# index

March 5, 2022

## 1 Generating Data - Lab

### 1.1 Introduction

In this lab, we shall practice some of the data generation techniques that we saw in the previous lesson in order to generate datasets for regression and classification purposes. We will run a couple of simple simulations to help us generate different datasets by controlling noise and variance parameters in the data generation process. We will also look at the statistical indicators and visual output to see how these parameters affect the accuracy of an algorithm.

### 1.2 Objectives

In this lab you will:

- Generate datasets for classification problems
- Generate datasets for regression problems

### 1.3 Generate data for classfication

Use `make_blobs()` to create a binary classification dataset with 100 samples, 2 features, and 2 centers (where each center corresponds to a different class label). Set `random_state = 42` for reproducibility.

*Hint: Here's a link to the documentation for* `make_blobs()`.

[12]:
```
# Your code here
from sklearn.datasets import make_blobs
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

X, y = make_blobs(n_samples=100,n_features=2,
                  centers=2,random_state = 42)
```

Place the data in a `pandas` DataFrame called `df`, and inspect the first five rows of the data.

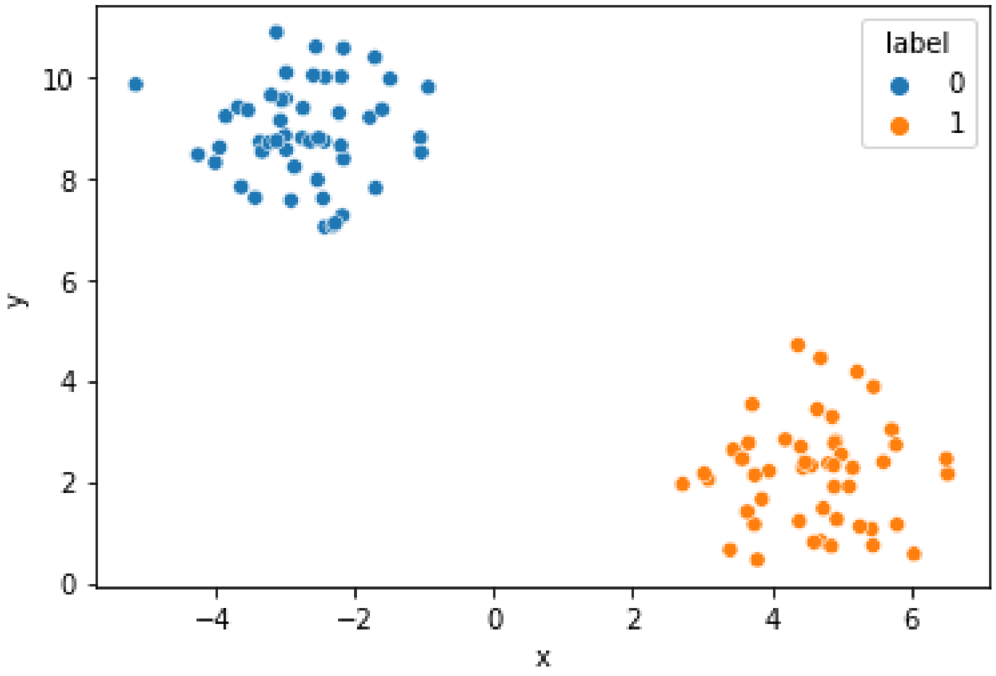
*Hint: Your dataframe should have three columns in total, two for the features and one for the class label.*

[17]:
```
# Your code here
df = pd.DataFrame(dict(x = X[:,0], y= X[:,1], label = y))
```

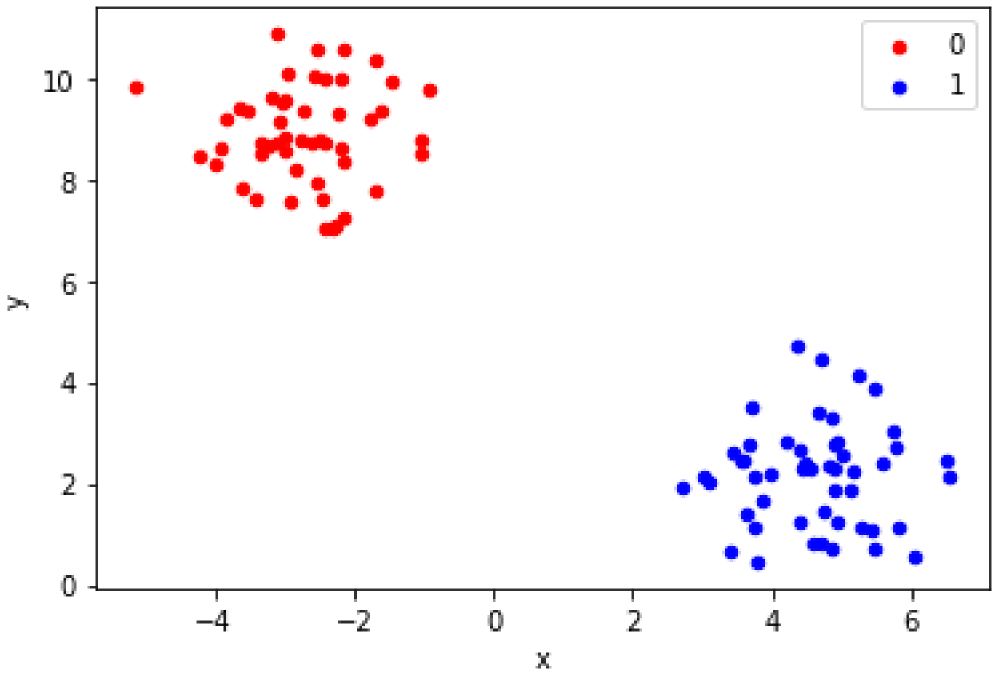
Create a scatter plot of the data, while color-coding the different classes.

*Hint: You may find this dictionary mapping class labels to colors useful:* `colors = {0: 'red', 1: 'blue'}`

[22]:
```
# Your code here
sns.scatterplot(data=df, x=df["x"], y=df["y"], hue=df["label"]);
```



[23]:
```
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = plt.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key,␣
 ↪color=colors[key])
plt.show()
```



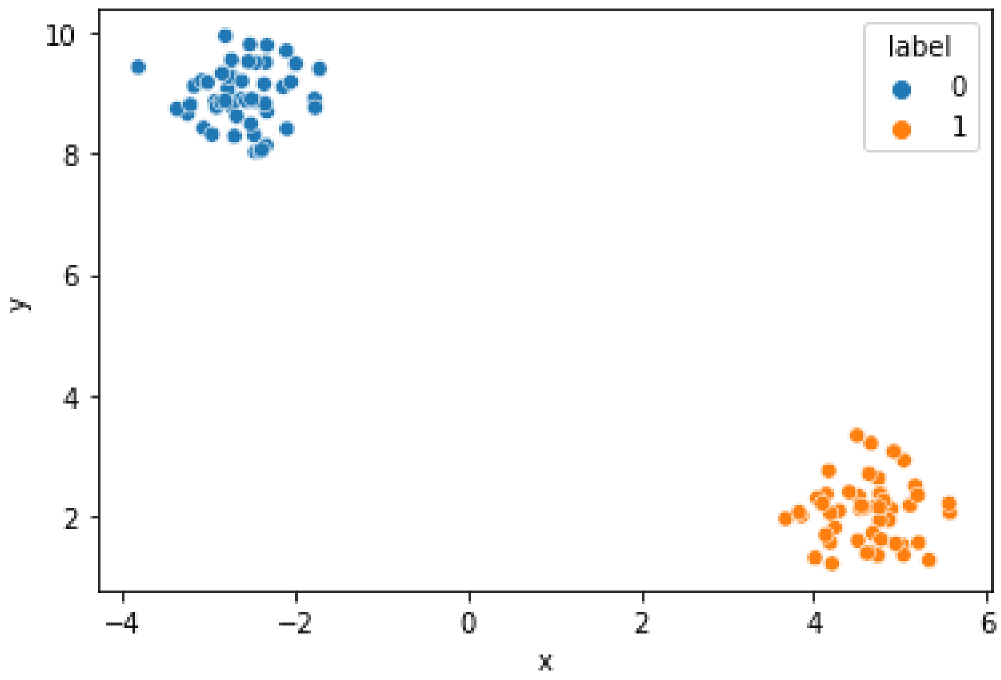
Repeat this exercise two times by setting `cluster_std = 0.5` and `cluster_std = 2`.

Keep all other parameters passed to `make_blobs()` equal.

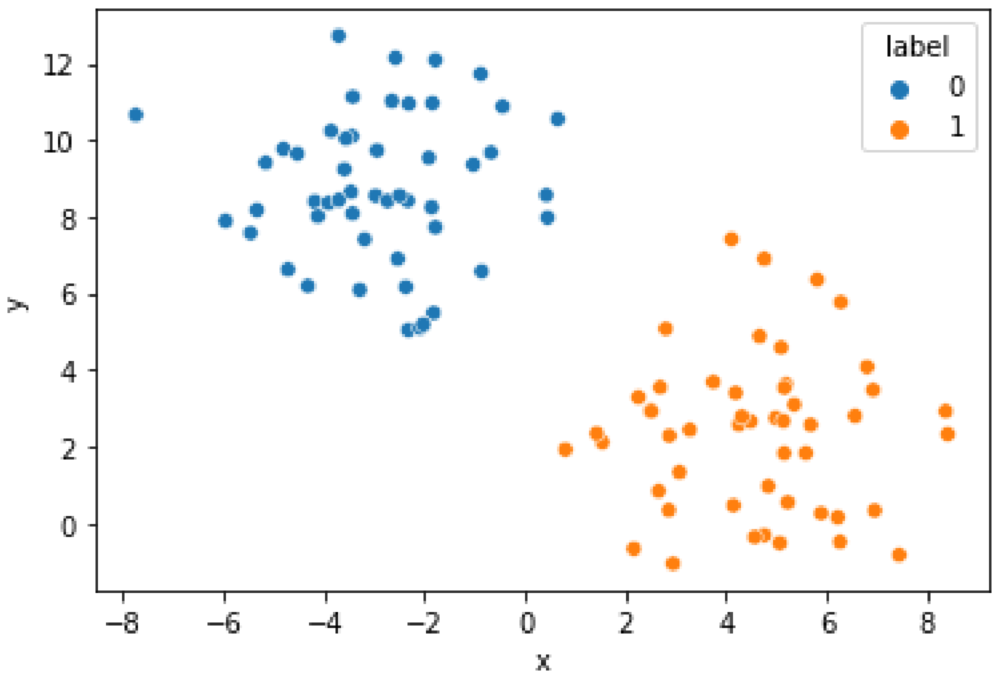
That is: * Create a classification dataset with 100 samples, 2 features, and 2 centers using `make_blobs()` * Set `random_state = 42` for reproducibility, and pass the appropriate value for `cluster_std`
* Place the data in a `pandas` DataFrame called `df`
* Plot the values on a scatter plot, while color-coding the different classes

What is the effect of changing `cluster_std` based on your plots?

[27]:
```
# Your code here:
# cluster_std = 0.5
X, y = make_blobs(n_samples=100,n_features=2,
                  centers=2,random_state = 42, cluster_std=0.5)

df = pd.DataFrame(dict(x = X[:,0], y= X[:,1], label = y))
sns.scatterplot(data=df, x=df["x"], y=df["y"], hue=df["label"]);
```



[28]:
```
# Your code here:
# clusted_std = 2
X, y = make_blobs(n_samples=100,n_features=2,
                  centers=2,random_state = 42, cluster_std=2)

df = pd.DataFrame(dict(x = X[:,0], y= X[:,1], label = y))
sns.scatterplot(data=df, x=df["x"], y=df["y"], hue=df["label"]);
```



```
[ ]: # Your comments here
```

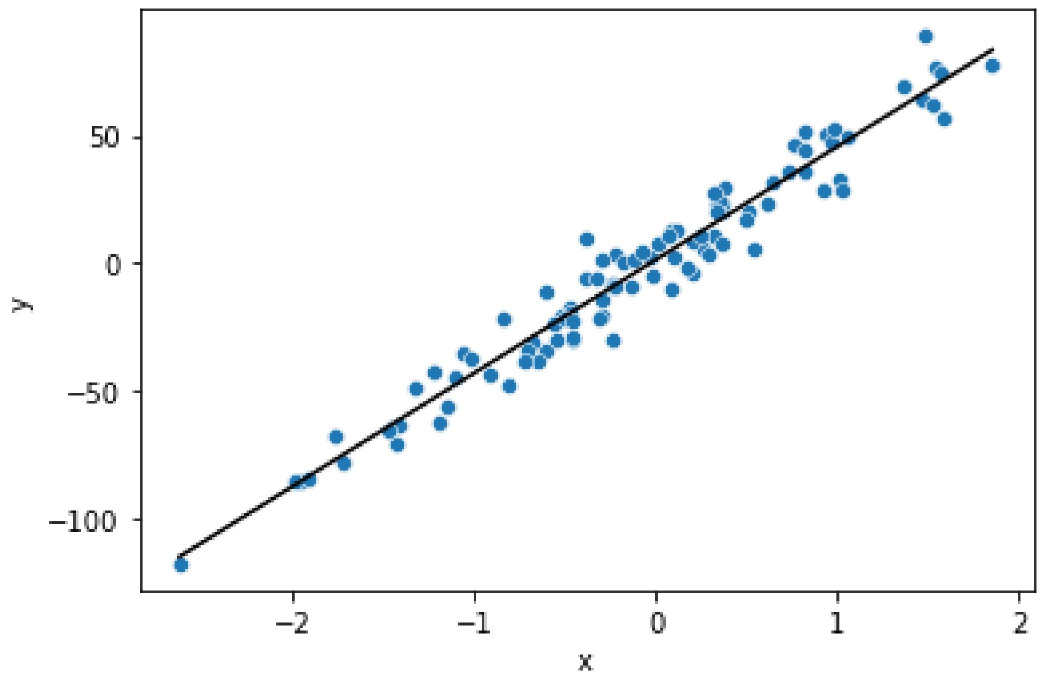
### 1.4 Generate data for regression

Create a function `reg_simulation()` to run a regression simulation creating a number of datasets with the `make_regression()` data generation function. Perform the following tasks:

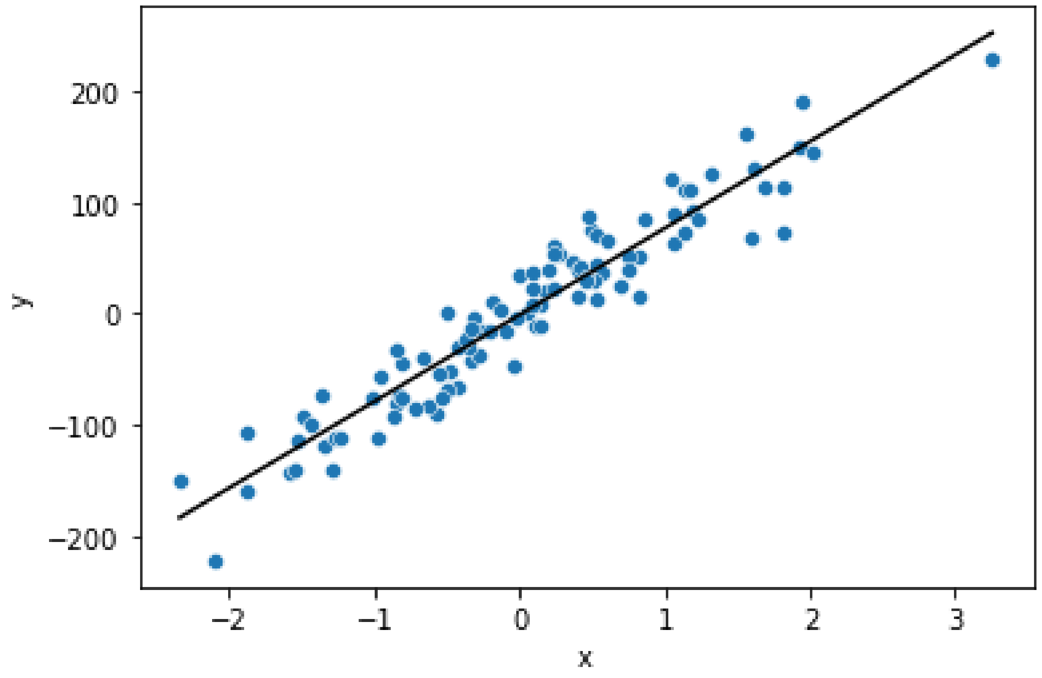
- Create `reg_simulation()` with `n` (noise) and `random_state` as input parameters
  - Make a regression dataset (X, y) with 100 samples using a given noise value and random state
  - Plot the data as a scatter plot
  - Calculate and plot a regression line on the plot and calculate $R^2$ (you can do this in `statsmodels` or `sklearn`)
  - Label the plot with the noise value and the calculated $R^2$
- Pass a fixed random state and values from `[10, 25, 40, 50, 100, 200]` as noise values iteratively to the function above
- Inspect and comment on the output

```
[69]: # Import necessary libraries
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

import pandas as pd
import numpy as np
```

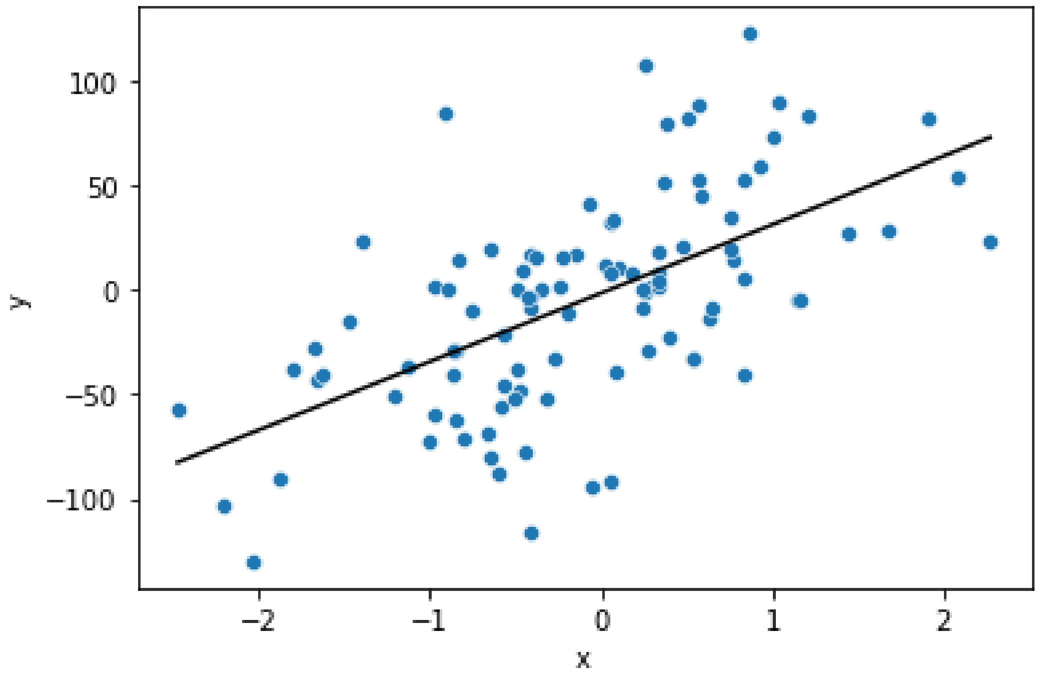
```python
def reg_simulation(n, random_state):

    # Generate X and y
    X, y = make_regression(n_samples = 100, n_features = 1, noise = n,
                           random_state = random_state)
    # Use X,y to draw a scatter plot
    df = pd.DataFrame(dict(x = X[:,0], y = y))

    # Fit a linear regression model to X , y and calculate r2
    # label and plot the regression line
    model = LinearRegression()
    model.fit(df[["x"]], df[["y"]])
    predict = model.predict(df[["x"]])
    sns.scatterplot(data = df, x = df["x"], y = df["y"])
    df2 = pd.DataFrame(dict(x = X[:,0], y = predict[:,0]))
#     sns.scatterplot(data = df2, x = df2["x"], y = df2["y"])
    sns.lineplot(x = df2["x"], y = df2["y"], color = "black")
    plt.show()

    r2_1 = model.score(df[["x"]], df[["y"]])
    r2_2 = r2_score(df[["y"]], predict)
    print("R2 is:", r2_1)
    print("Another R2 is:", r2_2)




random_state = random_state = np.random.RandomState(42)

for n in [10, 25, 40, 50, 100, 200]:
    reg_simulation(n, random_state)
```



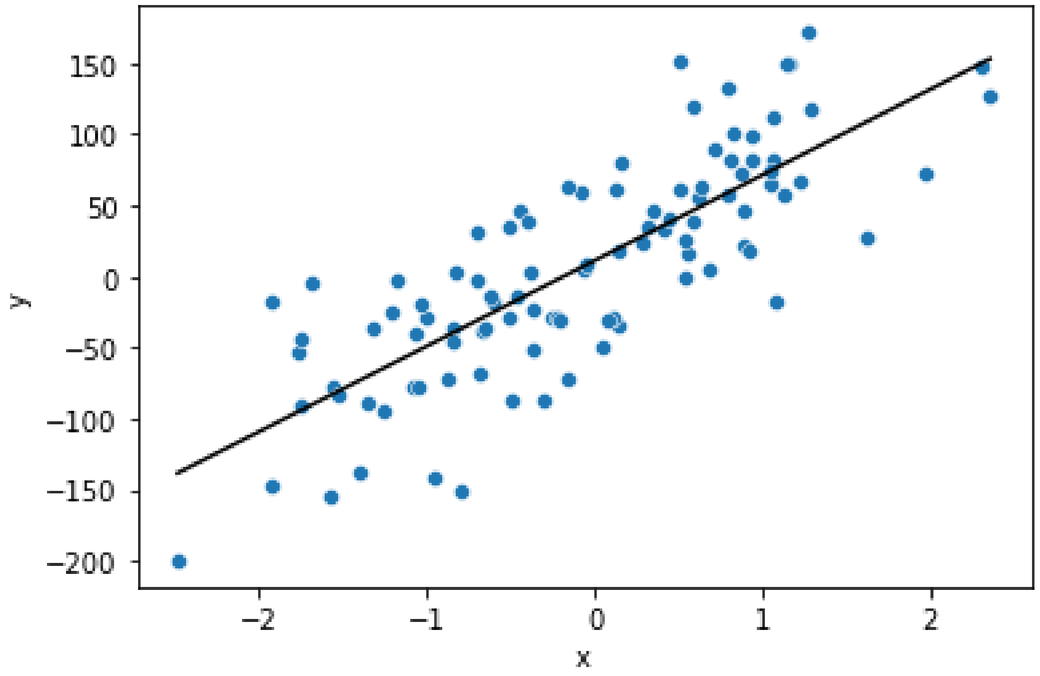
```
R2 is: 0.9538252459635717
Another R2 is: 0.9538252459635717
```



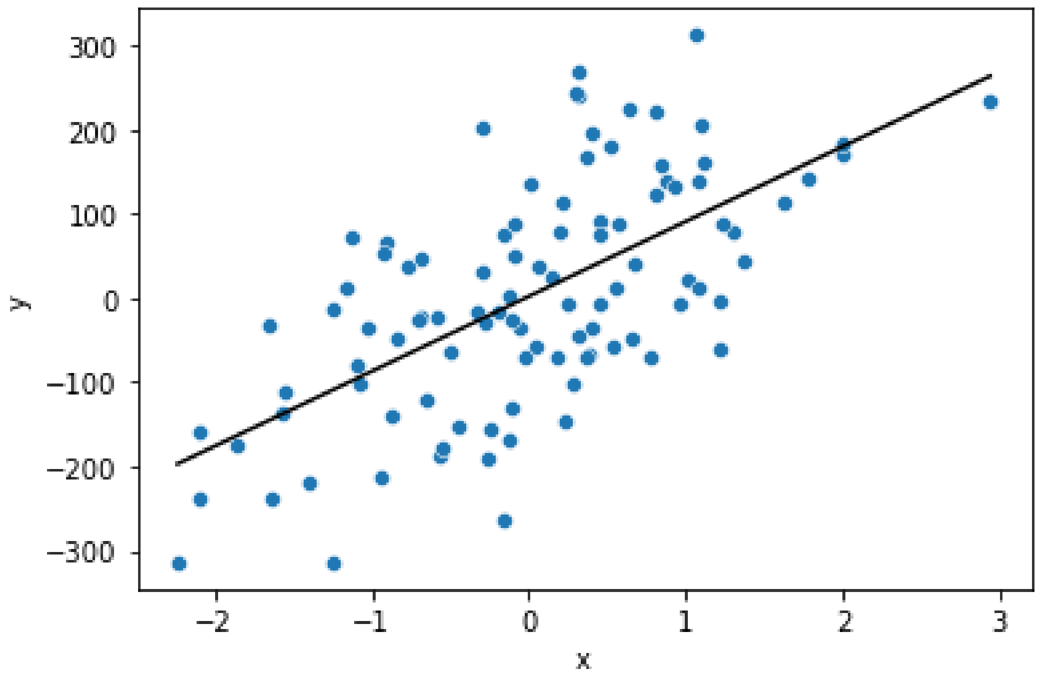
```
R2 is: 0.9106844757337625
Another R2 is: 0.9106844757337625
```



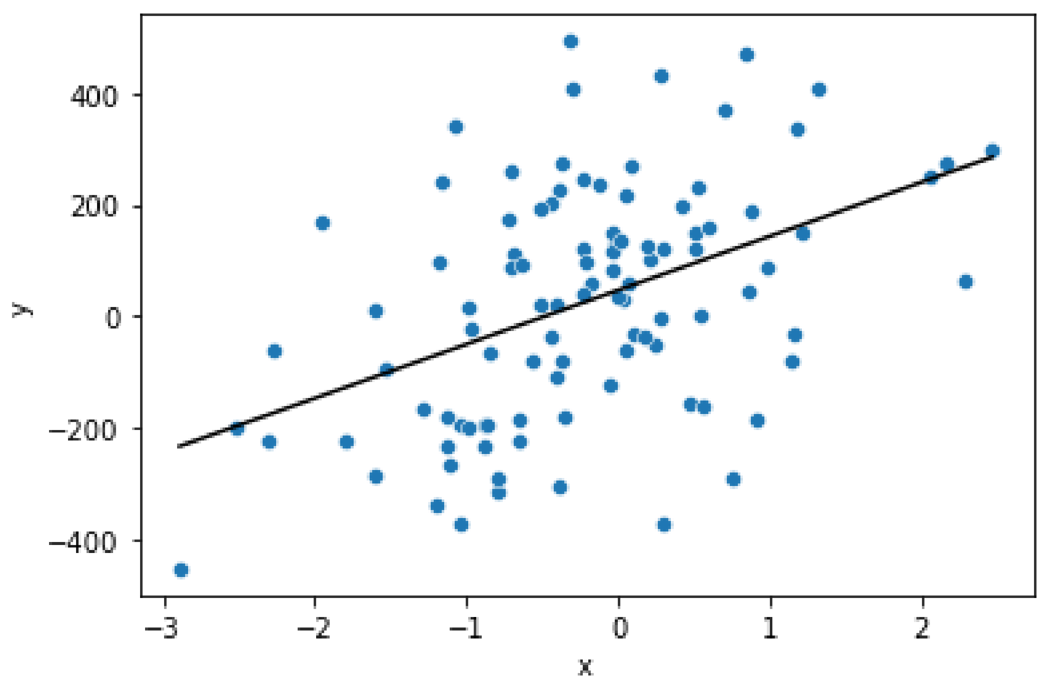
```
R2 is: 0.3547445830324881
Another R2 is: 0.3547445830324881
```



```
R2 is: 0.6401426264407646
Another R2 is: 0.6401426264407646
```



```
R2 is: 0.404121422526033
Another R2 is: 0.404121422526033
```



```
R2 is: 0.2042951311059742
Another R2 is: 0.2042951311059742
```

[ ]:
```
# Your comments here
```

### 1.5 Summary

In this lesson, we learned how to generate random datasets for classification and regression problems. We ran simulations for this and fitted simple models to view the effect of random data parameters including noise level and standard deviation on the performance of parameters, visually as well as objectively. These skills will come in handy while testing model performance and robustness in the future.