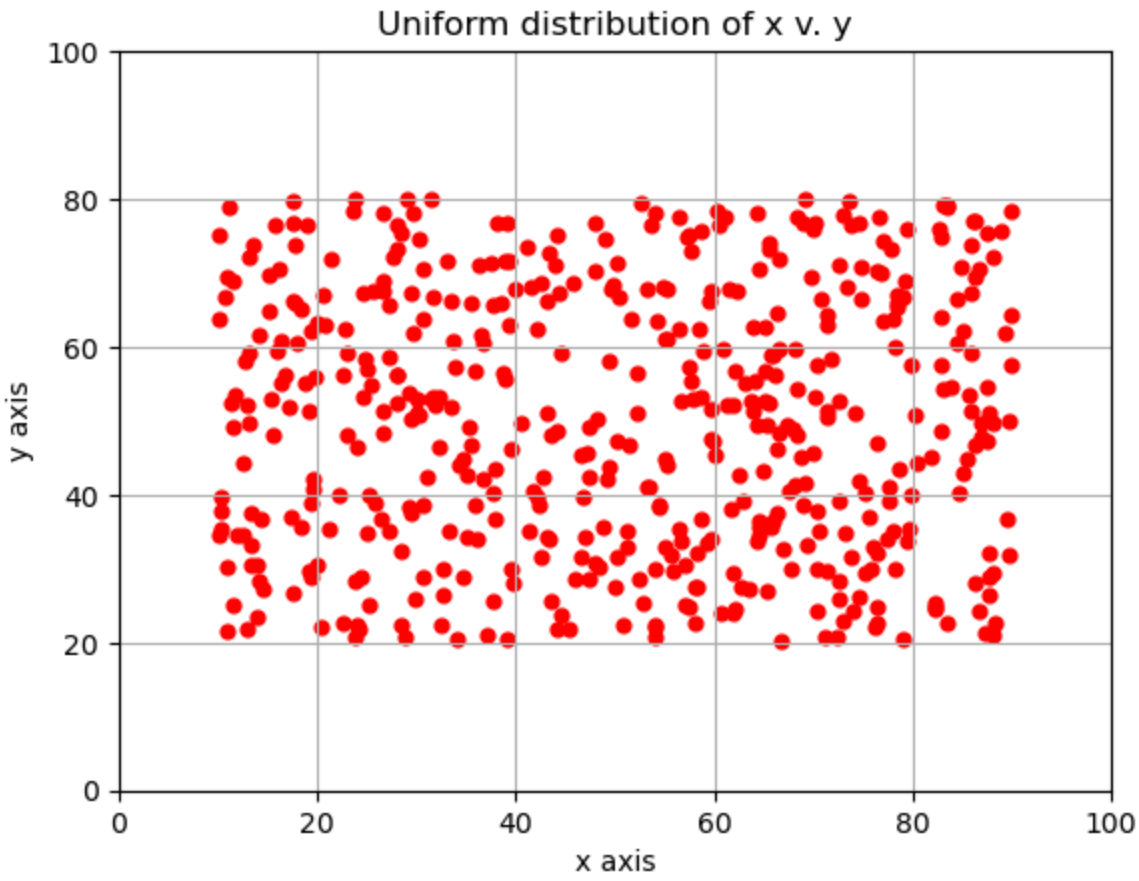
c. Change the x and y axis to display between 0 and 100

d. Change the default point size

e. Change the default color of the point

f. Display a grid

In [3]:
```
plt.scatter(df_uniform['x'], df_uniform['y'], s = 24, color = "red")
plt.title("Uniform distribution of x v. y")
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.grid()
plt.show()
```

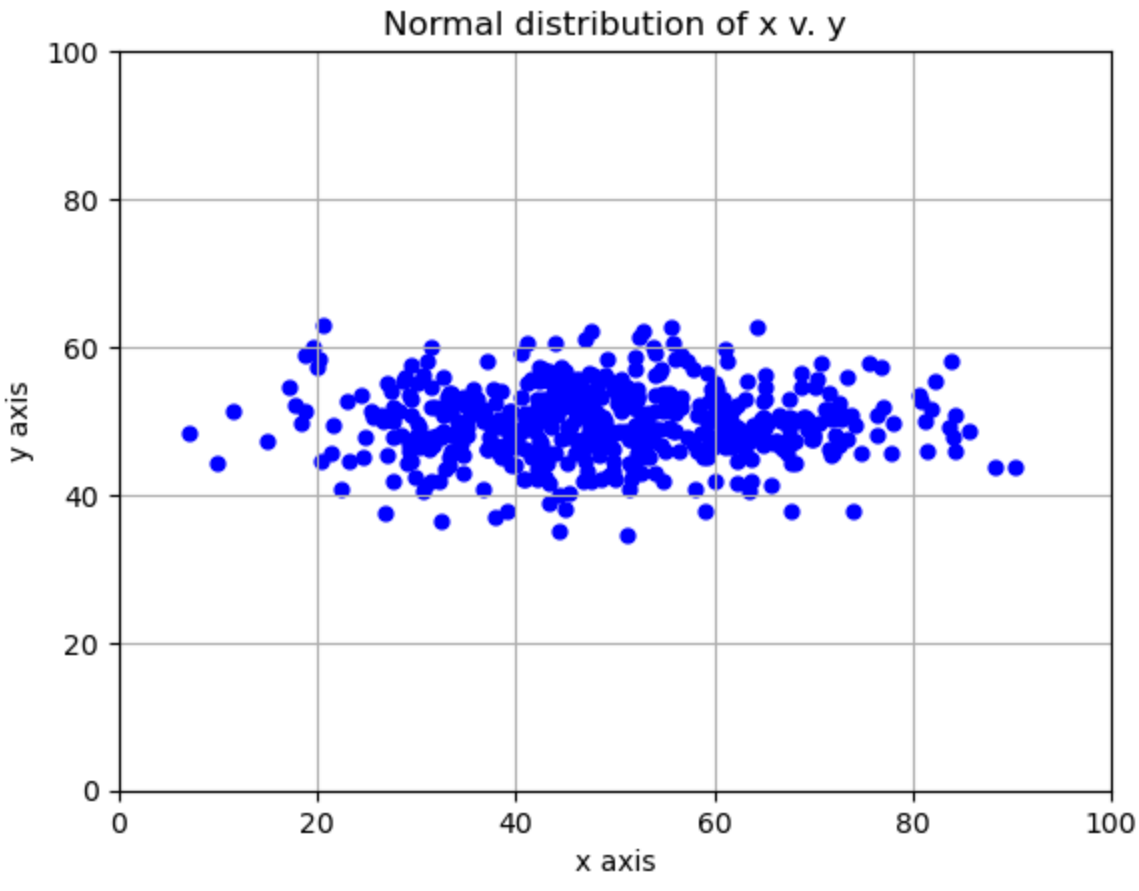


3) [P] Generate a data frame called df_normal with 1000 observations, two variables names x and y again. This time, x should be generated from a normal distribution with mean 50 and standard deviation 15, and y with mean 50 and standard deviation 5.

In [4]:
```
np.randomseed = 1234
x = np.random.normal(50, 15, 500)
y = np.random.normal(50, 5, 500)
temp1 = pd.Series(x, index=x)
temp2 = pd.Series(y, index = y)
df_normal = pd.DataFrame({'x': x, 'y':y})
```

4) [P] Repeat your plot above with df_normal, but use a different color point, and title your plot accordingly.

In [5]:
```
plt.scatter(df_normal['x'], df_normal['y'], s = 24, color = "blue")
plt.title("Normal distribution of x v. y")
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.grid()
plt.show()
```



5) [P] Generate a single figure that contains two axes that are adjacent to each other. You should have:

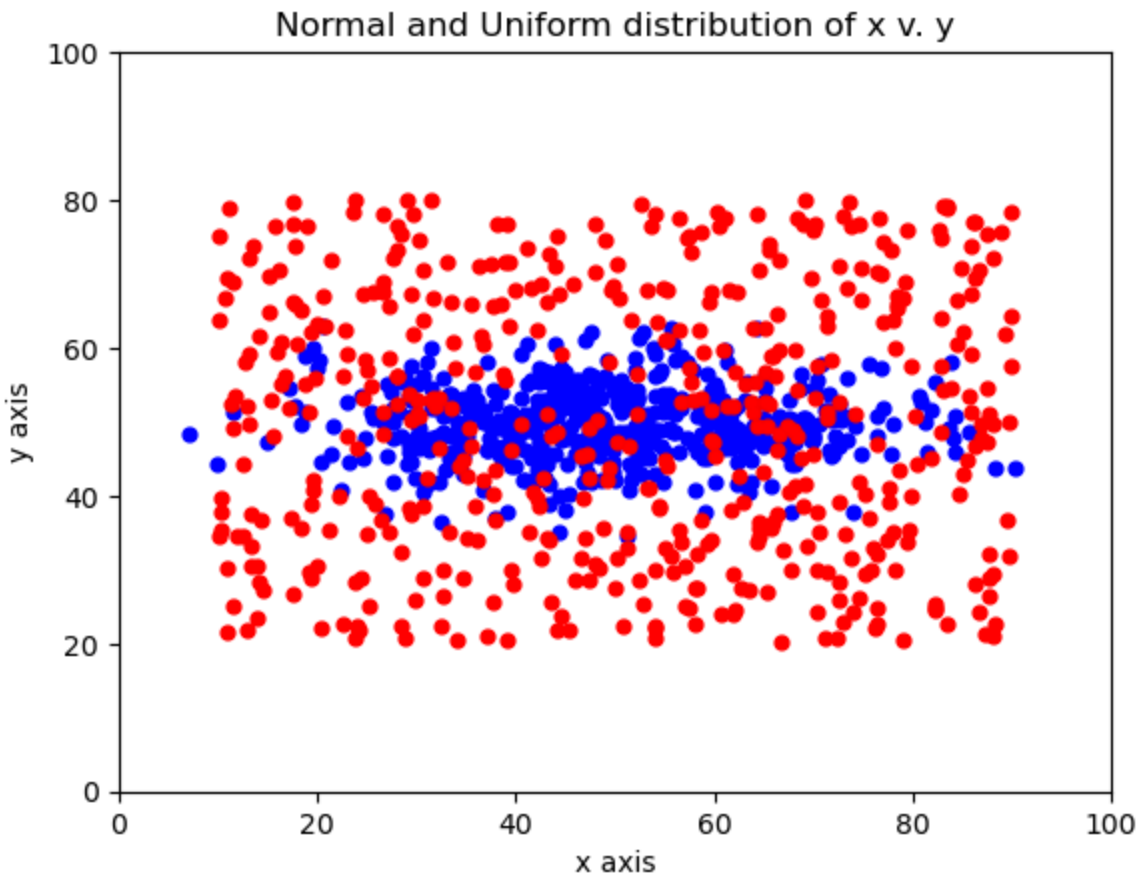
a. at least one shared axis

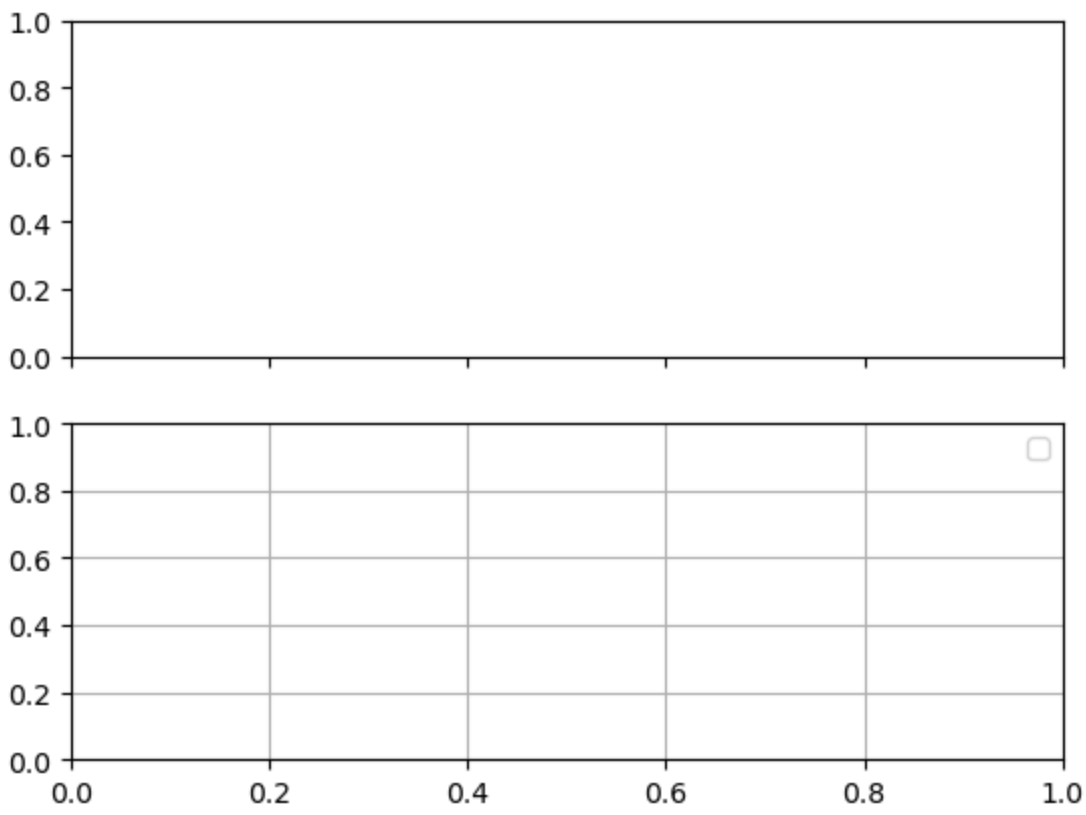
b. appropriate axis labels

c. make the range of the axis on both plots the same

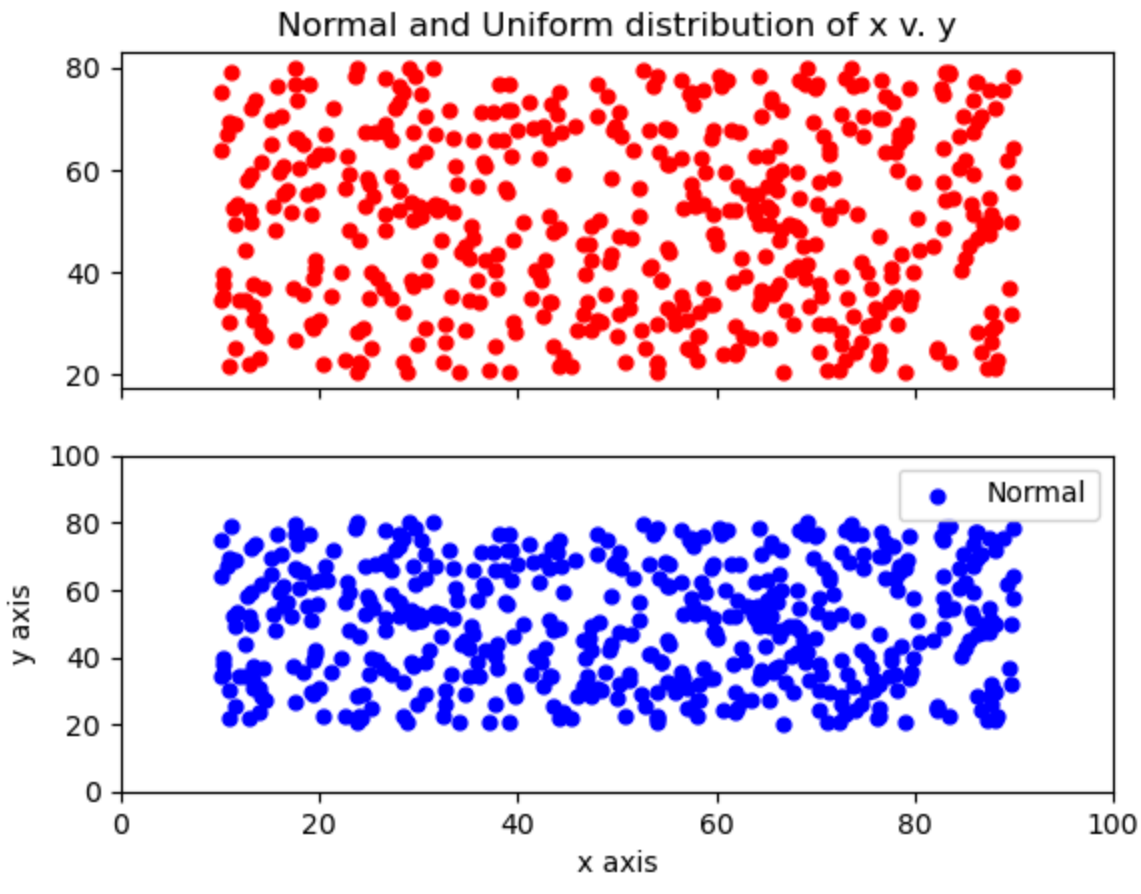
d. display a legend on each to be sure both are labeled correctly as "normal" or "uniform" e. One title at the top

In [6]:
```
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Normal and Uniform distribution of x v. y")
plt.xlim(0, 100)
plt.ylim(0, 100)
one = plt.scatter(df_normal['x'], df_normal['y'], s = 24, color = "blue")
two = plt.scatter(df_uniform['x'], df_uniform['y'], s = 24, color = "red")

plt.subplots(2, sharex = True)

plt.grid()
labels = ['Normal', 'Uniform']
plt.legend(labels)
plt.show()
```



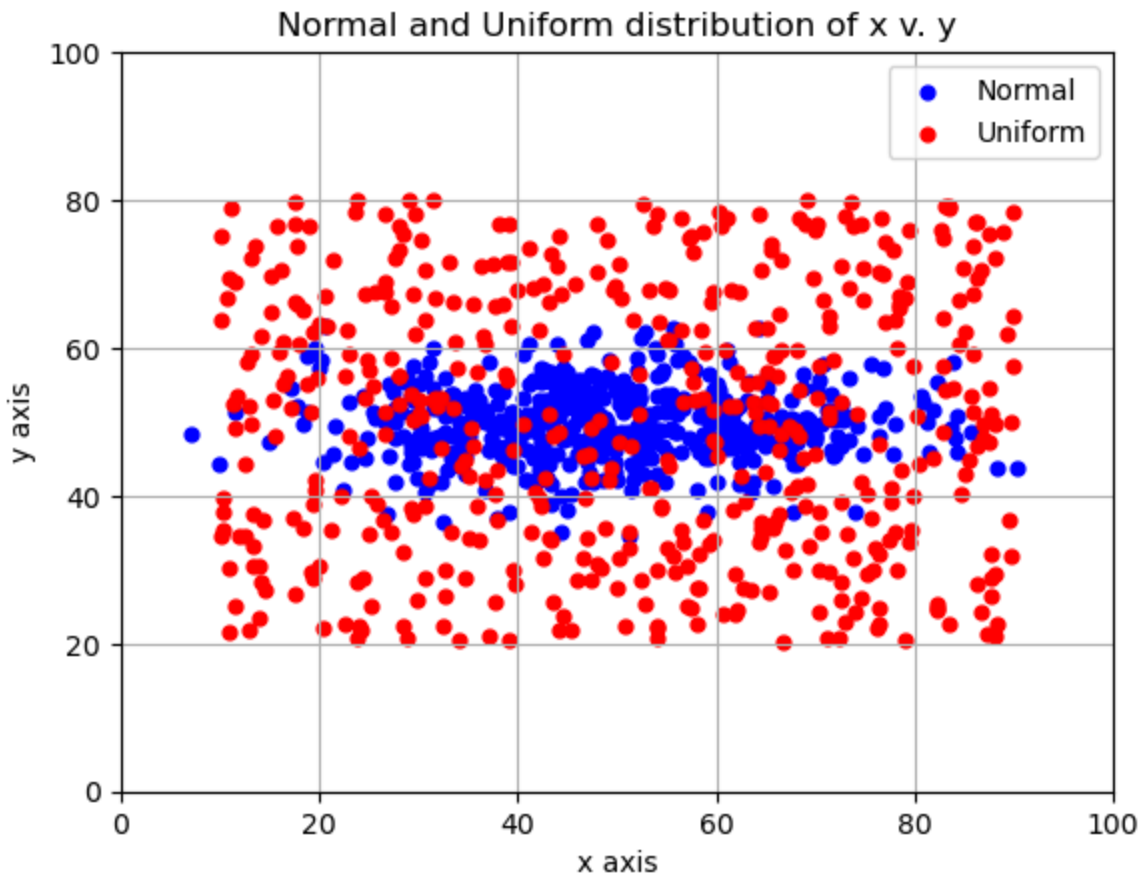


In [7]:
```
f, axarr = plt.subplots(2, sharex=True)
axarr[0].scatter(df_uniform['x'], df_uniform['y'], s = 24, color = 'red')
axarr[1].scatter(df_uniform['x'], df_uniform['y'], s = 24, color = 'blue')
plt.xlabel('x axis')
plt.ylabel('y axis')
axarr[0].set_title("Normal and Uniform distribution of x v. y")
plt.xlim(0, 100)
plt.ylim(0, 100)
labels = ['Normal', 'Uniform']
plt.legend(labels)
plt.show()
```



6) [P] Display both df_uniform and df_normal on one shared plot, with an appropriate legend

```
In [8]: plt.scatter(df_normal['x'], df_normal['y'], s = 24, color = "blue")
        plt.scatter(df_uniform['x'], df_uniform['y'], s = 24, color = "red")
        #colors = np.where(df_uniform, 'r', 'k')
        plt.title("Normal and Uniform distribution of x v. y")
        plt.xlabel('x axis')
        plt.ylabel('y axis')
        plt.xlim(0, 100)
        plt.ylim(0, 100)
        plt.grid()
        labels = ['Normal', 'Uniform']
        plt.legend(labels)
        #plt.legend(loc='upper left')
        plt.show()
```



7) [M] What is a histogram? In your answer, please clearly indicate what it is, why one would use it during their EDA phase, and whether it's good for one variable or to show relationships between multiple variables.

A histogram is a graph where the data inputed is translated into rectangles, whose area is proportianal to the frequency. During the EDA phase, one would want to graphically summarize their data, in which case a histogram would be a good choice. A histogram is most efficient for multiple variables. 1 variable would no help show it's relationship to other variables, therefore rendering a histogram worthless.

8) [M] What is a boxplot? In your answer, please clearly indicate what it is, why one would use it during their EDA phase, and whether it's good for one variable or to show relationships between multiple variables.

A box plot is a graph in which roughly 75% of the data is contained inside of box, and the remaining data is represented as points on the outside of the box. A box plot is very helpful to identify outliers, since the box for each variable only contains points that

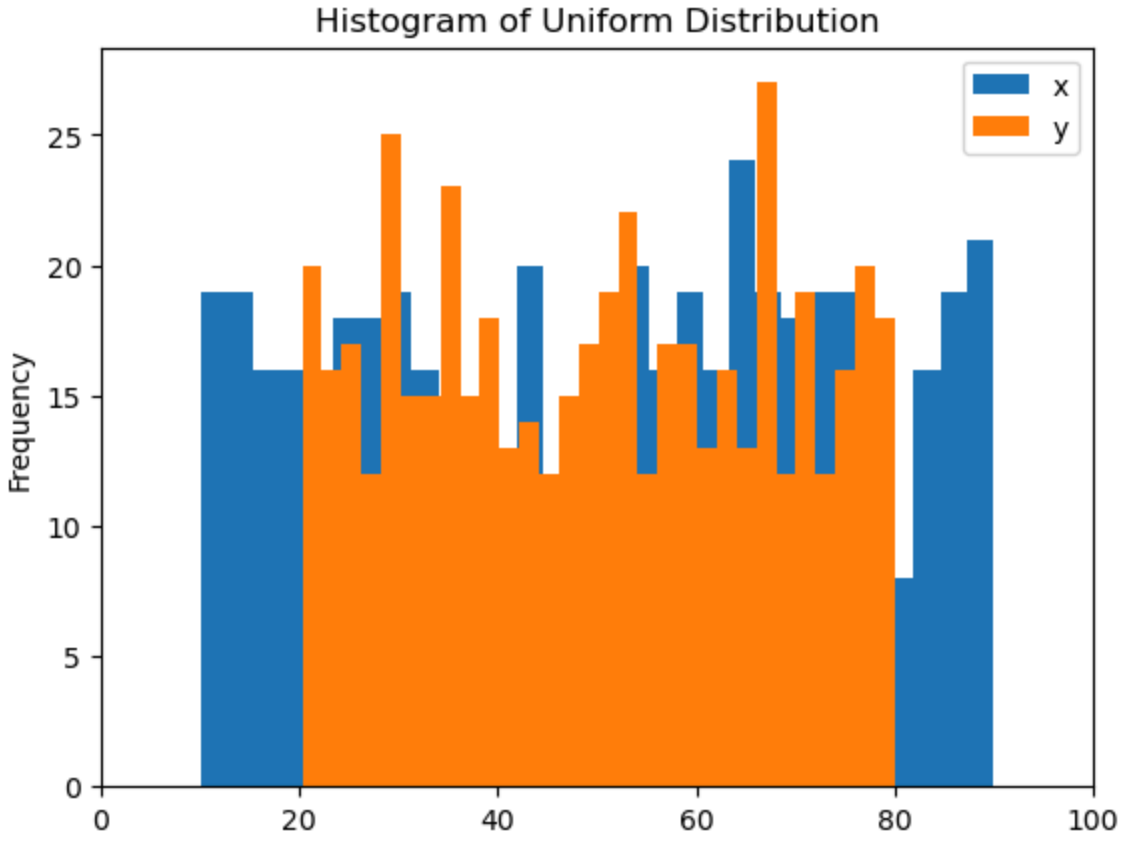
are contained within 1.5*IQR. A box plot is good for analyzing one variable and can also show the relationship between multiple variables.

9) [M] What is a density plot? In your answer, please clearly indicate what it is, why one would use it during their EDA phase, and whether it's good for one variable or to show relationships between multiple variables.

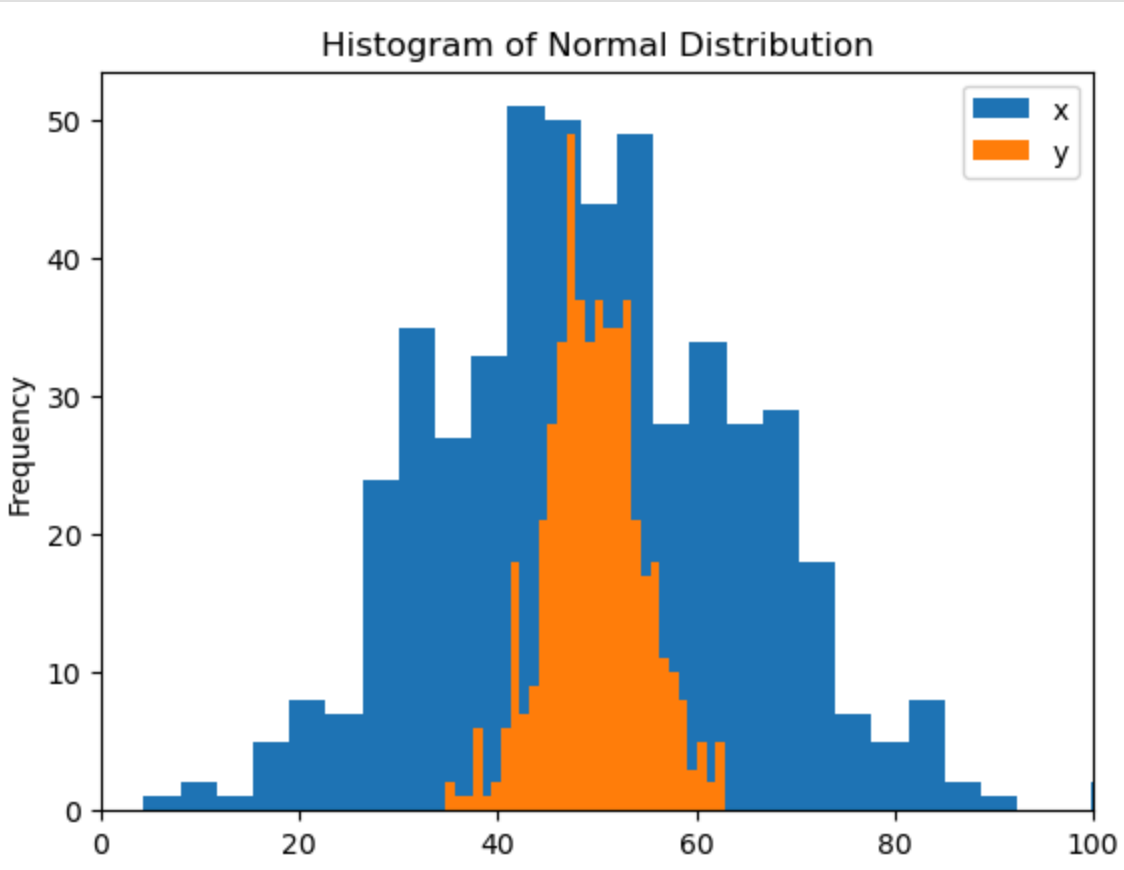
A density plot is another form of a graph that visualy represents the distribution of data over a chosen interval. Similiar to a box plot, it can be very helpful identifying the odds of getting a certain value and identifying outliers. Its best used for one variable, but you can compare the shapes of a density plot to other variables for further analysis.

10 [P] The pandas DataFrame class has a useful interface to matplotlib that will help you generate some quick plots as you explore your data. To get you started, generate a histogram of both the x and y variables for df_uniform. Use 30 bins, and set the range of both variables to be 0 – 100. Repeat this exercise on df_normal.

In [9]:
```
colors = ['red', 'blue']
df_uniform['x'].plot(kind = 'hist', bins = 30)
df_uniform['y'].plot(kind = 'hist', bins = 30)
plt.xlim(0, 100)
labels = ['x', 'y']
plt.legend(labels)
plt.title("Histogram of Uniform Distribution")
plt.show()
```

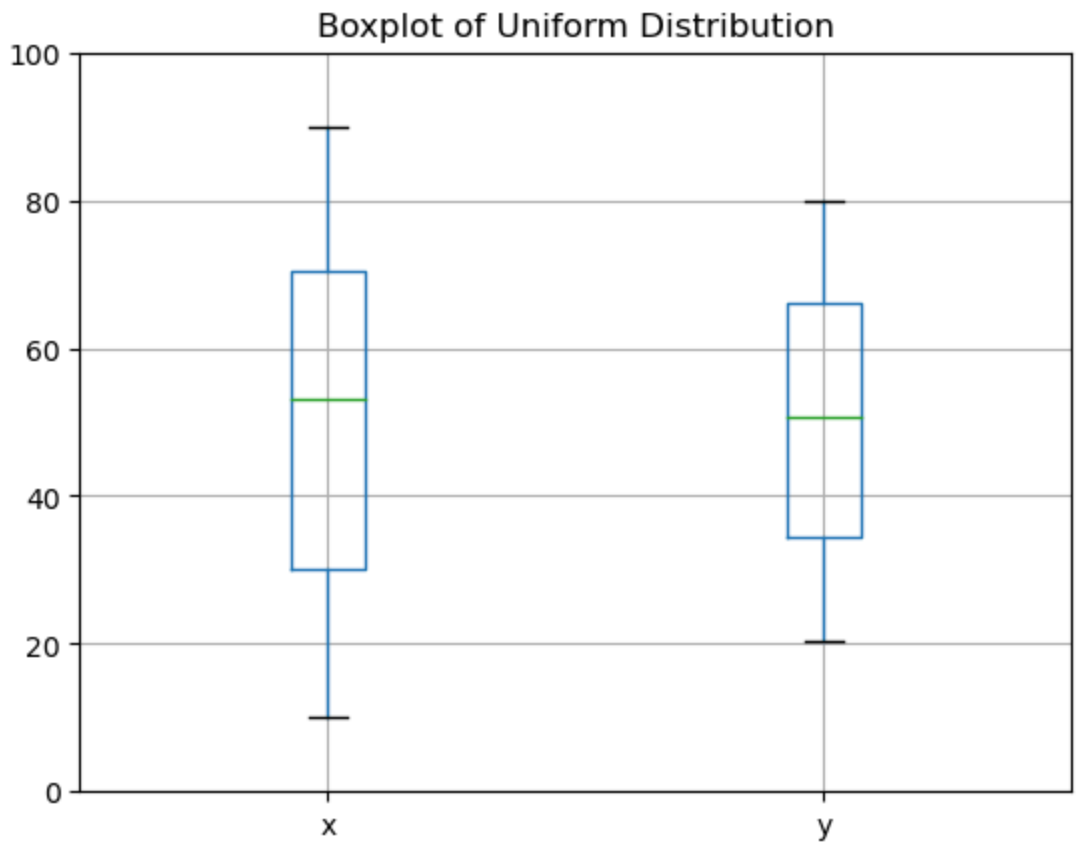


In [10]:
```
df_normal['x'].plot(kind = 'hist', bins = 30)
df_normal['y'].plot(kind = 'hist', bins = 30)
plt.xlim(0, 100)
plt.legend()
plt.title("Histogram of Normal Distribution")
plt.show()
```



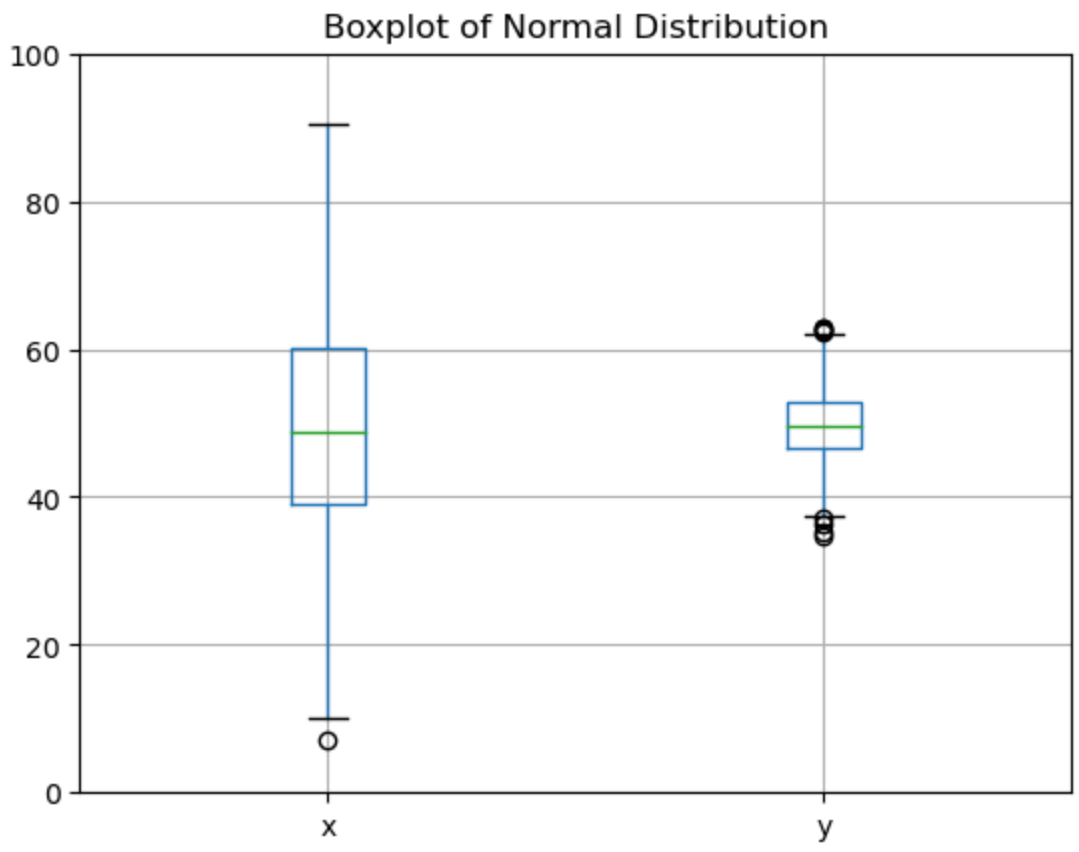
11) [P] Repeat the previous exercise to generate a box plot on both x and y variables of both df_uniform and df_normal.

In [11]:
```
df_uniform.boxplot(column = ['x', 'y'])
plt.ylim(0, 100)
plt.title("Boxplot of Uniform Distribution")
plt.show()
```



In [12]:
```
df_normal.boxplot(column = ['x', 'y'])
plt.ylim(0, 100)
plt.title("Boxplot of Normal Distribution")
```

Out[12]:
```
Text(0.5, 1.0, 'Boxplot of Normal Distribution')
```



12) [M] What is a quantile? What is a quartile? What is an Inter-quartile range (IQR)? Interpret the boxplot results in these terms.

A quantile is a subset of your data that distributes the values evenly. A quartile is 1/4 of your data. It's grouped into 4 sets of 25%, 50%, 75%, or 100%. An IQR is used to find outliers. If any value is less than Q1 - 1.5*IQR or greater than Q3 + 1.5*IQR, it is considered an outlier

In the boxplots above, the values that are close to the median are found inside of the box, and the lines with bars on them represent the outliers. Any data points located outside of those "whiskers" are considered outliers.

13 [M,P] Read about the quantile() method for dataframes, and use it to numerically show the 25th, median, and 75th percentiles, and compute the IQR (Inter-quartile range) for both variables, on both data frames. Compare and contrast.

In [13]:
```
print('25th quartile: \n',df_normal.quantile(.25))
print()
print('50th quartile: \n',df_normal.quantile(.5))
print()
print('75th quartile: \n',df_normal.quantile(.75))
print()
print('IQR for normal distribution: ', df_normal.quantile(.75) - df_normal.quantile(.25))
print()
print('25th quartile: \n',df_uniform.quantile(.25))
```

```
print()
print('50th quartile: \n',df_uniform.quantile(.5))
print()
print('75th quartile: \n',df_uniform.quantile(.75))
print()
print('IQR for uniform distribution: \n', df_uniform.quantile(.75) - df_uniform.quantile(.25))
print()
```

```
25th quartile:
 x    39.164675
y    46.664651
Name: 0.25, dtype: float64

50th quartile:
 x    48.724992
y    49.535241
Name: 0.5, dtype: float64

75th quartile:
 x    60.177894
y    52.875215
Name: 0.75, dtype: float64

IQR for normal distribution:  x    21.013219
y     6.210564
dtype: float64

25th quartile:
 x    29.992158
y    34.538914
Name: 0.25, dtype: float64

50th quartile:
 x    53.211991
y    50.670905
Name: 0.5, dtype: float64

75th quartile:
 x    70.349706
y    65.966655
Name: 0.75, dtype: float64

IQR for uniform distribution:
 x    40.357548
y    31.427741
dtype: float64
```

The 50th quartile was roughly the same for both data sets, which is to be expected. Normal distribution has a smaller deviation from the 50th quartile, when comparing to the 25th and 75th quartile, than uniform distribution had which is also to be expected.

14) [M,P] What is the standard definition of an outlier, in terms of IQR? Use the IQR to determine what the outliers are for each variable in each dataset, if any.

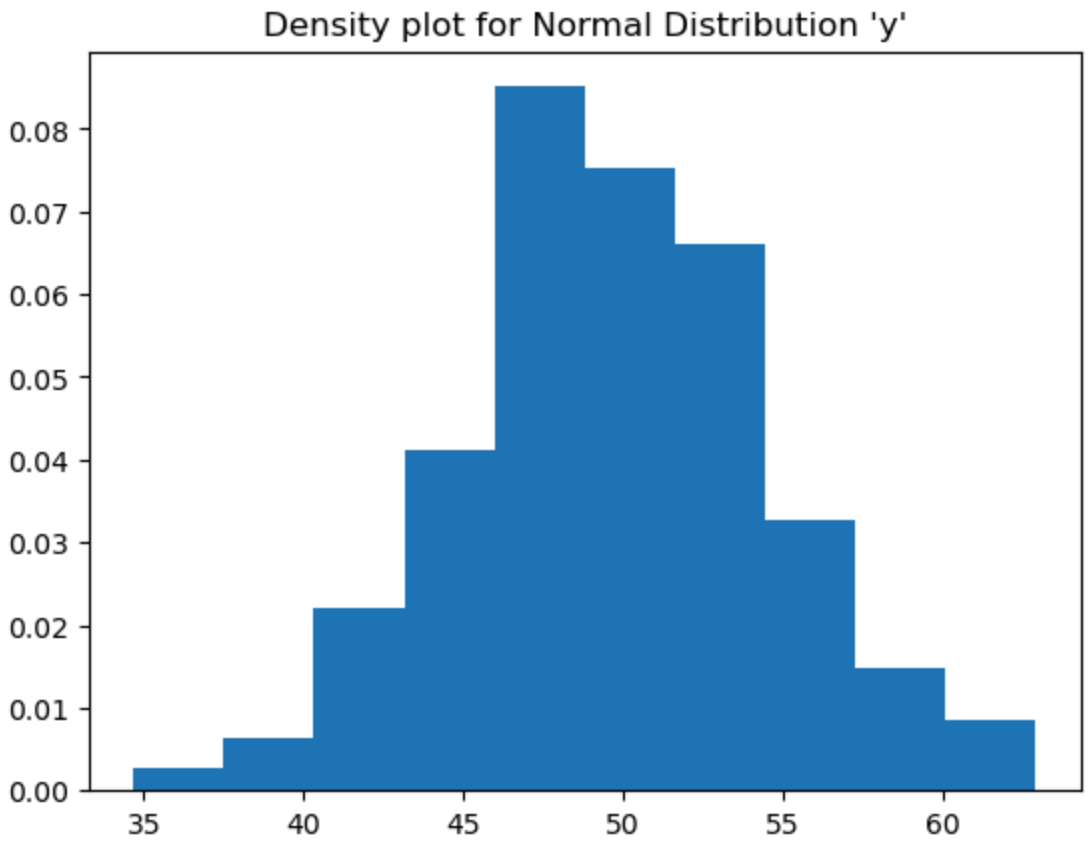
A value is considered an outlier when it falls below Q1 - 1.5IQR or is greater than Q3 + 1.5IQR.

In [14]:
```
iqr_u = df_uniform.quantile(.75) - df_uniform.quantile(.25)
iqr_n = df_normal.quantile(.75) - df_normal.quantile(.25)
print('A uniform outlier exists if a value if above \n', df_uniform.quantile(.75) + 1.5*iqr_u)
print('A uniform outlier exists if a value if below \n', df_uniform.quantile(.25) - 1.5*iqr_u)

print('A normal outlier exists if a value if above \n', df_normal.quantile(.75) + 1.5*iqr_n)
print('A normal outlier exists if a value if below \n', df_normal.quantile(.25) - 1.5*iqr_n)
```

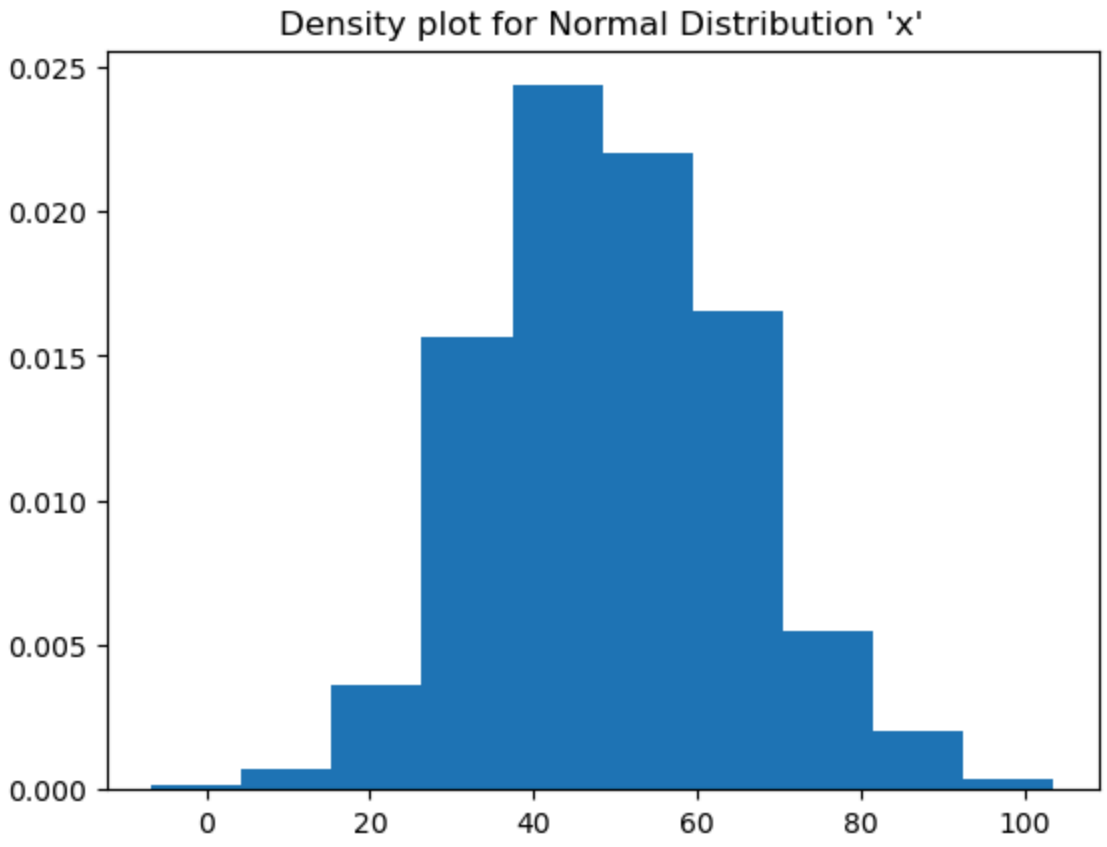
```
A uniform outlier exists if a value if above
 x    130.886028
y    113.108267
dtype: float64
A uniform outlier exists if a value if below
 x   -30.544164
y   -12.602698
dtype: float64
A normal outlier exists if a value if above
 x    91.697722
y    62.191061
dtype: float64
A normal outlier exists if a value if below
 x     7.644847
y    37.348804
dtype: float64
```

15) [P] Generate a density plot for both x and y variables of both df_uniform and df_normal.

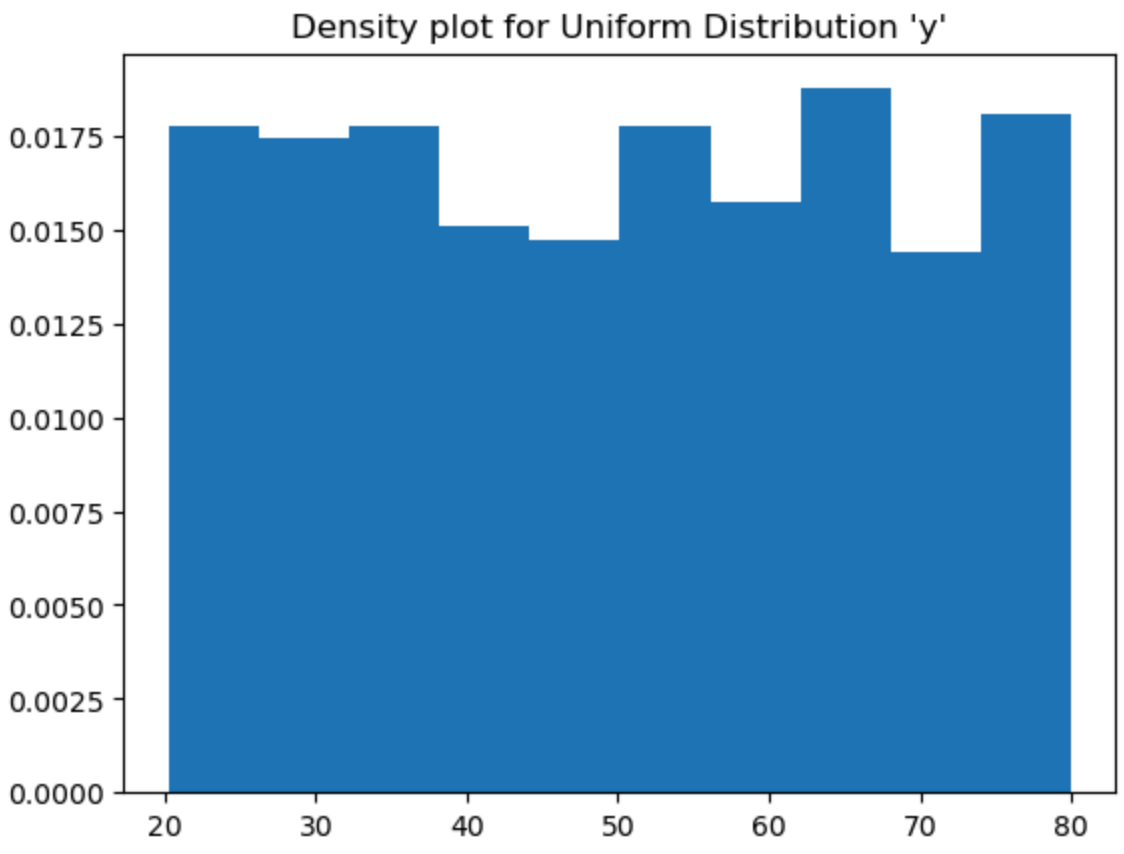
In [15]:
```
plt.hist(df_normal['y'], density = True, stacked = False)
plt.title("Density plot for Normal Distribution 'y'")
plt.show()
```



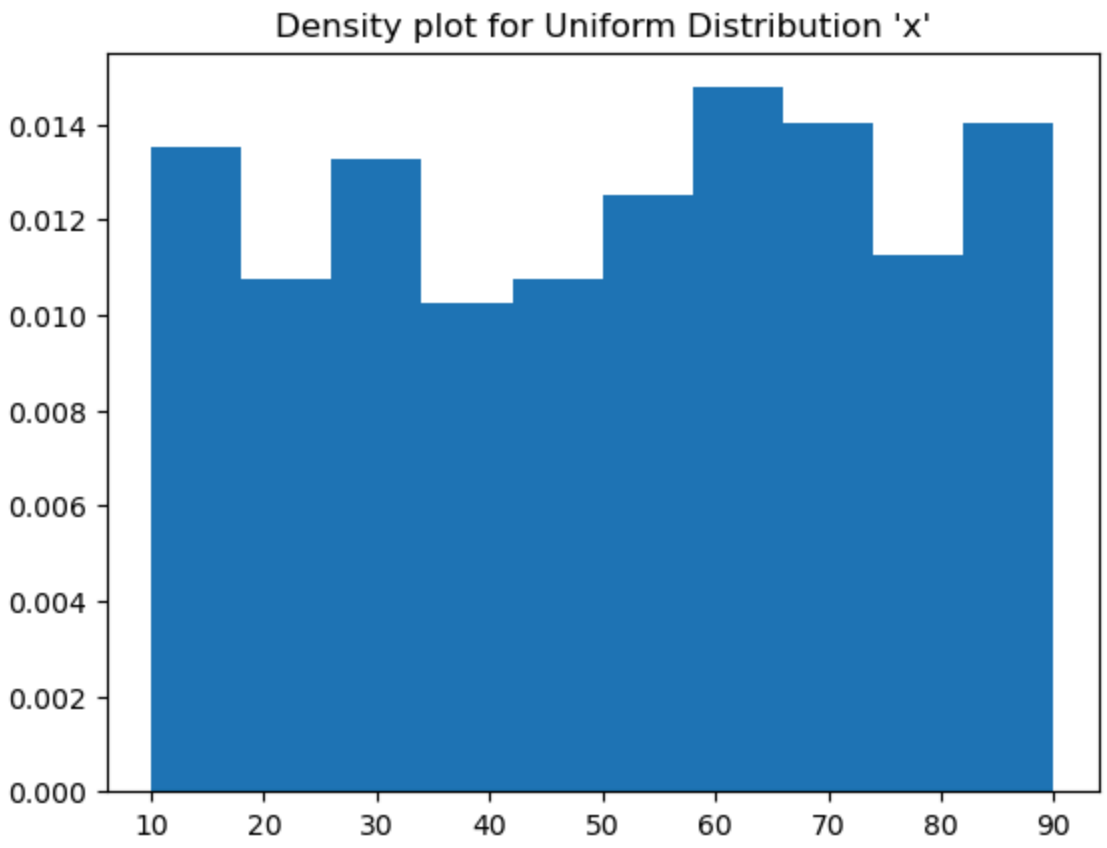
In [16]:
```
plt.hist(df_normal['x'], density = True, stacked = True)
plt.title("Density plot for Normal Distribution 'x'")
plt.show()
```



In [17]:
```
plt.hist(df_uniform['y'], density = True, stacked = True)
plt.title("Density plot for Uniform Distribution 'y'")
plt.show()
```



In [18]:
```
plt.hist(df_uniform['x'], density = True, stacked = True)
plt.title("Density plot for Uniform Distribution 'x'")
plt.show()
```



16) [M] Interepret the density plot results

As expected, the density plot for uniform distribution shows an overall rectange shape while the density plot for normal distribution looks more like a bell curve. This makes sense since uniform distribution would aim to have the same frequency for all values, bu definition. Normal distribution has a larger frequency of values around the median, and this is confirmed in the density plot.

17) [P] Go back to the describe() method you learn about in previous labs. Show the results of describe() for both data frames.

In [19]:
```
print(df_uniform.describe())
print(df_normal.describe())
```

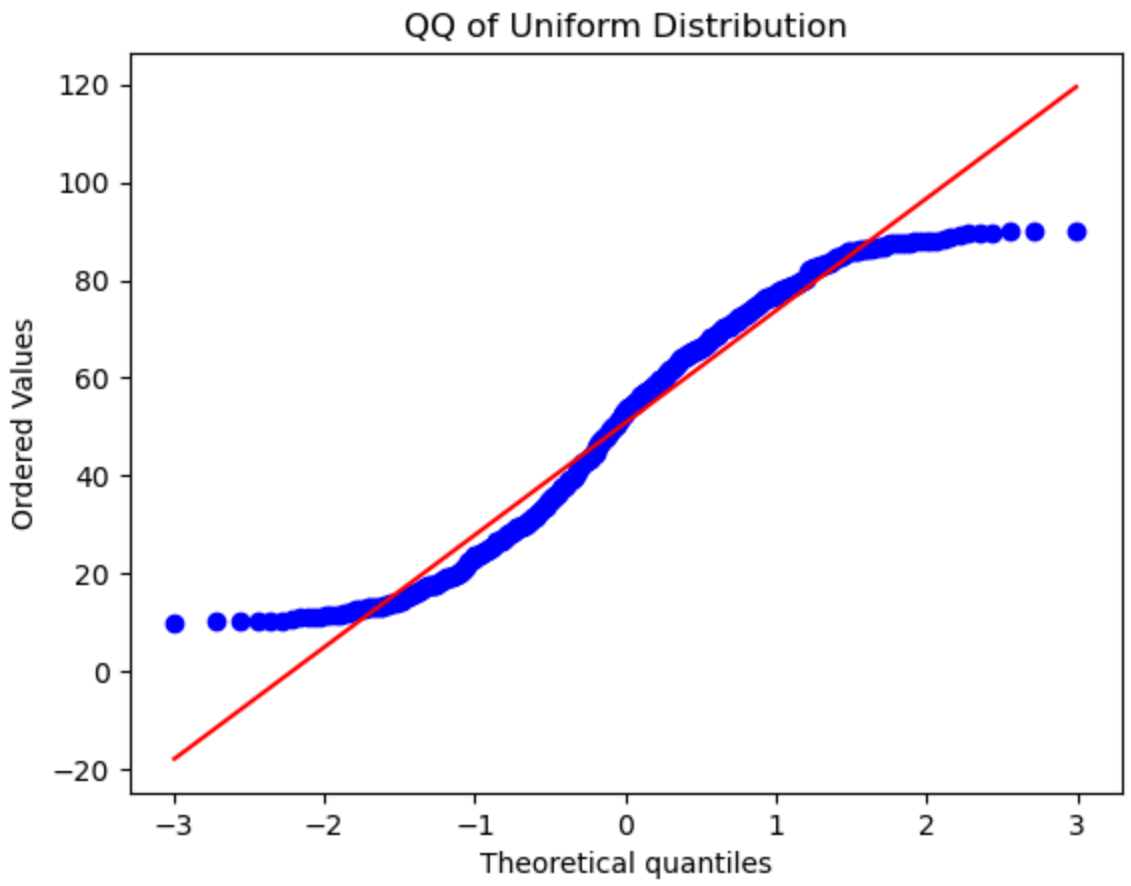
```
                x           y
count  500.000000  500.000000
mean    50.793054   49.964214
std     23.408354   17.568849
min     10.021620   20.277499
25%     29.992158   34.538914
50%     53.211991   50.670905
75%     70.349706   65.966655
max     89.933504   79.983488
                x           y
count  500.000000  500.000000
mean    49.439827   49.678452
std     15.556445    4.886542
min     -6.696019   34.682059
25%     39.164675   46.664651
50%     48.724992   49.535241
75%     60.177894   52.875215
max    103.394798   62.884215
```

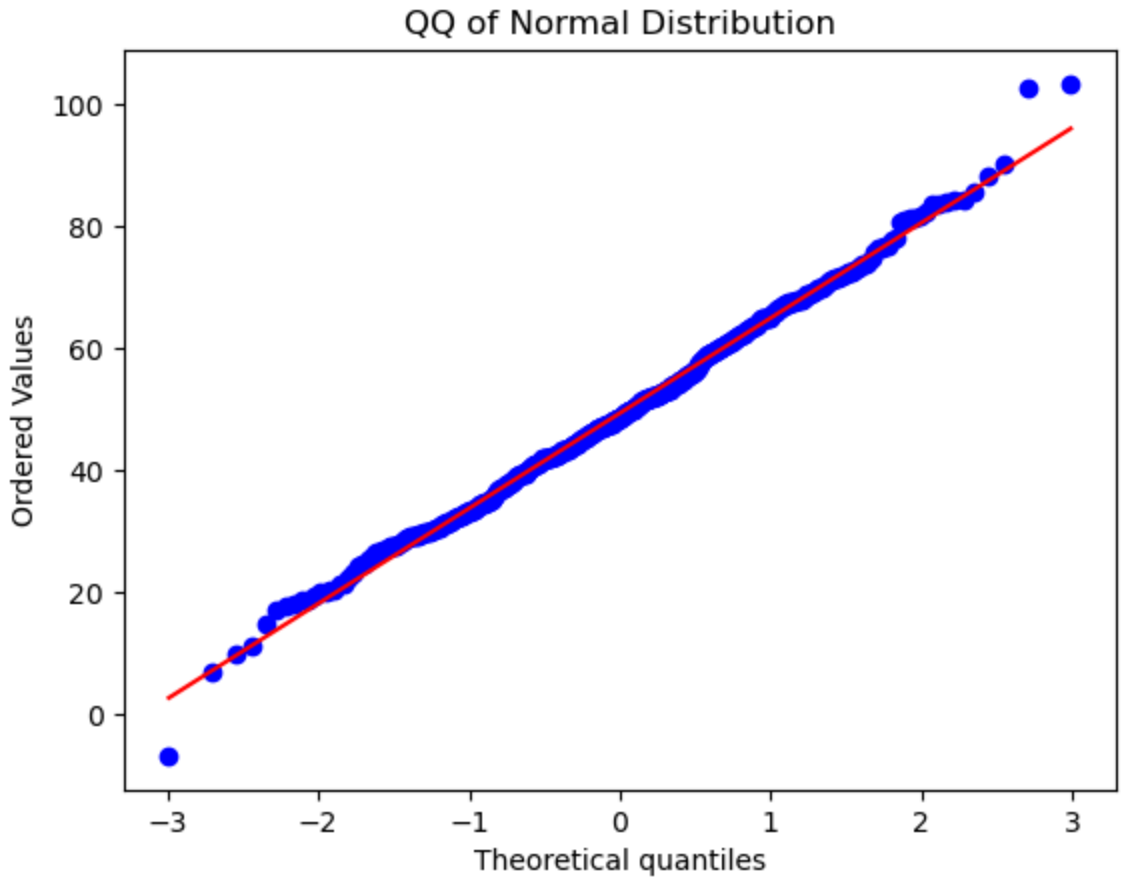
18) [M] What is a quantile-quantile plot?

A q-q plot compares two data sets to see if they are related, i.e. if they come from the same population.

19) [P, M] Load the scipy.stats package, and read about the probplot function. This can generate a QQ plot for you. Use it to generate a plot for df_uniform.x, and df_normal.x. assume your distribution is normal for both (even though we know it is not!) Compare and contrast your resulting plot. Does the output suggest that one is indeed normally distributed, and the other is not?

In [20]:
```
import scipy.stats
scipy.stats.probplot(df_uniform.x, dist = 'norm', plot = plt)
plt.title("QQ of Uniform Distribution")
plt.show()
scipy.stats.probplot(df_normal.x, dist = 'norm', plot = plt)
plt.title("QQ of Normal Distribution")
plt.show()
```





Yes, the graph shows that one is normally distributed and the other is not. We can see in the second graph that the majority of the blue dots are centered around the 45 degree red line, save for a few outliers at both ends. In the first graph it is clear that our data for uniform distribution does not fit into the 45 degree red line, showing that the data is not normally distributed.

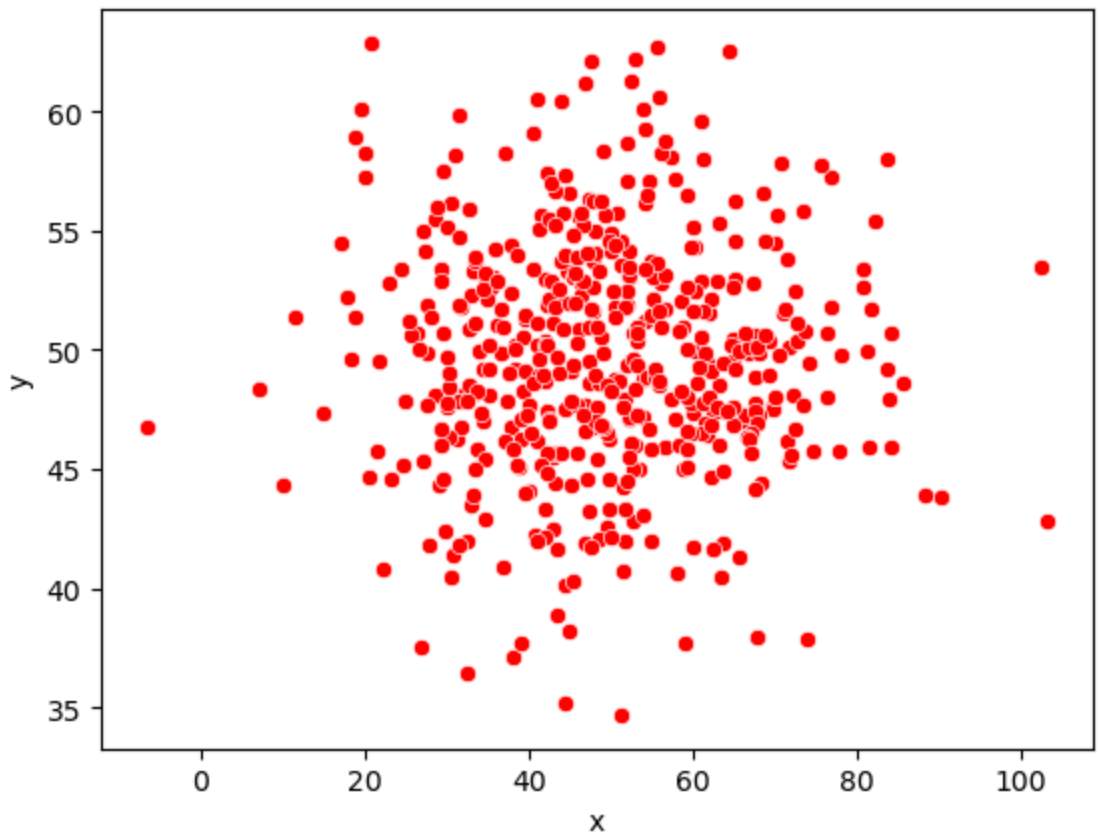
Part II - seaborn

20) [P] Add the following import statement:

In [21]:
```
import seaborn as sns
```

21) [P] Show a single scatterplot of df_normal using sns. Change the default color and point type that is used in the plot.

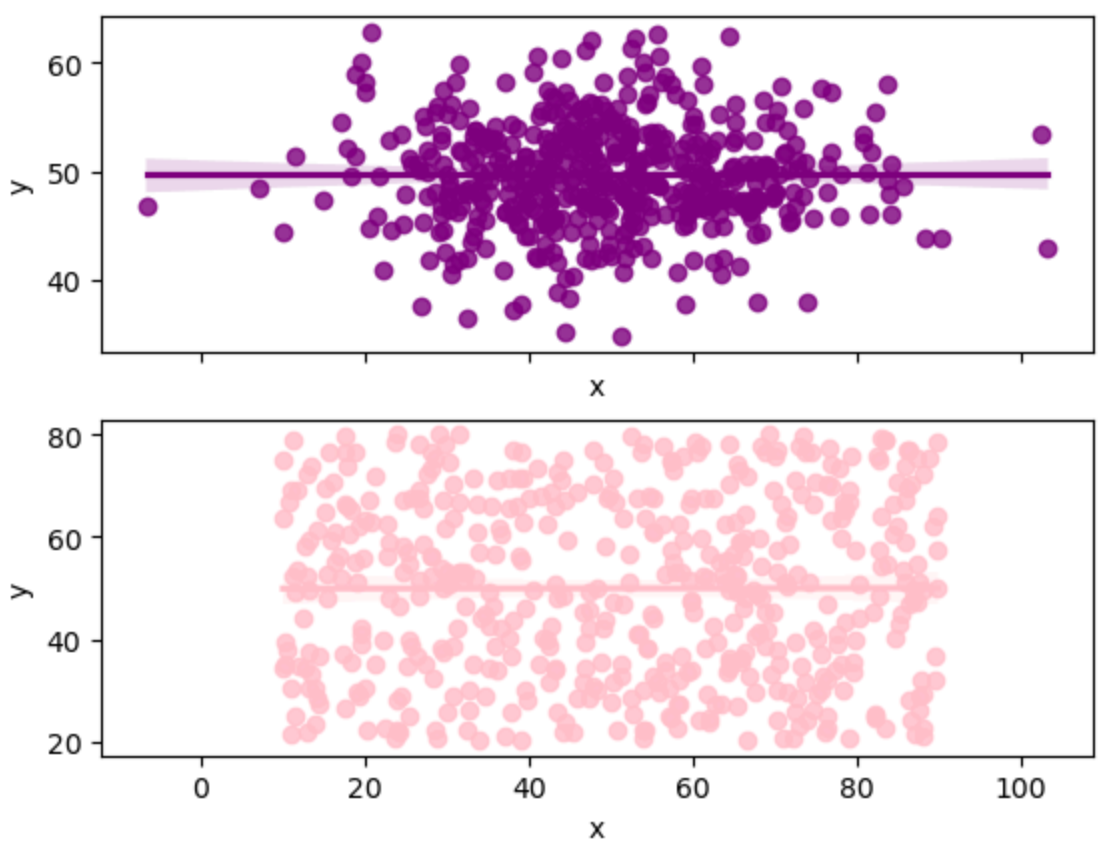
In [22]:
```
sns.scatterplot(df_normal['x'], df_normal['y'], color = 'red', sizes = 24)
plt.show()
```



For the remainder of these exercises, you are required to use seaborn, but select at least two aspects of your plot to make them unique. It could be the color of the point, size, background, grid, etc. etc. There are many choices. Use these exercises to learn about this wonderful visualization framework, and to tap into the artist in you!

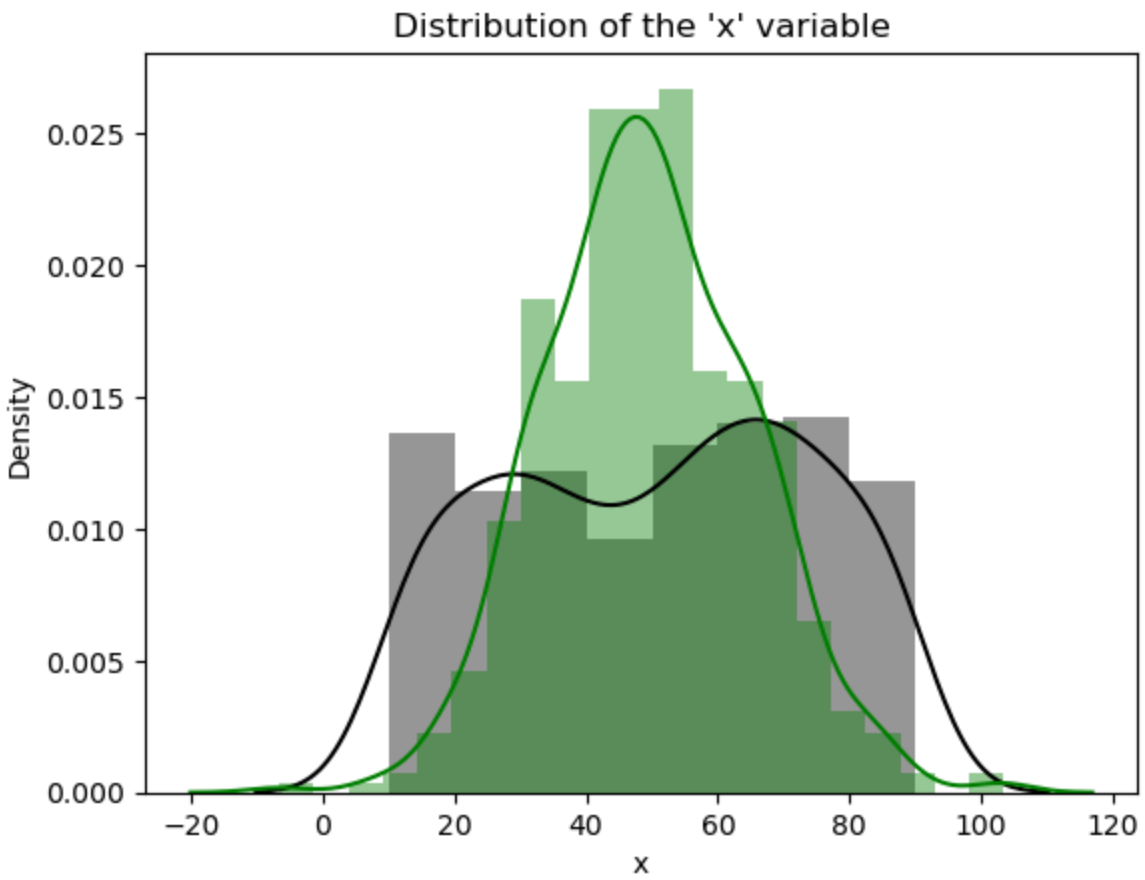
22) [P] Show a scatterplot of both df_uniform and df_normal side by side on the same figure

In [23]:
```
fig, (one, two) = plt.subplots(2, sharex = True)
sns.regplot(x = df_normal['x'], y = df_normal['y'], ax = one, color = 'purple')
sns.regplot(x = df_uniform['x'], y = df_uniform['y'], ax = two, color = 'pink')
plt.show()
```



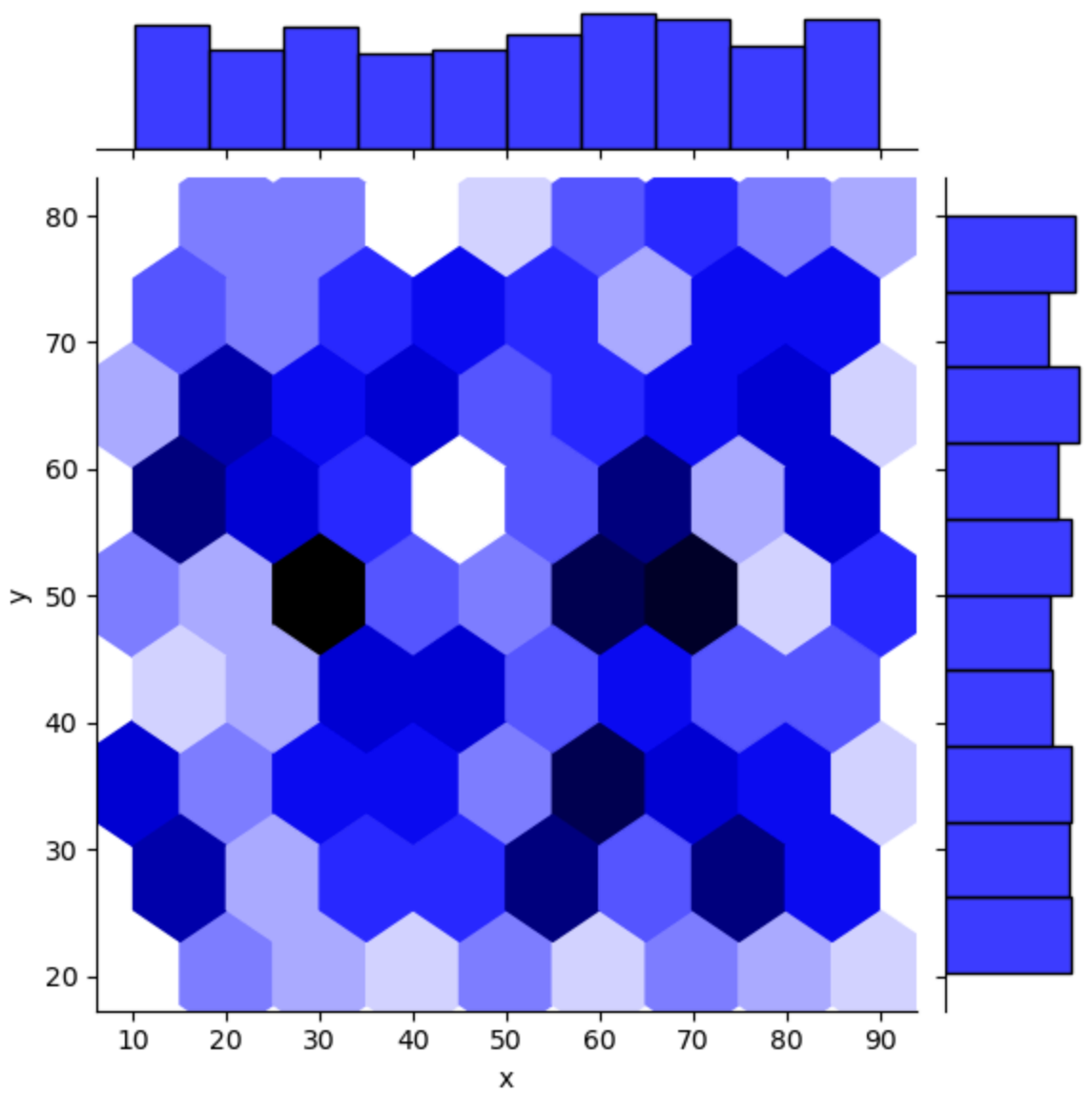
23) [P] Show the distribution of only the x variable for both df_uniform and df_normal, with a density curve and a rugplot at the bottom. (Look at sns.distplot)

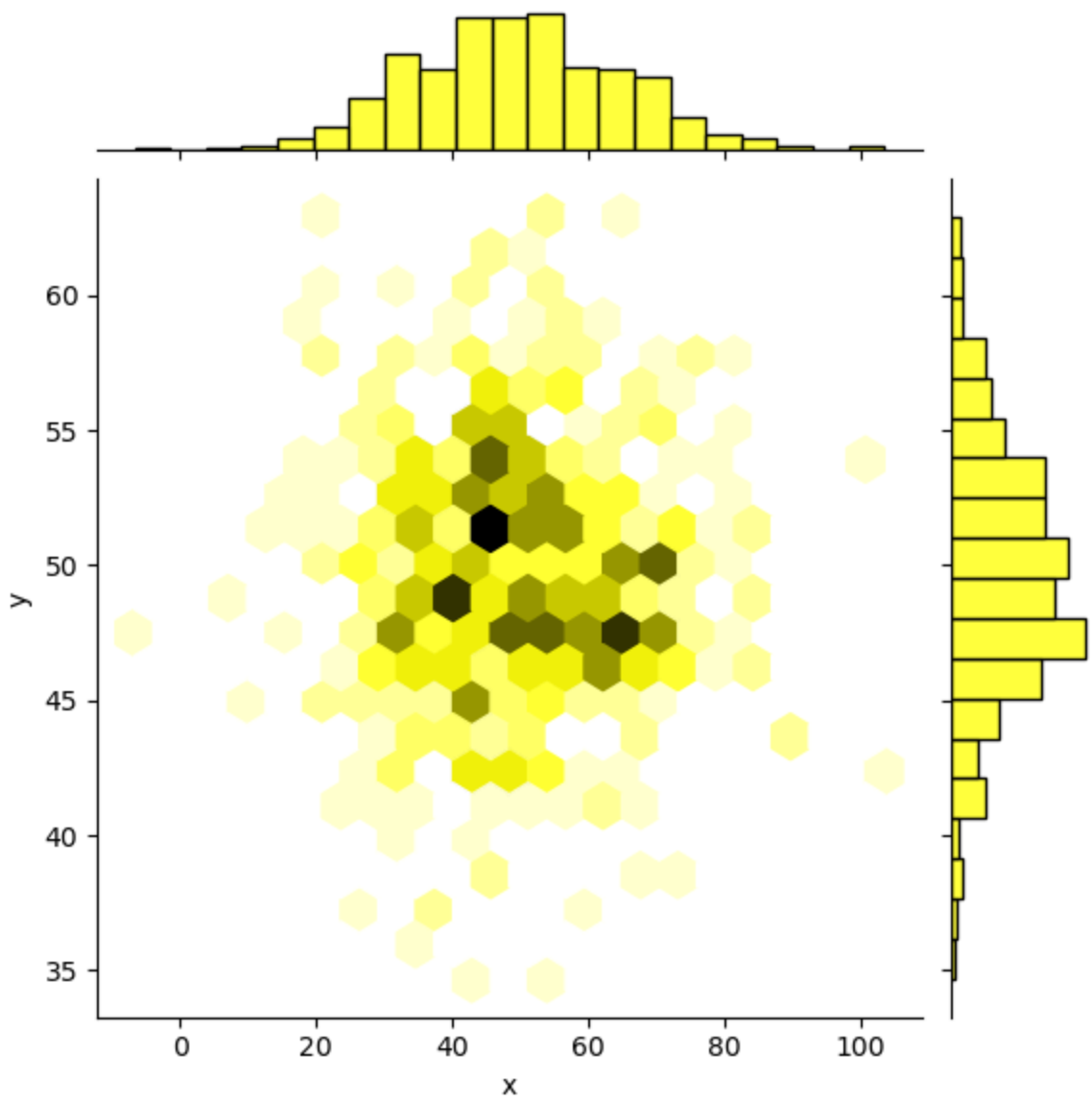
In [24]:
```
sns.distplot(df_uniform['x'], color = 'black').set_title('Distribution of the \'x\' variable')
sns.distplot(df_normal['x'], color = 'green')
plt.show()
```



24) [P] Use sns.jointplot to show the bivariate distribution of x and y for df_uniform and df_normal

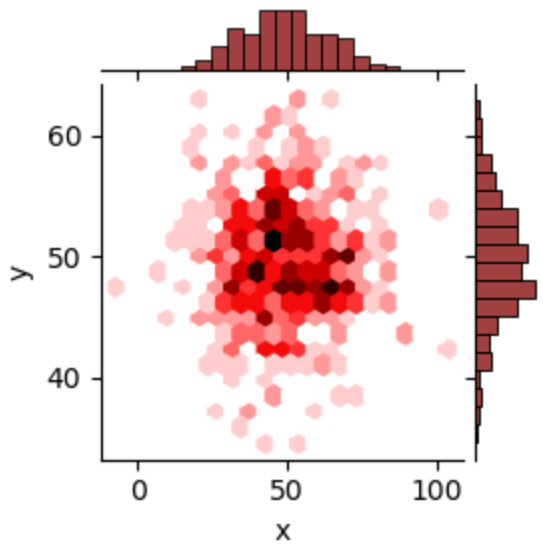
In [25]:
```
jf = sns.jointplot(df_uniform['x'], df_uniform['y'], color = 'blue', kind = 'hex')
sns.jointplot(df_normal['x'], df_normal['y'], color = 'yellow', kind = 'hex')
plt.show()
```





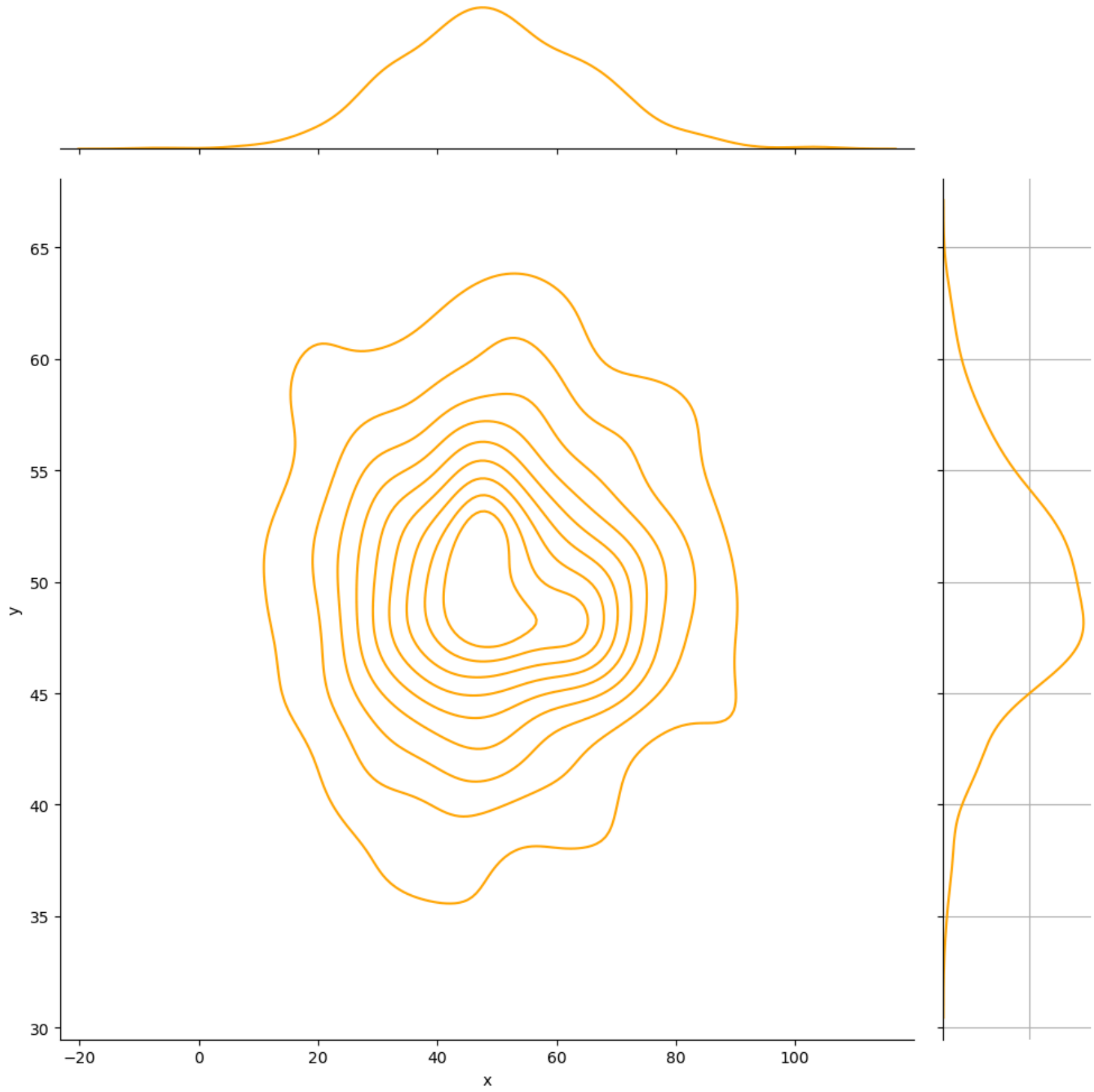
25) [P] Show a hexbin plot using sns.jointplot for df_normal

In [26]:
```
sns.jointplot(df_normal['x'], df_normal['y'], height = 3, color = 'maroon', kind = 'hex')
plt.show()
```



26) [P] Show a kernel density estimation (kde) using sns.jointplot for df_normal

In [27]:
```
sns.jointplot(df_normal['x'], df_normal['y'], height = 10, color = 'orange', kind = 'kde')
plt.grid()
plt.show()
```



Part III – Some basic data preprocessing

27) [P] Create an additional variable in df_uniform called x_fac1 that represents a factor with 3 levels, "X1", "X2", and "X3". You should discretize according to equal width bins over the distribution of x. (Divide the range of x into three.)

In [28]:
```
x_fac1 = pd.qcut(df_uniform['x'], 3, labels = ['x1', 'x2', 'x3'])
df_uniform['x_fac1'] = pd.Series(x_fac1)
df_uniform.head(5)
```

Out[28]:

	x	y	x_fac1
0	17.206520	51.924212	x1
1	58.755818	75.729721	x2
2	31.961267	52.138041	x1
3	43.391695	72.659295	x2
4	77.808930	73.294549	x3

28) [P] Create an additional variable in df_uniform called x_fac2 that represents a factor with 3 levels, "X1", "X2", and "X3". This time, you should discretize using equal depth bins over the distribution of x. Select your division criteria such that there are an equal number of data in each bin. Verify that the distribution of your data each has the same number of data (within 1).

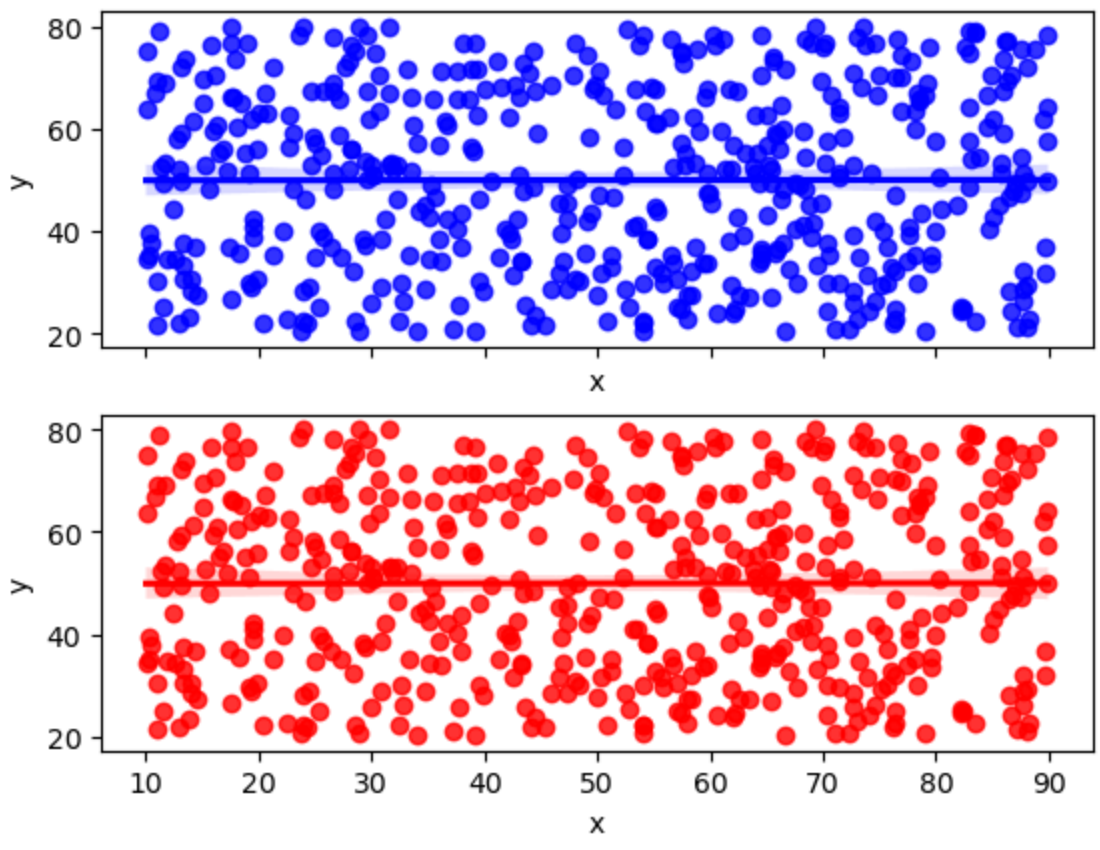
In [29]:
```
bins = [df_uniform['x'].min(), df_uniform['x'].max()/3, 2*df_uniform['x'].max()/3, df_uniform['x'].max()]
df_uniform['x_fac2'] = pd.cut(df_uniform['x'], bins, labels = ['x1', 'x2', 'x3'])
df_uniform.head(5)
```

Out[29]:

	x	y	x_fac1	x_fac2
0	17.206520	51.924212	x1	x1
1	58.755818	75.729721	x2	x2
2	31.961267	52.138041	x1	x2
3	43.391695	72.659295	x2	x2
4	77.808930	73.294549	x3	x3

29) [P] Create a side by side scatter plot showing the distribution of df_uniform, using x_fac1 as the color for one plot, and x_fac2 as the color for your other plot.

In [30]:
```
fig, (one, two) = plt.subplots(2, sharex = True)
sns.regplot(x = df_uniform['x'], y = df_uniform['y'], ax = one, color = 'blue')
sns.regplot(x = df_uniform['x'], y = df_uniform['y'], ax = two, color = 'red')
plt.show()
```



Credit- #5 was influenced by: https://matplotlib.org/examples/pylab_examples/subplots_demo.html