Logistic Regression

- It's a classification algorithm that is used where the target var is of categorical nature.
- The main objective behind LR is to determine the relationship between features and the prob of a particular outcome.
- 3 Types of LR: 1. Binary LR (ex. 0, 1/ pass, fail/ true, false, 2. Multinomial LR, 3. Ordinal LR
- LR supports linear solutions, KNN supports nonlinear

$$\hat{x} = m * x + b$$

$$b = \frac{\sum y - m \sum x}{n}$$

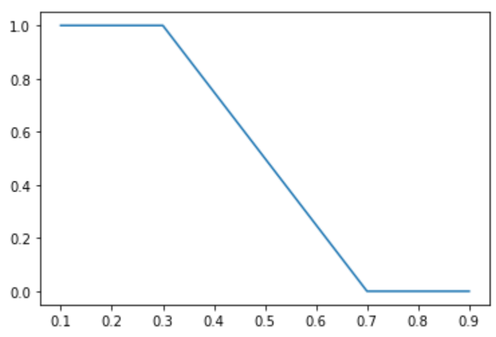
In [63]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
from sklearn.preprocessing import LabelEncoder #to convers categorical features into numeric values
```

In [121…
```
class Logistic_Regression:

    def __init__(self, df):
        self.df = df
        self.x_sum = sum(df['X'])
        self.y_sum = sum(df['y'])
        self.n = len(df)

    def m_value(self):
        m_nume = (self.n * sum(df['X']*df['y'])) - (self.x_sum*self.y_sum)
        m_deno = (self.n * sum(df['X']**2))- (self.x_sum**2)
        return m_nume/m_deno

    def b_value(self):
        self.b_val = (self.y_sum - (self.m_value()*self.x_sum))

    def x_cap(self):
        self.x_cap_val = [(self.m_value() * i) + self.b_val for i in df['X']]
        return self.x_cap_val

#     def sigmoid(self):
#         for x_cap in self.x_cap_val:
#             y_pred = 1/(1+np.exp(x_cap))
#             print(y_pred)

#     def sigmoid(self):
#         y_pred = [1/(1+np.exp(x_cap)) for x_cap in self.x_cap_val]
#         return y_pred


#     def y_pred_val(self):
#         for pred in self.sigmoid():
#             if pred >= 0.5:
#                 print(1)
#             else:
#                 print(0)

#     def y_decode(self):
#         for pred_decode in self.y_pred_val():
#             if pred_decode == 1:
#                 print('Y')
#             else:
#                 print('N')

df = {'X' : [0.1, 0.2,0.3,0.7,0.8,0.9],'y' : ['Y', 'Y', 'Y', 'N', 'N', 'N']}

df = pd.DataFrame(df)
le = LabelEncoder()
encoded_y = le.fit_transform(df['y'])
df['y'] = encoded_y

log_reg = Logistic_Regression(df)
print(log_reg.m_value())
print(log_reg.b_value())
print(log_reg.x_cap())
#print(log_reg.sigmoid())
```

```
-1.551724137931034
None
[7.499999999999999, 7.344827586206895, 7.189655172413792, 6.568965517241379, 6.413793103448275, 6.258620689655172]
```

In [118…
```
plt.plot(df['X'], df['y'])
```

Out[118…
```
[<matplotlib.lines.Line2D at 0x15368bcedf0>]
```



In [141…
```
# predicted

y_pred = [1/(1+np.exp(pred)) for pred in log_reg.x_cap()]
```

Out[141…
```
[1, 2, 3]
```

In [130…
```
#threshold
threshold = [1 if j >=0.5 else 0 for j in y_pred]
```

In [131…
```
#decode to org values

y_decode = ['Y 'if k==1 else 'N' for k in y_pred]
```

In [132…
```
# accuracy score

y_org = df['y']
y_pred = y_decode
```

In [134…
```
c = 0
total = 0
for yo, yp in zip(y_org, y_pred):
    if yo == yp:
        c = c+1
    else:
        pass
    total = total+1
```

In [137…
```
c, total
```

Out[137…
```
(0, 6)
```

In [138…
```
acc = (c/total)*100
```

In [139…
```
acc
```

Out[139…
```
0.0
```