Exercise for Tutorial 14

linke to exercise:
https://github.com/kzapalac/PythonTutorials/blob/main/tutorial_ex_14/tutorial014Exercise.ipynb

Learn how to understand and describe code with words!

So far we have read text that describes code. We have looked at code and tried to understand what it does, with the help of a verbal description (in most cases).

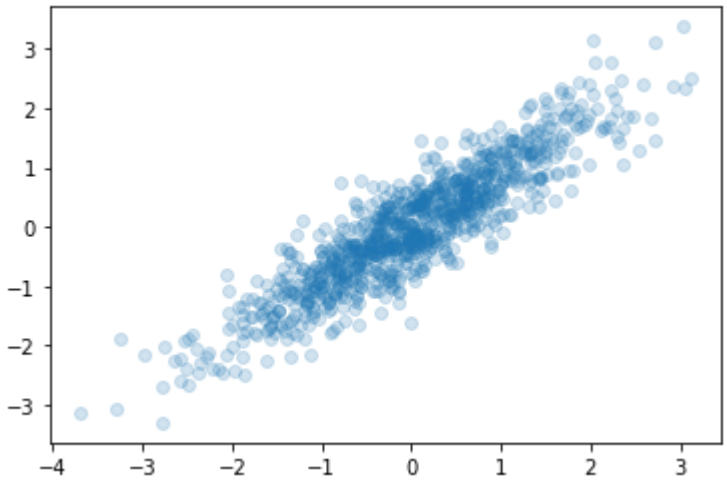
Here after, you will be asked to look at a block of code (that you might or might have not seen before) and describe in words what the code is doing.

You are welcome to look at the documentation of the methods, functions and libraries used but it would be asked to not use Google the answer in this case (besides using Google to find the documentation of the function on the python libraries web pages).

In [26]:
```
import numpy as np
from  numpy.random import multivariate_normal
import matplotlib.pyplot as plt
import matplotlib as mpl

# defining some parameters
mu1 = 0;   # the means
mu2 = 0;
var1 = 1;  # the variances
var2 = 1;
cov = 0.9; # the relationships between the datasets - the covariance
cov_m = [[var1, cov], # make the covariance matrix
         [cov, var2]]
# now make the actual 2x100 data array
data = multivariate_normal([mu1, mu2], cov_m, size=1000)
plt.scatter(data[:,0], data[:,1], alpha=0.2) # and plot the data!
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x1acb7d63070>
```



Explain in your words what the code is doing. Do that by commenting each line of the code. The

description should cober each parameter (say the variables initialized at the beginning) but also each operation in the code, for example the calls to a function should be described as well and the indexing operations. For the indexing please make sure to be explicit about which dimensions of the array are being addressed and also report the full dimensions of the array as you describe the indexing operation. We expect 1-3 pragraphs of description max.

In the code above, I imported the libraries that I needed to create a a correlated dataset and to plot it. Then, I defined the means and variances of the 2 datasets. The mean was equal to 0 and the variance was equal to 1 for both datasets. I also defined the covariance, which represents the the relationship between the datasets. The covariance was equal to 0.9, which means that the datasets have a positive relationship with eachother. The last preliminary step before creating the datasets was to create a matrix containing the variance and covariance for each dataset.

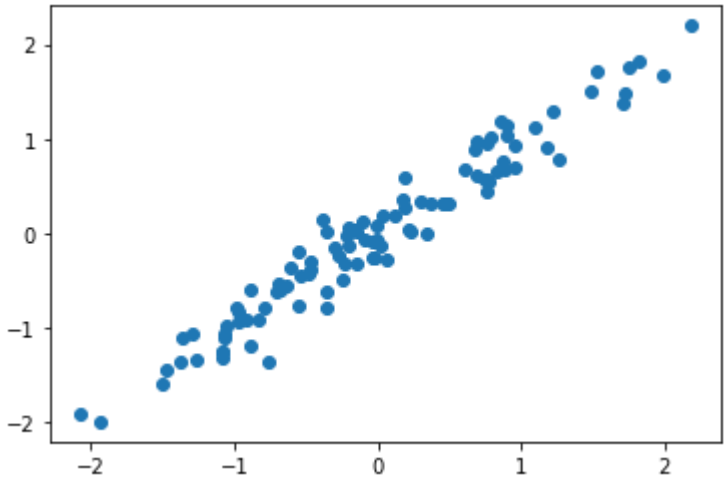
After completing all the preliminaries, I created the 2 datasets using the function multivariate_normal, and named the dataset "data". For the first part of this function, I wrote both of the means in brackets, then I wrote the name for the covariance matrix, and made the datasets 1000 observations long. Lastly, I plotted the first column of the data and the second column of the data, and I made the points fainter.

Create a series of correlated datasets using `for loops`

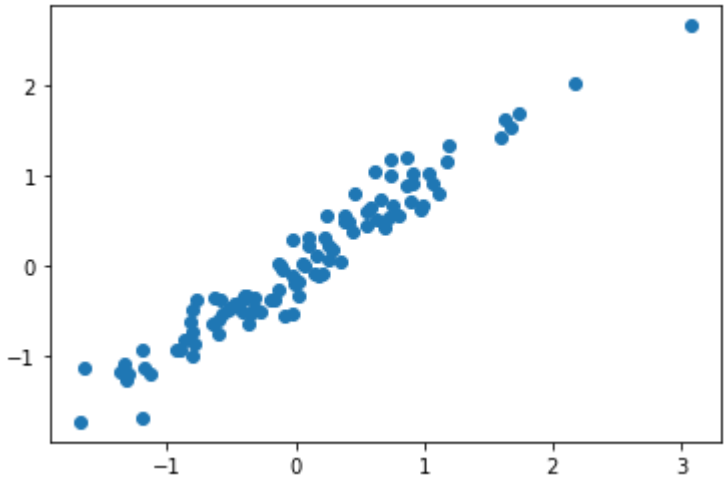
The code below, shows how to create and plot a series of correlated datasets.

In [2]:

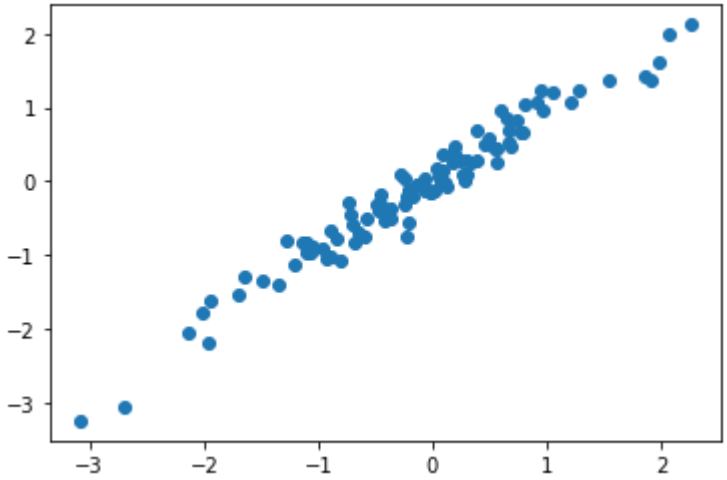
```
counter = 0;
m = 5; #number of datasets
n = 100;
scaling = 0.2
while counter < m :
    y = np.random.randn(n,1)
    x = y + scaling*np.random.randn(n,1)
    plt.scatter(x, y)
    plt.show()
    print("We are plotting because we are INSIDE the while loop.")
    plt.pause(0.05)
    counter = counter + 1
else:
    print("We are NOT plotting because we are OUTSIDE the while loop.")
```



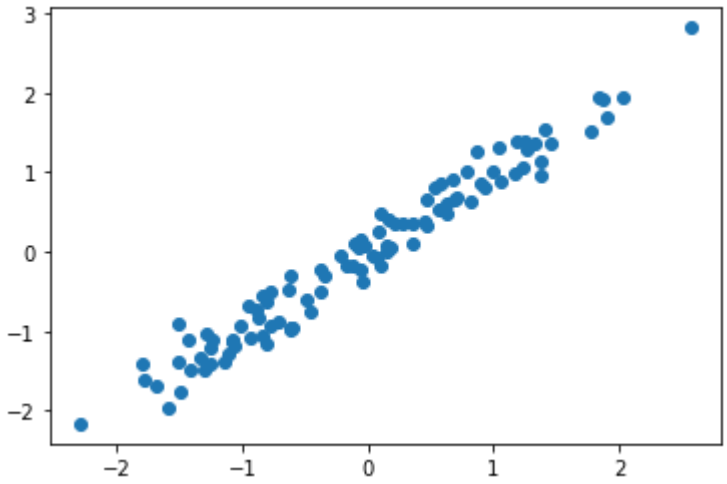
We are plotting because we are INSIDE the while loop.



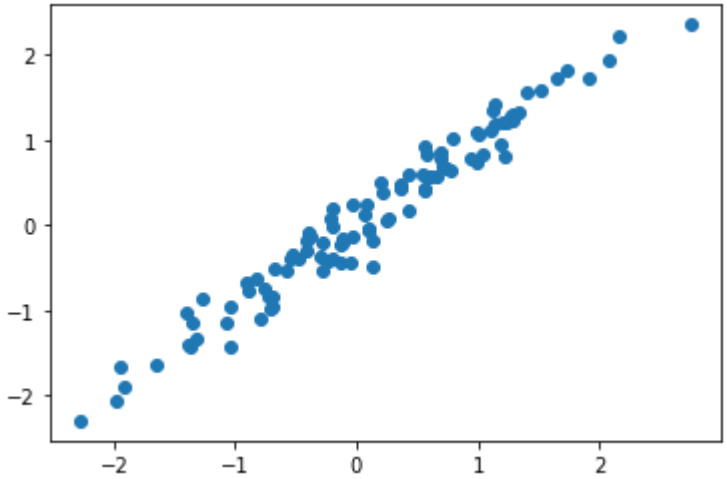
We are plotting because we are INSIDE the while loop.



We are plotting because we are INSIDE the while loop.



```
We are plotting because we are INSIDE the while loop.
```

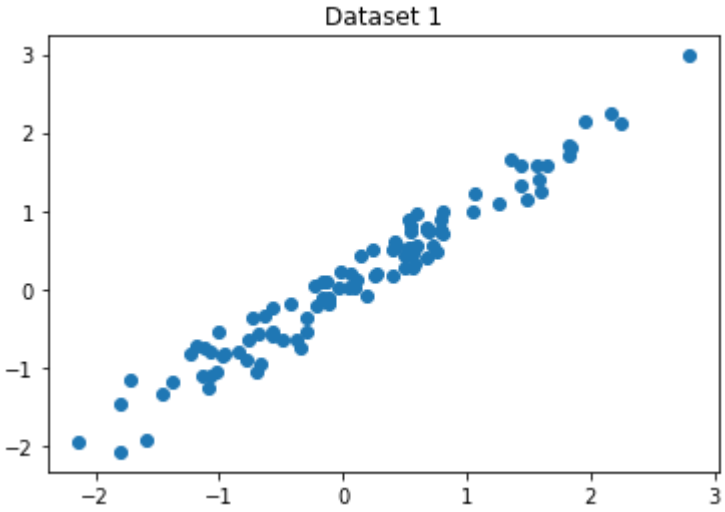


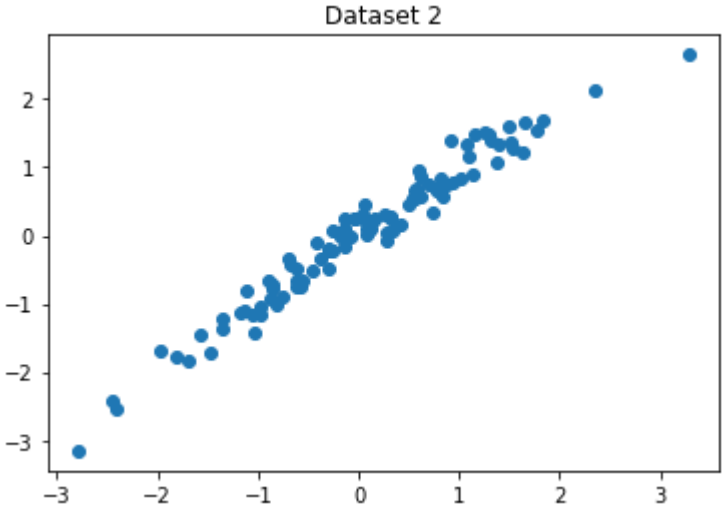
```
We are plotting because we are INSIDE the while loop.
We are NOT plotting because we are OUTSIDE the while loop.
```

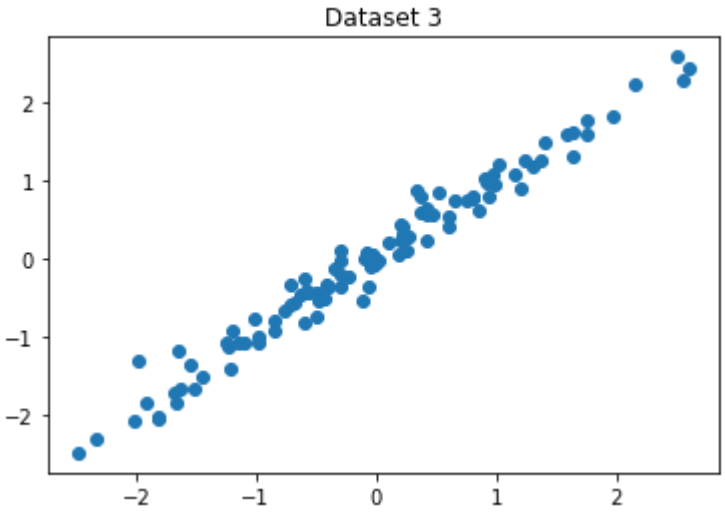
You are tasked to write similar code (that generates and plots correlated datasets). But, your code should use a `for` loop instead of the `while` loop shown in the example.

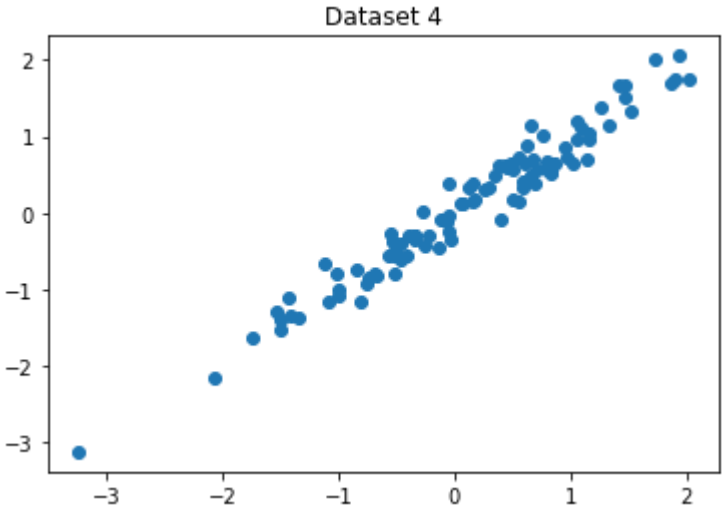
In [3]:

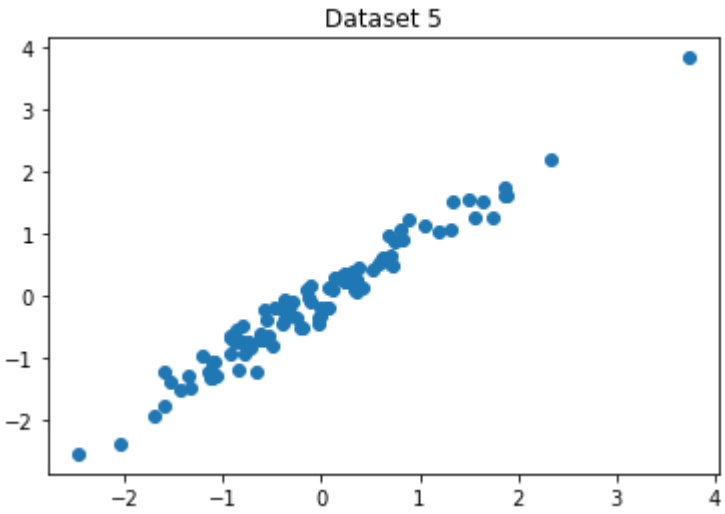
```
m = 5; #number of datasets
n = 100; # 100 observations in each dataset
scaling = 0.2 # decreasing the variance of the dataset
titles = ['Dataset 1', 'Dataset 2', 'Dataset 3', 'Dataset 4', 'Dataset 5'] # titles for
for i in range(m) :
    y = np.random.randn(n,1)
    x = y + scaling*np.random.randn(n,1)
    plt.scatter(x, y)
    plt.title(titles[i])
    plt.show()
```


Dataset 1
3
2
1
0
−1
−2
−2
−1
0
1
2
3


Dataset 2
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Dataset 3
2
1
0
−1
−2
−2
−1
0
1
2





Make pretty plots

A majority of data science tasks, will not be a simple plug and play of previously learned tools and code snippet. Instead, they will require learning new skills on the job. Here we ask you to make a few plots based on the previously learned functions. Yet, we ask that you go a little beyond that, by finding ways to improve the visuals of the plots. You can do this by learning about the functionality of `scatter` and `plot` online.

What does it mean to improve the visuals of the plots? Improving the visuals means that the plots look simple, slick, elegant the colors are not the default but are personalized and well chosen. You are free to do the customization of the plots as you prefer, have fun with it and see what comes out of the fun!

This is the plot we ask you to do: Use plt.subplot to make

- a 3 x 3 array of scatter plots of negatively correlated datasets
- the correlated datasets should have
  - Dataset 1: mu = 1, variance = 1
  - Dataset 2: mu = 1, variance = 3
  - covariances should be between 0.3 and 0.9

- Make the plots as pretty as possible, for example:
  - change the colors
  - Removing any superfluous visuals, such as boxes aroung the axis (good plots only have two axis not 4!)
  - Add labels
  - Add titles
  - etc

In [4]:
```
# defining some parameters
datasets = 9
mu1 = 1
mu2 = 1
var1 = 1
var2 = 3
steps = (0.9 - 0.3)/9
cov = np.arange(-0.9, -0.3, steps)
cov2d = np.reshape(cov,(3,3))

print(cov2d)
```

```
[[-0.9        -0.83333333 -0.76666667]
 [-0.7        -0.63333333 -0.56666667]
 [-0.5        -0.43333333 -0.36666667]]
```

In [22]:
```
axs_titles = np.zeros((3,3)) # tested out creating an array to hold title for each plot
for i in range(3):
    for j in range(3):
        axs_titles[i,j] = print('Covariance of',cov2d[i,j])
axs_titles
# well that didn't work
```

```
Covariance of -0.9
Covariance of -0.8333333333333334
Covariance of -0.7666666666666667
Covariance of -0.7000000000000001
Covariance of -0.6333333333333334
Covariance of -0.5666666666666668
Covariance of -0.5000000000000001
Covariance of -0.43333333333333346
Covariance of -0.3666666666666668
```

Out[22]:
```
array([[nan, nan, nan],
       [nan, nan, nan],
       [nan, nan, nan]])
```

In [27]:
```
mpl.style.available # gives me all the styles I can use
```

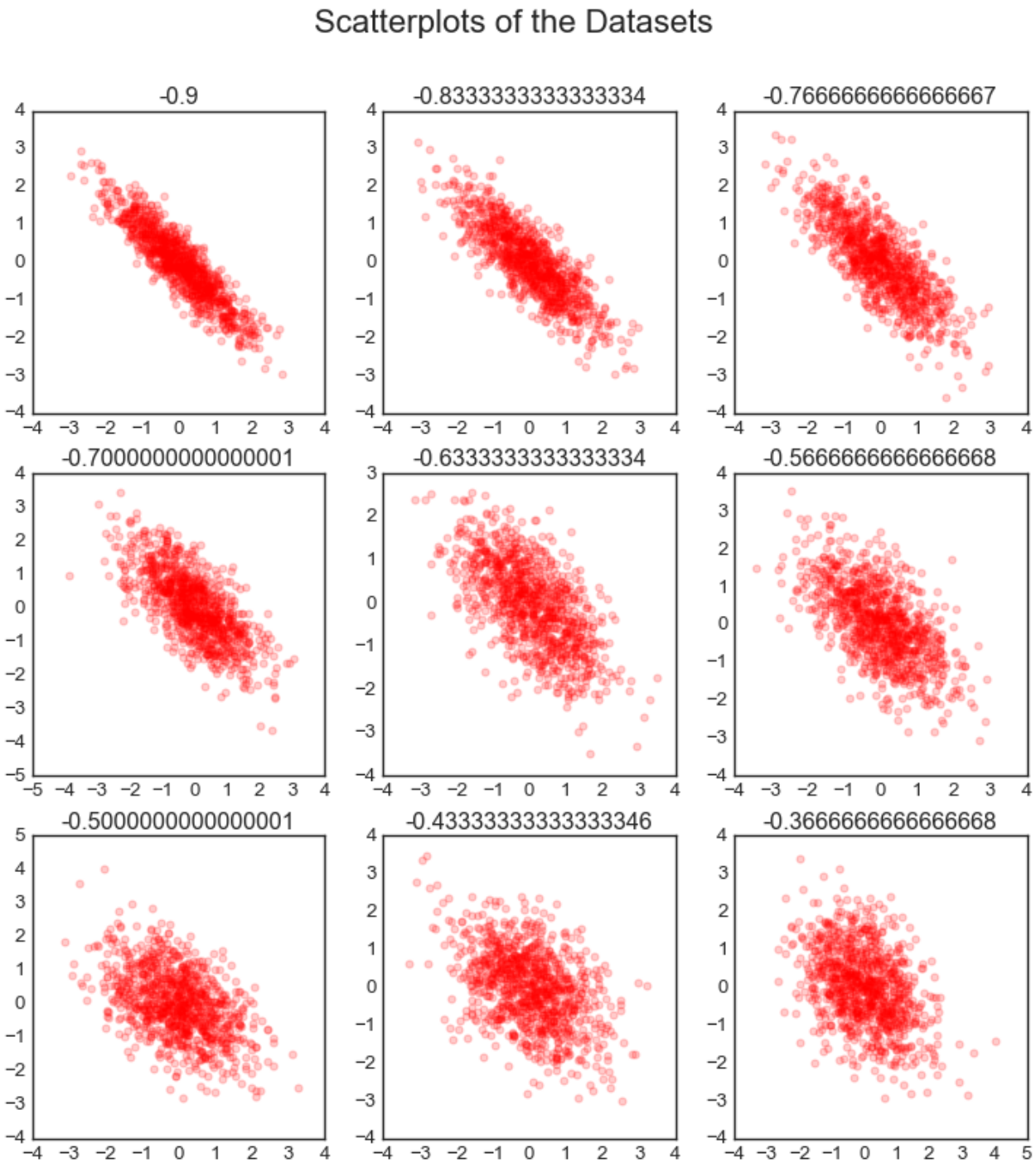
Out[27]:
```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
```

```
'seaborn',
'seaborn-bright',
'seaborn-colorblind',
'seaborn-dark',
'seaborn-dark-palette',
'seaborn-darkgrid',
'seaborn-deep',
'seaborn-muted',
'seaborn-notebook',
'seaborn-paper',
'seaborn-pastel',
'seaborn-poster',
'seaborn-talk',
'seaborn-ticks',
'seaborn-white',
'seaborn-whitegrid',
'tableau-colorblind10']
```

In [52]:

```
mpl.style.use('seaborn-white') # sets the style of the plots below

fig, axs = plt.subplots(3,3, figsize=(11,11)) # here I create 9 subplots and set the fi
for i in range(3) :
    for j in range(3) :
        cov_m = [[var1, cov2d[i,j]], [cov2d[i,j], var2]]  # creating a new matrix for e
        data = multivariate_normal([mu1, mu2], cov_m, size = 1000) # creating the datas
        axs[i,j].scatter(data[:,0], data[:,1], alpha=0.2, color='r') # setting the axis
        axs[i,j].set_title(cov2d[i,j]) # sets the title for each individual plot as the
        fig.suptitle("Scatterplots of the Datasets", fontsize=20) # sets the main title
```


Scatterplots of the Datasets
-0.9
-0.8333333333333334
-0.7666666666666667
-0.7000000000000001
-0.6333333333333334
-0.5666666666666668
-0.5000000000000001
-0.43333333333333346
-0.3666666666666668

In [ ]: