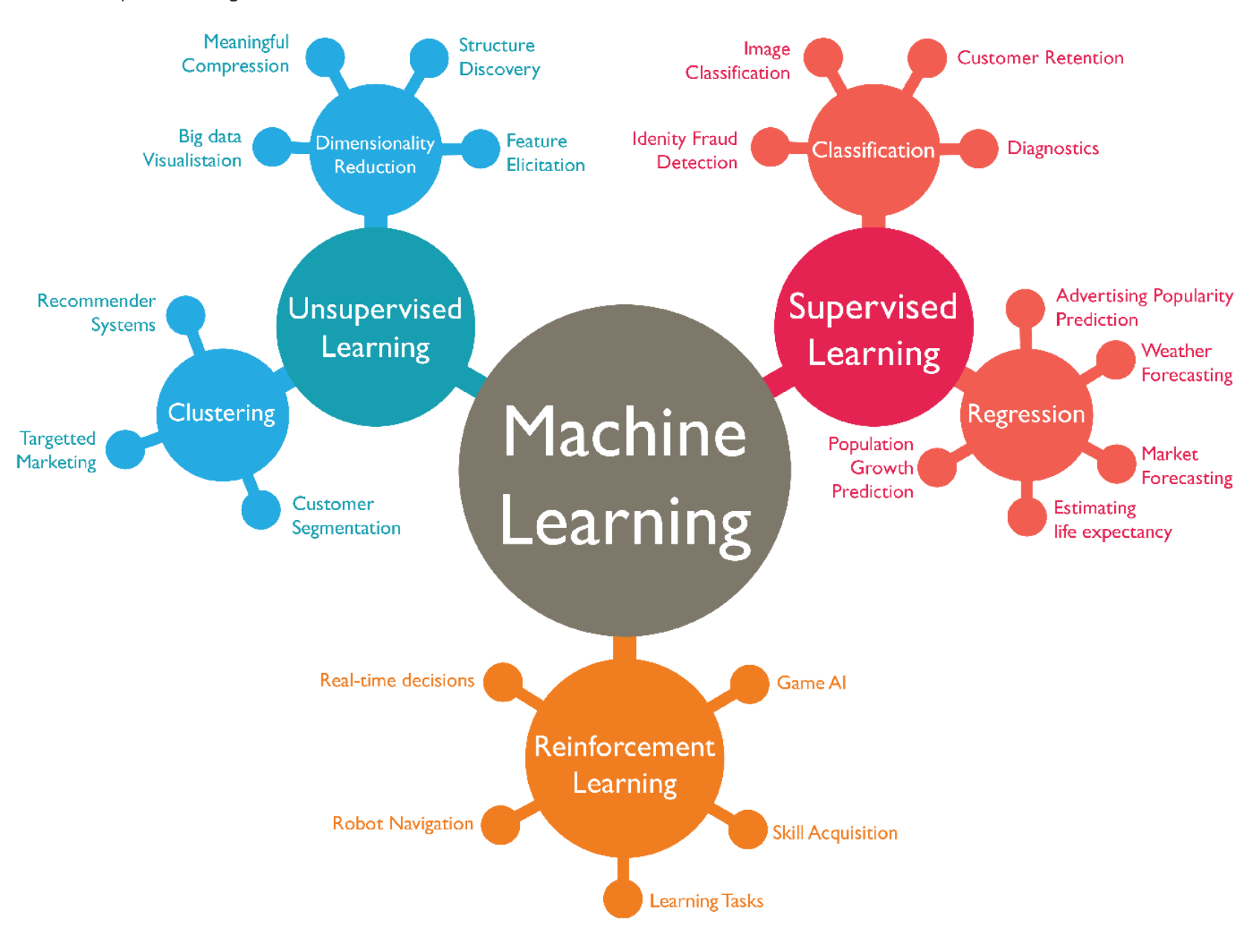
Machine Learning

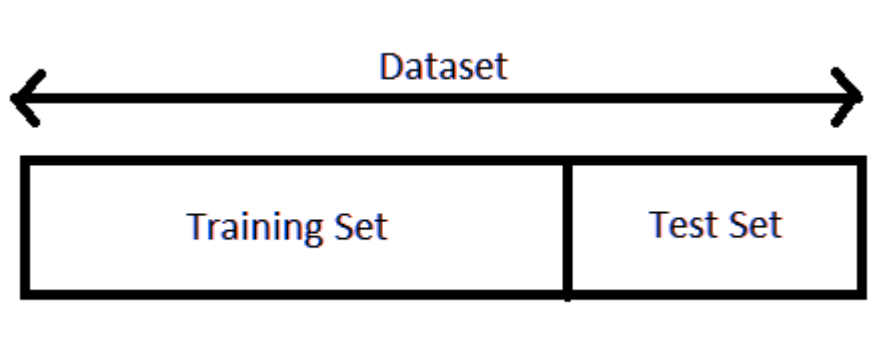
- ML is a technique followed to make a computer learn from the previous experience and make an assumption for the future outcome.
- It can learn and adapt to the new data without any human intervention.
- It needs prior training so that it can be tested to the new data.



Credits - Image from Internet (www.favouriteblog.com)

ML Dataset

In machine learning, we divide the dataset into two.



- **Training Data** - Here we train the machine learning model by showing both inputs and outputs.
- **Testing Data** - Here we test the model where inputs (new data) are not mapped with outputs. We will check the model performance in terms of accuracy.

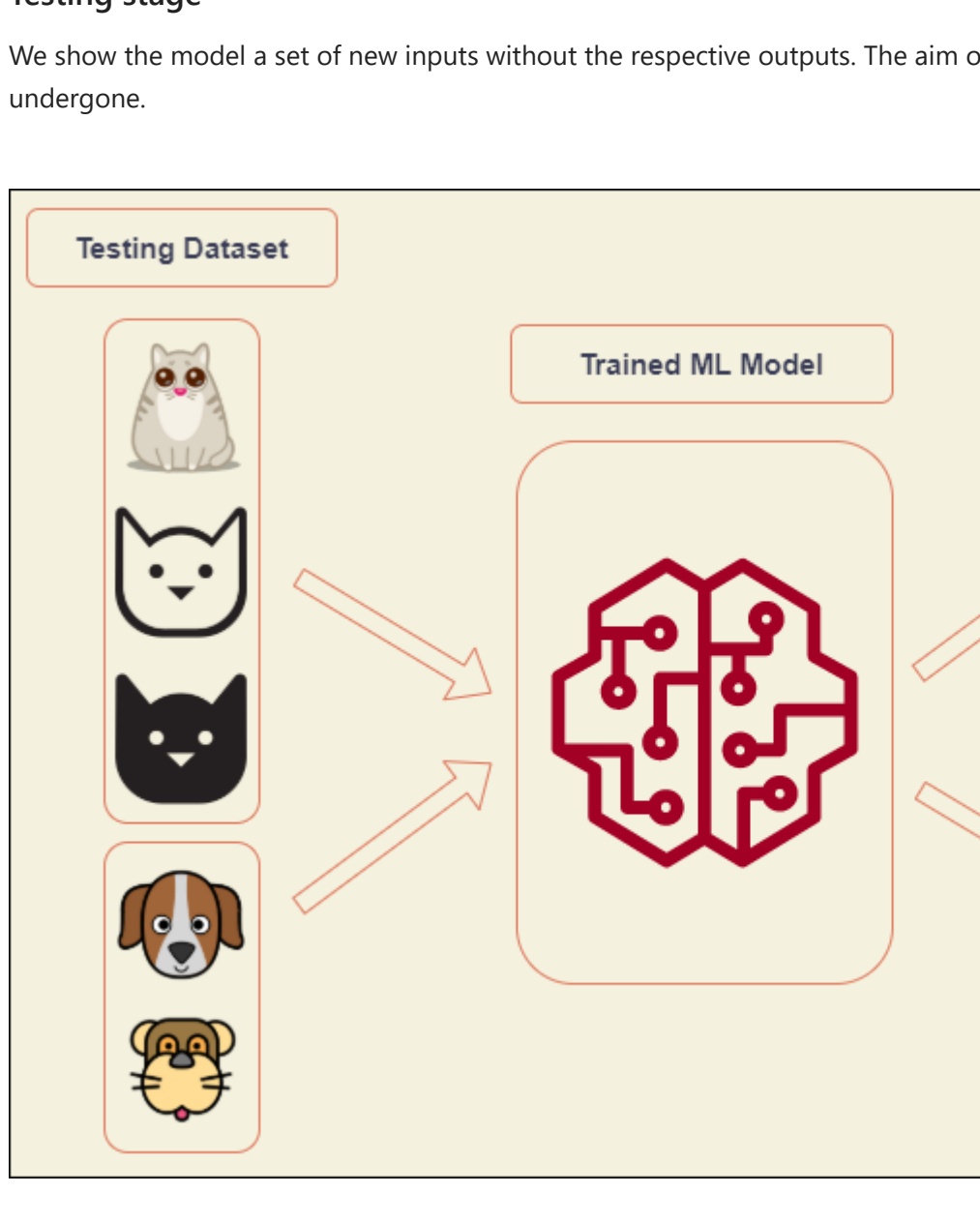
Credits - Image from Internet

Supervised Learning

The computer is presented with both example inputs and their respective outputs. The algorithm learns a general rule to map the inputs to the outputs.

Training stage

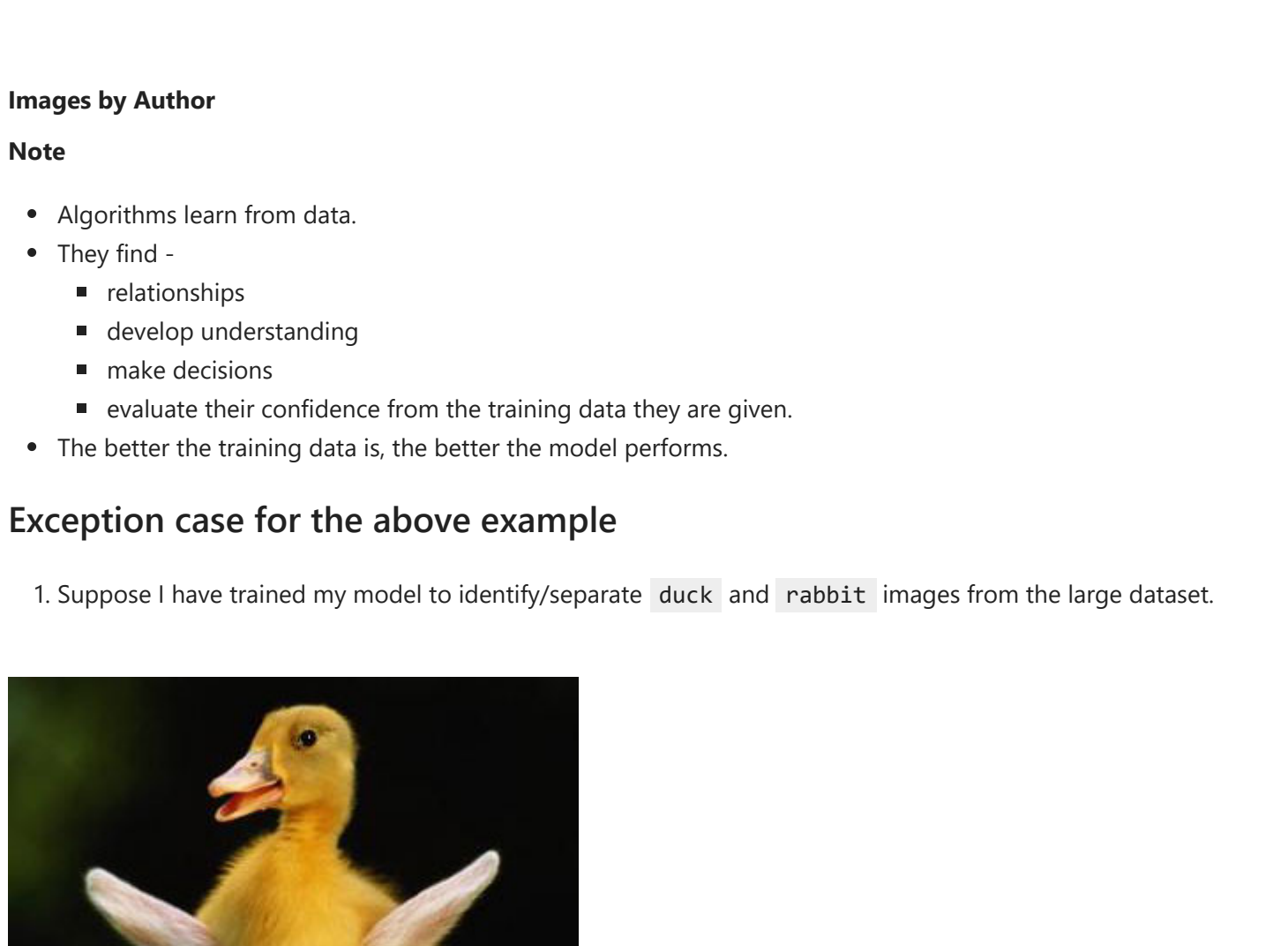
We show the model a set of inputs along with the respective outputs. The task of the model is to learn by mapping the inputs and outputs.



The model is trained from the dataset that is fed. It will completely learn from it.

Testing stage

We show the model a set of new inputs without the respective outputs. The aim of the model is to predict based on the learning it had undergone.



The model predicts the category based on the previous training or learning.

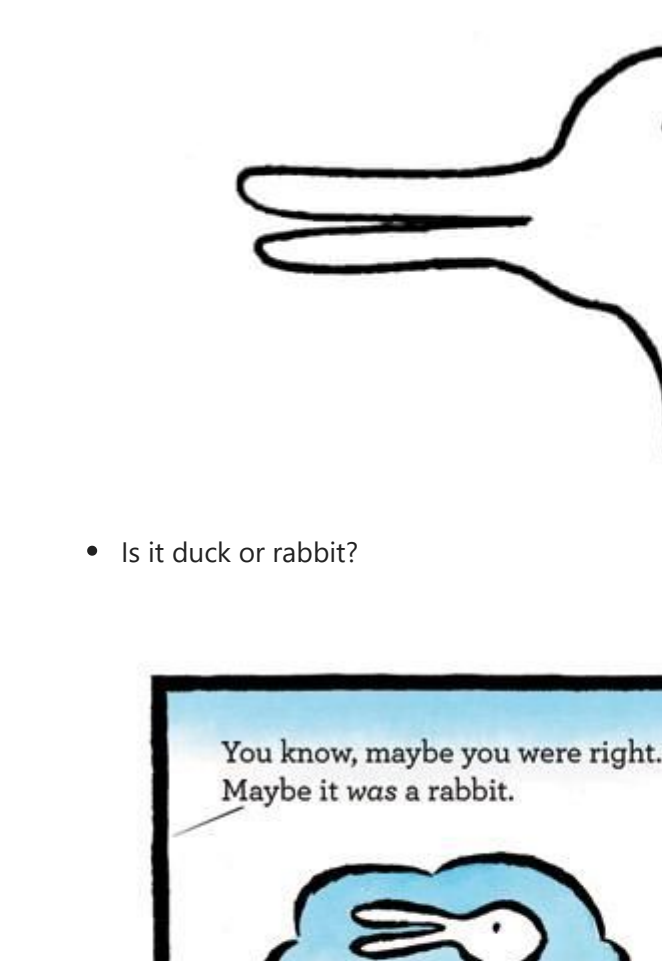
Images by Author

Note

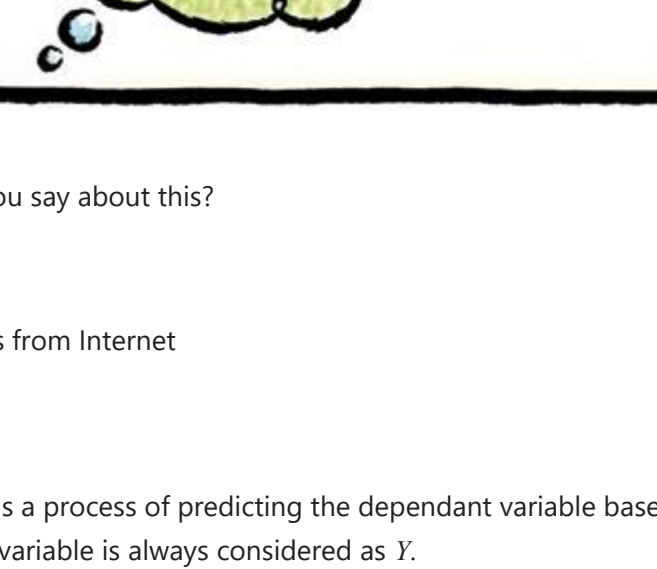
- Algorithms learn from data.
- They find -
  - relationships
  - develop understanding
  - make decisions
  - evaluate their confidence from the training data they are given.
- The better the training data is, the better the model performs.

Exception case for the above example

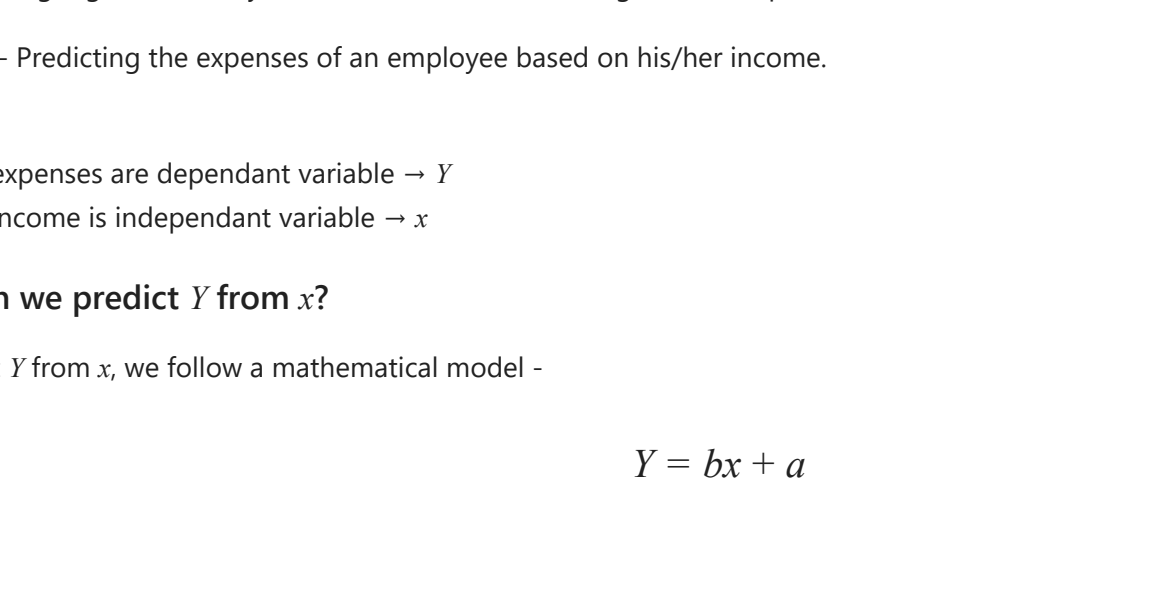
1. Suppose I have trained my model to identify/separate `duck` and `rabbit` images from the large dataset.



1. Now, I need to test my model with the new data.
   - New image



   - Is it duck or rabbit?



2. What can you say about this?

Credits - Images from Internet

Regression

- Regression is a process of predicting the dependant variable based on the independant variable.
- Dependant variable is always considered as Y.
- Independant variable is always considered as x.
- For doing regression analysis, there needs to be a strong relationship between x and Y.

Example - Predicting the expenses of an employee based on his/her income.

- Here,
  - expenses are dependant variable → Y
  - income is independant variable → x

How can we predict Y from x?

To predict Y from x, we follow a mathematical model -

$$Y = bx + a$$

- Here,
  - b → Co-efficient parameter
  - a → Bias parameter
- Other terms
  - $\hat{Y}$ (Y hat) → Predicted Y value
  - Y → Actual value

What is Error?

- The distance between $\hat{Y}$ and Y is called Error.
- Our aim while building the Regression model is to minimize the Error.

Credits - Image from Internet

Let's fit the line ... !

`import` packages

```
In [1]:
import warnings
warnings.filterwarnings("ignore")
```

To ignore warnings that arise while executing the library methods

```
In [2]:
import pandas as pd
import numpy as np
import seaborn as sns

from matplotlib import pyplot as plt
```

If you do not have `seaborn`

```
py -m pip install seaborn --user
```

Data exploration

```
In [3]:
data_source = 'https://bit.ly/3EXRnij'
```

```
In [4]:
# read data
df = pd.read_csv(data_source)
```

```
In [5]:
# shape
df.shape
```

```
Out[5]: (200, 2)
```

```
In [6]:
# head
df.head()
```

Out[6]:

	Height(Inches)	Weight(Pounds)
0	65.78	112.99
1	71.52	136.49
2	69.40	153.03
3	68.22	142.34
4	67.79	144.30

- Dependant variable (Y) → Weight
- Independant variable (x) → Height

```
In [7]:
# Create Y and x (array)
Y = np.array(df['Weight(Pounds)'].to_list())
x = np.array(df['Height(Inches)'].to_list())
```

How x is related to Y?

```
In [8]:
# correlation
hw_corr = df.corr()
```

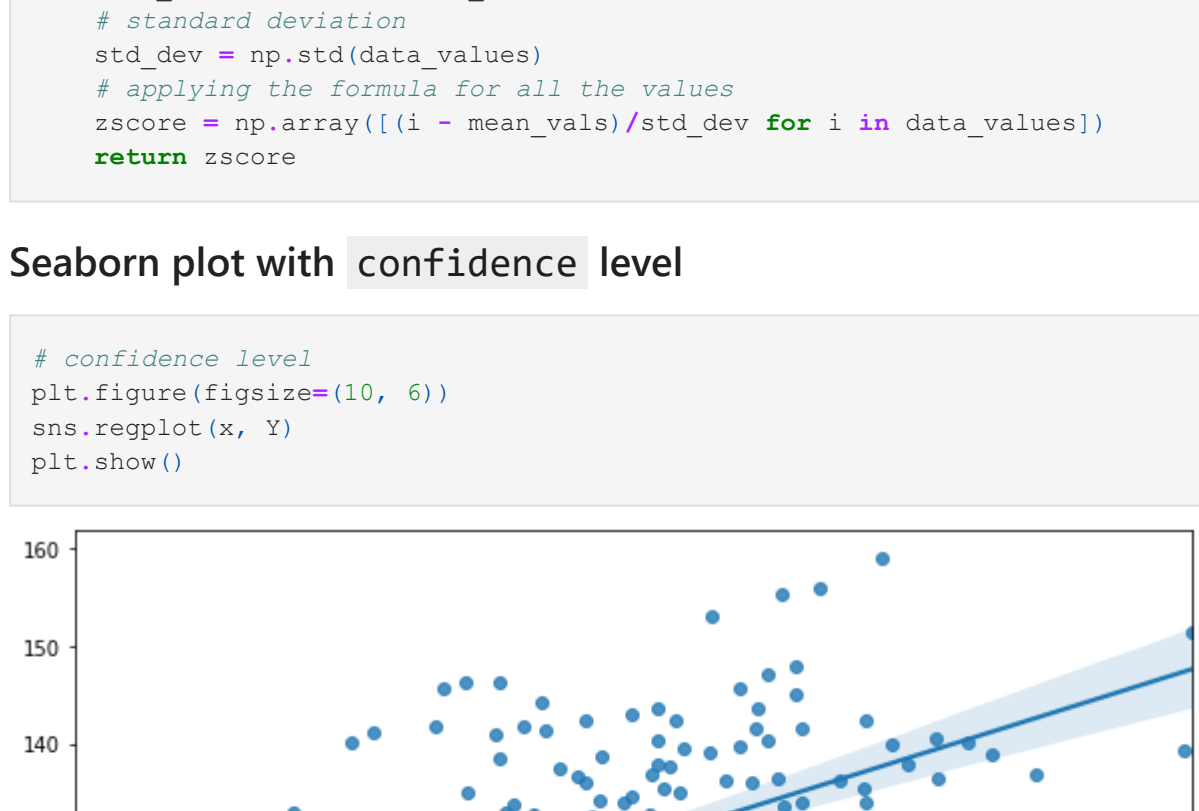
```
In [9]:
hw_corr
```

Out[9]:

	Height(Inches)	Weight(Pounds)
Height(Inches)	1.000000	0.556865
Weight(Pounds)	0.556865	1.000000

From above, we can say `Weights` and `Heights` are highly correlated.

```
In [10]:
# scatter plot (10, 6)
plt.figure(figsize=(10, 6))
plt.scatter(x, Y)
plt.show()
```



There are outliers in our data.

Outlier detection function

```
In [11]:
# whatever value is > 3 and < -3 is an outlier

def calculate_zscore(data_values):
    # mean
    mean_vals = np.mean(data_values)
    # standard deviation
    std_dev = np.std(data_values)
    # applying the formula for all the values
    zscore = np.array([(i - mean_vals)/std_dev for i in data_values])
    return zscore
```

Seaborn plot with `confidence` level

```
In [12]:
# confidence level
plt.figure(figsize=(10, 6))
sns.regplot(x, Y)
plt.show()
```

Model Implementation

- More info → https://towardsdatascience.com/gradient-descent-animation-1-simple-linear-regression-e49315b24672

```
In [13]:
# b, a - polyfit
b, a = np.polyfit(x, Y, 1) # sgd
```

```
In [14]:
# b
b
```

```
Out[14]: 3.432676129271628
```

```
In [15]:
# a
a
```

```
Out[15]: -106.02770644878133
```

Broadcasting

```
In [16]:
f = [1, 2, 3, 4]
s = f * 3
print(s)
```

```
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

```
In [17]:
g = np.array([1, 2, 3, 4])
h = g * 3
print(h)
```

```
[ 3  6  9 12]
```

```
In [18]:
h + 4
```

```
Out[18]: array([ 7, 10, 13, 16])
```

```
In [19]:
type(x)
```

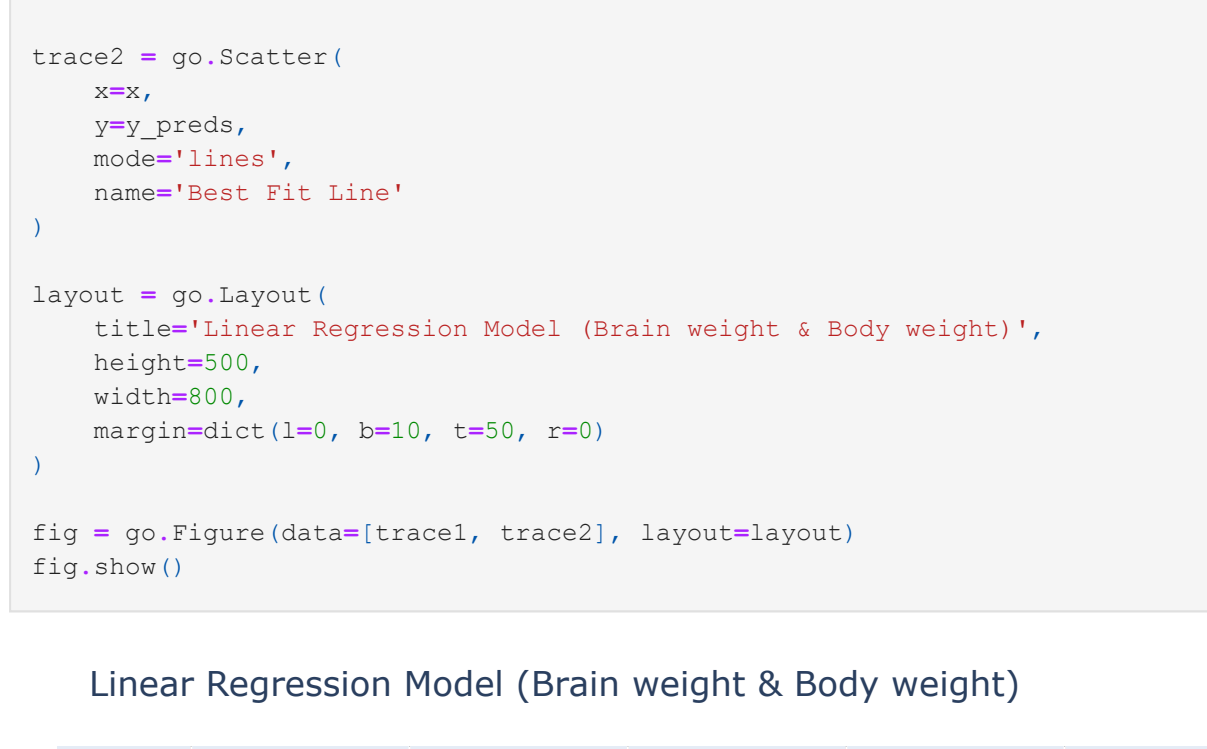
```
Out[19]: numpy.ndarray
```

```
In [20]:
# y_preds = bx + a
y_preds = (b * x) + a
```

```
In [21]:
# y_preds
# print(y_preds)
```

Best fit line

```
In [22]:
# scatter and line
plt.figure(figsize=(10, 6))
plt.scatter(x, Y, color='blue')
plt.plot(x, y_preds, color='red')
plt.show()
```



Plotly plot for interactivity

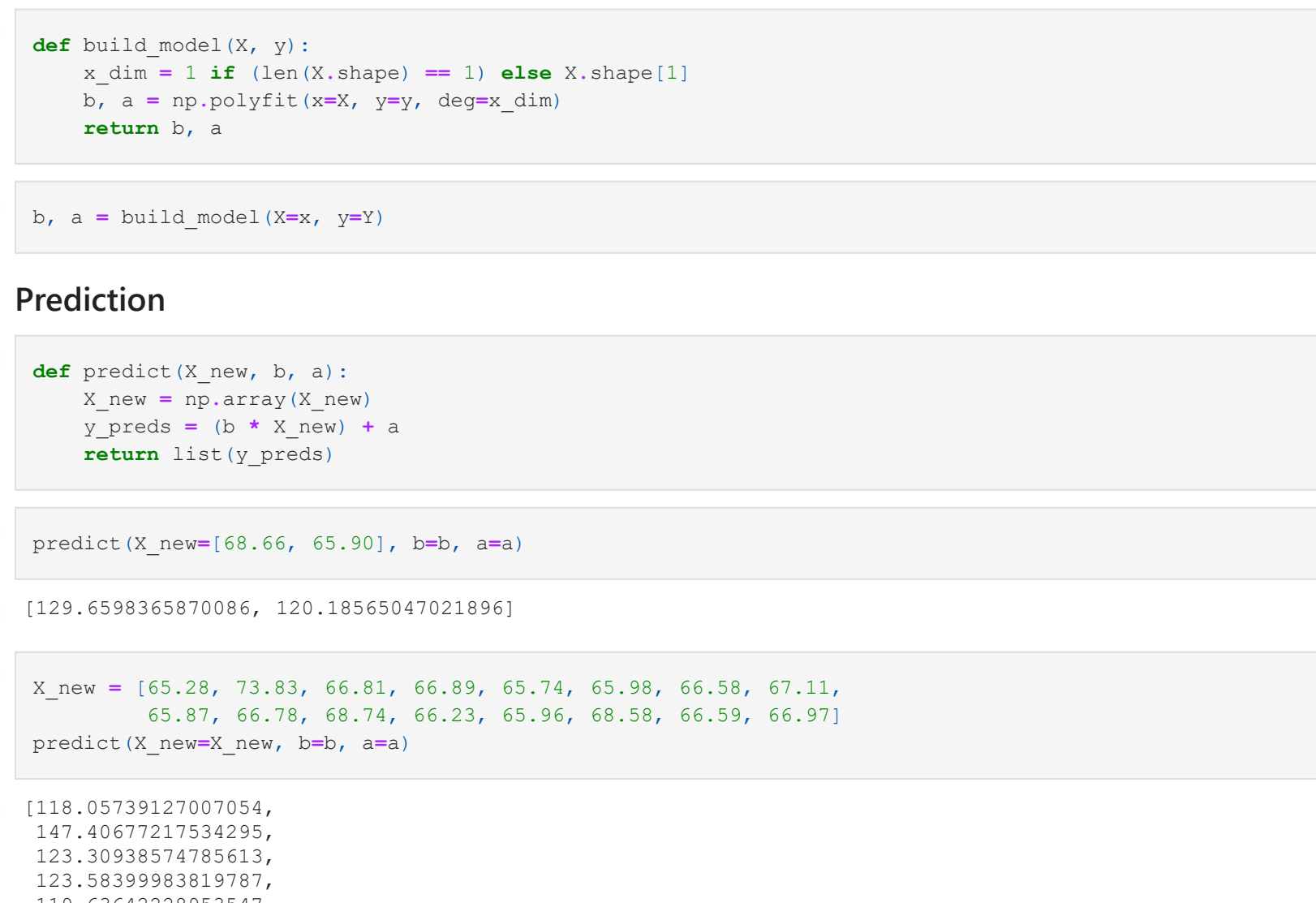
```
In [23]:
import plotly.graph_objects as go
```

```
In [24]:
trace1 = go.Scatter(
    x=x,
    y=Y,
    mode='markers',
    name='Data Values'
)

trace2 = go.Scatter(
    x=x,
    y=y_preds,
    mode='lines',
    name='Best Fit Line'
)

layout = go.Layout(
    title='Linear Regression Model (Brain weight & Body weight)',
    height=500,
    width=800,
    margin=dict(l=0, b=10, t=50, r=0)
)

fig = go.Figure(data=[trace1, trace2], layout=layout)
fig.show()
```



Model development

```
In [25]:
def build_model(X, y):
    x_dim = 1 if (len(X.shape) == 1) else X.shape[1]
    b, a = np.polyfit(x=X, y=y, deg=x_dim)
    return b, a
```

```
In [26]:
b, a = build_model(X=x, y=Y)
```

Prediction

```
In [27]:
def predict(X_new, b, a):
    X_new = np.array(X_new)
    y_preds = (b * X_new) + a
    return list(y_preds)
```

```
In [28]:
predict(X_new=[68.66, 65.90], b=b, a=a)
```

```
Out[28]: [129.6598365870086, 120.18565047021896]
```

```
In [29]:
X_new = [65.28, 73.83, 66.81, 66.89, 65.74, 65.98, 66.58, 67.11,
         65.87, 66.78, 68.74, 66.23, 65.96, 68.58, 66.59, 66.97]
predict(X_new=X_new, b=b, a=a)
```

```
Out[29]: [118.05739127007054,
 147.40677217534295,
 123.30938574785613,
 123.58399983819787,
 119.63642228953547,
 120.4602645605607,
 122.51987023812366,
 124.3391885866376,
 120.08267018634082,
 123.20640546397799,
 129.93445067735036,
 121.3184335928786,
 120.39161103797524,
 129.38522249666693,
 122.55419699941639,
 123.85861392853958]
```

What about Multiple Linear Regression model?

Dependant variable is depends on mutiple Independant variables.

Model -

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n + a$$

- $x_1, x_2, x_3, \ldots, x_n$ are independant variables.
- $b_1, b_2, b_3, \ldots, b_n$ are coefficient variables.
- a is bias variable.

The main aim of any machine learning model is to minimize the error.

Your Task

- Implement the above model with both `seaborn` and `numpy` for the dataset of **Brain and Body Relationship**
  - Data source → https://bit.ly/349nDxL