**Cancer Prediction**

Dataset Information:

Target Variable (y):

* Diagnosis (M = malignant, B = benign)

Ten features (X) are computed for each cell nucleus:

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension (coastline approximation - 1)

For each characteristic three measures are given:

a. Mean

b. Standard error

c. Largest/ Worst

[**Watch Video Tutorial**](https://www.youtube.com/c/YBIFoundation?sub_confirmation=1)

In [1]:

*# Step 1 : import library*

import pandas as pd

In [2]:

*# Step 2 : import data*

cancer = pd.read\_csv('https://github.com/YBIFoundation/Dataset/raw/main/Cancer.csv')

In [3]:

cancer.head()

Out[3]:

|  | id | diagnosis | radius\_mean | texture\_mean | perimeter\_mean | area\_mean | smoothness\_mean | compactness\_mean | concavity\_mean | concave points\_mean | ... | texture\_worst | perimeter\_worst | area\_worst | smoothness\_worst | compactness\_worst | concavity\_worst | concave points\_worst | symmetry\_worst | fractal\_dimension\_worst | Unnamed: 32 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | ... | 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 | NaN |
| 1 | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | ... | 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.2416 | 0.1860 | 0.2750 | 0.08902 | NaN |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | ... | 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.4504 | 0.2430 | 0.3613 | 0.08758 | NaN |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | ... | 26.50 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.6869 | 0.2575 | 0.6638 | 0.17300 | NaN |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | ... | 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.4000 | 0.1625 | 0.2364 | 0.07678 | NaN |

5 rows × 33 columns

In [4]:

cancer.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 569 entries, 0 to 568

Data columns (total 33 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 569 non-null int64

1 diagnosis 569 non-null object

2 radius\_mean 569 non-null float64

3 texture\_mean 569 non-null float64

4 perimeter\_mean 569 non-null float64

5 area\_mean 569 non-null float64

6 smoothness\_mean 569 non-null float64

7 compactness\_mean 569 non-null float64

8 concavity\_mean 569 non-null float64

9 concave points\_mean 569 non-null float64

10 symmetry\_mean 569 non-null float64

11 fractal\_dimension\_mean 569 non-null float64

12 radius\_se 569 non-null float64

13 texture\_se 569 non-null float64

14 perimeter\_se 569 non-null float64

15 area\_se 569 non-null float64

16 smoothness\_se 569 non-null float64

17 compactness\_se 569 non-null float64

18 concavity\_se 569 non-null float64

19 concave points\_se 569 non-null float64

20 symmetry\_se 569 non-null float64

21 fractal\_dimension\_se 569 non-null float64

22 radius\_worst 569 non-null float64

23 texture\_worst 569 non-null float64

24 perimeter\_worst 569 non-null float64

25 area\_worst 569 non-null float64

26 smoothness\_worst 569 non-null float64

27 compactness\_worst 569 non-null float64

28 concavity\_worst 569 non-null float64

29 concave points\_worst 569 non-null float64

30 symmetry\_worst 569 non-null float64

31 fractal\_dimension\_worst 569 non-null float64

32 Unnamed: 32 0 non-null float64

dtypes: float64(31), int64(1), object(1)

memory usage: 146.8+ KB

In [5]:

cancer.describe()

Out[5]:

|  | id | radius\_mean | texture\_mean | perimeter\_mean | area\_mean | smoothness\_mean | compactness\_mean | concavity\_mean | concave points\_mean | symmetry\_mean | ... | texture\_worst | perimeter\_worst | area\_worst | smoothness\_worst | compactness\_worst | concavity\_worst | concave points\_worst | symmetry\_worst | fractal\_dimension\_worst | Unnamed: 32 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 5.690000e+02 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | ... | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 0.0 |
| mean | 3.037183e+07 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.104341 | 0.088799 | 0.048919 | 0.181162 | ... | 25.677223 | 107.261213 | 880.583128 | 0.132369 | 0.254265 | 0.272188 | 0.114606 | 0.290076 | 0.083946 | NaN |
| std | 1.250206e+08 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.052813 | 0.079720 | 0.038803 | 0.027414 | ... | 6.146258 | 33.602542 | 569.356993 | 0.022832 | 0.157336 | 0.208624 | 0.065732 | 0.061867 | 0.018061 | NaN |
| min | 8.670000e+03 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.019380 | 0.000000 | 0.000000 | 0.106000 | ... | 12.020000 | 50.410000 | 185.200000 | 0.071170 | 0.027290 | 0.000000 | 0.000000 | 0.156500 | 0.055040 | NaN |
| 25% | 8.692180e+05 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.064920 | 0.029560 | 0.020310 | 0.161900 | ... | 21.080000 | 84.110000 | 515.300000 | 0.116600 | 0.147200 | 0.114500 | 0.064930 | 0.250400 | 0.071460 | NaN |
| 50% | 9.060240e+05 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.092630 | 0.061540 | 0.033500 | 0.179200 | ... | 25.410000 | 97.660000 | 686.500000 | 0.131300 | 0.211900 | 0.226700 | 0.099930 | 0.282200 | 0.080040 | NaN |
| 75% | 8.813129e+06 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.130400 | 0.130700 | 0.074000 | 0.195700 | ... | 29.720000 | 125.400000 | 1084.000000 | 0.146000 | 0.339100 | 0.382900 | 0.161400 | 0.317900 | 0.092080 | NaN |
| max | 9.113205e+08 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.345400 | 0.426800 | 0.201200 | 0.304000 | ... | 49.540000 | 251.200000 | 4254.000000 | 0.222600 | 1.058000 | 1.252000 | 0.291000 | 0.663800 | 0.207500 | NaN |

8 rows × 32 columns

In [6]:

*# Step 3 : define target (y) and features (X)*

In [7]:

cancer.columns

Out[7]:

Index(['id', 'diagnosis', 'radius\_mean', 'texture\_mean', 'perimeter\_mean',

'area\_mean', 'smoothness\_mean', 'compactness\_mean', 'concavity\_mean',

'concave points\_mean', 'symmetry\_mean', 'fractal\_dimension\_mean',

'radius\_se', 'texture\_se', 'perimeter\_se', 'area\_se', 'smoothness\_se',

'compactness\_se', 'concavity\_se', 'concave points\_se', 'symmetry\_se',

'fractal\_dimension\_se', 'radius\_worst', 'texture\_worst',

'perimeter\_worst', 'area\_worst', 'smoothness\_worst',

'compactness\_worst', 'concavity\_worst', 'concave points\_worst',

'symmetry\_worst', 'fractal\_dimension\_worst', 'Unnamed: 32'],

dtype='object')

In [8]:

y = cancer['diagnosis']

In [9]:

X = cancer.drop(['id','diagnosis','Unnamed: 32'],axis=1)

In [10]:

*# Step 4 : train test split*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

In [11]:

*# check shape of train and test sample*

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

Out[11]:

((398, 30), (171, 30), (398,), (171,))

In [12]:

*# Step 5 : select model*

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(max\_iter=5000)

In [13]:

*# Step 6 : train or fit model*

model.fit(X\_train,y\_train)

Out[13]:

LogisticRegression(max\_iter=5000)

In [14]:

model.intercept\_

Out[14]:

array([-29.89794524])

In [15]:

model.coef\_

Out[15]:

array([[-0.8987571 , -0.18472023, 0.25082147, -0.02542693, 0.14122364,

0.18978336, 0.41378945, 0.22182788, 0.1734014 , 0.03280365,

0.00205045, -1.15928761, -0.20539986, 0.1253185 , 0.024131 ,

-0.05334342, 0.01690814, 0.02921515, 0.02999633, -0.01471961,

-0.27823109, 0.44309644, 0.1795841 , 0.01185975, 0.29111449,

0.5870389 , 1.05455886, 0.45091109, 0.52893031, 0.07529152]])

In [16]:

*# Step 7 : predict model*

y\_pred = model.predict(X\_test)

In [17]:

y\_pred

Out[17]:

array(['B', 'M', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B',

'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M',

'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B',

'M', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B',

'B', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'M', 'M',

'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B', 'B',

'M', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'M',

'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B',

'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B',

'M', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M',

'M', 'B', 'M', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'B',

'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B',

'B', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B',

'B', 'B'], dtype=object)

In [18]:

*# Step 8 : model accuracy*

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

In [19]:

confusion\_matrix(y\_test,y\_pred)

Out[19]:

array([[97, 5],

[ 2, 67]])

In [20]:

accuracy\_score(y\_test,y\_pred)

Out[20]:

0.9590643274853801

In [21]:

print(classification\_report(y\_test,y\_pred))

precision recall f1-score support

B 0.98 0.95 0.97 102

M 0.93 0.97 0.95 69

accuracy 0.96 171

macro avg 0.96 0.96 0.96 171

weighted avg 0.96 0.96 0.96 171

**Chance of Admission for Higher Studies**[¶](https://www.kaggleusercontent.com/kf/121441923/eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..DHiRBqGpkwJjY0-abhEtCw.DKQPnHHJ6J1weG_p35h7_KmG7xF6SulUprfKwQT3vSxX2ao_Z1pcLQFYAsh_ZeGmmfIWk1zAHJMVC3kjwmmdirH7QQzkeuFzpuGLATTJKj2zztkZxwJCr9xbW8fZsQxwb-wV6-m94B_Q4vJeqYgnbwb7LrMGzxmsl-aak0RIsYcvCA0gUvJ22eqAd-simGDzCvayp_d-s0WkHyt2Pbuj4rysp24bLMSKow1oLo_8oF_Kdm00V80xqUWP2XQerH6C9Kq3d2yRNkwTkxIZ9cjKR-QKnmZxXju1Mz6QeDn4ggSE1UC75G8VSlt9NVjrymSkGfyXKho2DUCqGnRBPU_2aibCq3TnAQ_mzCBe6xrJLabvmQiRe-UIyFMODxF7elCp5fFpbeiUXg_6clxpxK0VVHAq0zy2-PAf9c87vy7mLyq8bPCA_VuuDiWbJJpiExTK6XHqjcbbjDE-OLR3cuIdDr8mgj6qPXtvj1fk-pTmrza146xL4_u8YY1mRKegVETZJj03CvGBXM0Bi0AuyaBxCc9_qJCl9fqydhg3Z0N5Ubp-1RDhhXLuB1NxrGhqHD-LZNLuPSyMuSiMpigsprutGOT_emyr6U8Bl1Jh9eO2b5nVTtKxx59DfLh0aTWz6Mw5fAQWP3CAkgwl3x4F22wapiR2J1qREd7YwB2ymrBtHpw.1pkAFsSyoHmW9U58z9WQ-w/__results__.html#Chance-of-Admission-for-Higher-Studies)

Predict the chances of admission of a student to a Graduate program based on:

1. GRE Scores (290 to 340)
2. TOEFL Scores (92 to 120)
3. University Rating (1 to 5)
4. Statement of Purpose (1 to 5)
5. Letter of Recommendation Strength (1 to 5)
6. Undergraduate CGPA (6.8 to 9.92)
7. Research Experience (0 or 1)
8. Chance of Admit (0.34 to 0.97)

[**Watch Video Tutorial**](https://www.youtube.com/c/YBIFoundation?sub_confirmation=1)

In [1]:

*# Step 1 : import library*

import pandas as pd

In [2]:

*# Step 2 : import data*

admission = pd.read\_csv('https://github.com/ybifoundation/Dataset/raw/main/Admission%20Chance.csv')

In [3]:

admission.head()

Out[3]:

|  | Serial No | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| 1 | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| 2 | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| 3 | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| 4 | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

In [4]:

admission.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 400 entries, 0 to 399

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Serial No 400 non-null int64

1 GRE Score 400 non-null int64

2 TOEFL Score 400 non-null int64

3 University Rating 400 non-null int64

4 SOP 400 non-null float64

5 LOR 400 non-null float64

6 CGPA 400 non-null float64

7 Research 400 non-null int64

8 Chance of Admit 400 non-null float64

dtypes: float64(4), int64(5)

memory usage: 28.2 KB

In [5]:

admission.describe()

Out[5]:

|  | Serial No | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 400.000000 | 400.000000 | 400.000000 | 400.000000 | 400.000000 | 400.000000 | 400.000000 | 400.000000 | 400.000000 |
| mean | 200.500000 | 316.807500 | 107.410000 | 3.087500 | 3.400000 | 3.452500 | 8.598925 | 0.547500 | 0.724350 |
| std | 115.614301 | 11.473646 | 6.069514 | 1.143728 | 1.006869 | 0.898478 | 0.596317 | 0.498362 | 0.142609 |
| min | 1.000000 | 290.000000 | 92.000000 | 1.000000 | 1.000000 | 1.000000 | 6.800000 | 0.000000 | 0.340000 |
| 25% | 100.750000 | 308.000000 | 103.000000 | 2.000000 | 2.500000 | 3.000000 | 8.170000 | 0.000000 | 0.640000 |
| 50% | 200.500000 | 317.000000 | 107.000000 | 3.000000 | 3.500000 | 3.500000 | 8.610000 | 1.000000 | 0.730000 |
| 75% | 300.250000 | 325.000000 | 112.000000 | 4.000000 | 4.000000 | 4.000000 | 9.062500 | 1.000000 | 0.830000 |
| max | 400.000000 | 340.000000 | 120.000000 | 5.000000 | 5.000000 | 5.000000 | 9.920000 | 1.000000 | 0.970000 |

In [6]:

*# Step 3 : define target (y) and features (X)*

In [7]:

admission.columns

Out[7]:

Index(['Serial No', 'GRE Score', 'TOEFL Score', 'University Rating', ' SOP',

'LOR ', 'CGPA', 'Research', 'Chance of Admit '],

dtype='object')

In [8]:

y = admission['Chance of Admit ']

In [9]:

X = admission.drop(['Serial No','Chance of Admit '],axis=1)

In [10]:

*# Step 4 : train test split*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

In [11]:

*# check shape of train and test sample*

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

Out[11]:

((280, 7), (120, 7), (280,), (120,))

In [12]:

*# Step 5 : select model*

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

In [13]:

*# Step 6 : train or fit model*

model.fit(X\_train,y\_train)

Out[13]:

LinearRegression()

In [14]:

model.intercept\_

Out[14]:

-1.2831244932033985

In [15]:

model.coef\_

Out[15]:

array([ 0.00204057, 0.00287273, 0.00566887, -0.00380559, 0.01973175,

0.11314449, 0.02061553])

In [16]:

*# Step 7 : predict model*

y\_pred = model.predict(X\_test)

In [17]:

y\_pred

Out[17]:

array([0.71426327, 0.72534136, 0.69677103, 0.66566584, 0.57483872,

0.93087527, 0.93701113, 0.72361387, 0.81130158, 0.62223963,

0.59629648, 0.80084072, 0.52537944, 0.79174558, 0.84064992,

0.66429594, 0.65136589, 0.66990687, 0.75794085, 0.86072023,

0.66088101, 0.85570763, 0.84777425, 0.95033179, 0.68750762,

0.65907671, 0.65279623, 0.5709259 , 0.55895645, 0.57990205,

0.54497918, 0.7570717 , 0.69682571, 0.77286067, 0.64320811,

0.5183554 , 0.43816818, 0.84654064, 0.90398354, 0.80517781,

0.72218971, 0.72882587, 0.68145136, 0.88592237, 0.77208852,

0.78778085, 0.95526121, 0.88586486, 0.59980416, 0.50690214,

0.59947098, 0.63380406, 0.82841217, 0.44911724, 0.71068577,

0.77335748, 0.68851557, 0.64486026, 0.85537724, 0.65517768,

0.65046031, 0.90818978, 0.63422429, 0.68658606, 0.72150268,

0.69030545, 0.59381287, 0.93813035, 0.58997351, 0.91542587,

0.59283415, 0.93351713, 0.59478751, 0.71380389, 0.54346237,

0.84710913, 0.6084418 , 0.7257337 , 0.67545704, 0.81387503,

0.70259527, 0.88600461, 0.67084016, 0.53064995, 0.77790726,

0.65780713, 0.78970635, 0.54709634, 0.77924705, 0.66750436,

0.69363338, 0.69891086, 0.92185813, 0.70469056, 0.62554306,

0.62208829, 0.73828086, 0.67369114, 0.76391913, 0.61985049,

0.92865957, 0.70430038, 0.9828821 , 0.82502993, 0.78261009,

0.83438446, 0.66840368, 0.70165011, 0.64534281, 0.5715406 ,

0.80739359, 0.69273815, 0.80585447, 0.6102703 , 0.54641206,

0.76301749, 0.71080317, 0.6261331 , 0.83951248, 0.68578269])

In [18]:

*# Step 8 : model accuracy*

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error

In [19]:

mean\_absolute\_error(y\_test,y\_pred)

Out[19]:

0.04400128934232653

In [20]:

mean\_absolute\_percentage\_error(y\_test,y\_pred)

Out[20]:

0.07575278864605442

In [21]:

mean\_squared\_error(y\_test,y\_pred)

Out[21]:

0.004038263715495693

**Binary Classification**

[**Watch Video Tutorial**](https://www.youtube.com/c/YBIFoundation?sub_confirmation=1)

In [1]:

*# Step 1 : import library*

import pandas as pd

In [2]:

*# Step 2 : import data*

diabetes = pd.read\_csv('https://github.com/YBIFoundation/Dataset/raw/main/Diabetes.csv')

In [3]:

diabetes.head()

Out[3]:

|  | pregnancies | glucose | diastolic | triceps | insulin | bmi | dpf | age | diabetes |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [4]:

diabetes.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 pregnancies 768 non-null int64

1 glucose 768 non-null int64

2 diastolic 768 non-null int64

3 triceps 768 non-null int64

4 insulin 768 non-null int64

5 bmi 768 non-null float64

6 dpf 768 non-null float64

7 age 768 non-null int64

8 diabetes 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

In [5]:

diabetes.describe()

Out[5]:

|  | pregnancies | glucose | diastolic | triceps | insulin | bmi | dpf | age | diabetes |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| mean | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| std | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

In [6]:

*# Step 3 : define target (y) and features (X)*

In [7]:

diabetes.columns

Out[7]:

Index(['pregnancies', 'glucose', 'diastolic', 'triceps', 'insulin', 'bmi',

'dpf', 'age', 'diabetes'],

dtype='object')

In [8]:

y = diabetes['diabetes']

In [9]:

X = diabetes.drop(['diabetes'],axis=1)

In [10]:

*# Step 4 : train test split*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

In [11]:

*# check shape of train and test sample*

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

Out[11]:

((537, 8), (231, 8), (537,), (231,))

In [12]:

*# Step 5 : select model*

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(max\_iter=500)

In [13]:

*# Step 6 : train or fit model*

model.fit(X\_train,y\_train)

Out[13]:

LogisticRegression(max\_iter=500)

In [14]:

model.intercept\_

Out[14]:

array([-8.13058703])

In [15]:

model.coef\_

Out[15]:

array([[ 1.01259801e-01, 3.60553853e-02, -2.09736871e-02,

-2.57281495e-03, -2.04295785e-04, 8.24680082e-02,

9.51017756e-01, 2.53493255e-02]])

In [16]:

*# Step 7 : predict model*

y\_pred = model.predict(X\_test)

In [17]:

y\_pred

Out[17]:

array([0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,

0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,

0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,

1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,

0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,

0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,

0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1])

In [18]:

*# Step 8 : model accuracy*

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

In [19]:

confusion\_matrix(y\_test,y\_pred)

Out[19]:

array([[133, 12],

[ 41, 45]])

In [20]:

accuracy\_score(y\_test,y\_pred)

Out[20]:

0.7705627705627706

In [21]:

print(classification\_report(y\_test,y\_pred))

precision recall f1-score support

0 0.76 0.92 0.83 145

1 0.79 0.52 0.63 86

accuracy 0.77 231

macro avg 0.78 0.72 0.73 231

weighted avg 0.77 0.77 0.76 231

**Fish Weight Prediction**

With a dataset of fish species, with some of it characteristic like it vertical, diagonal, length, height, and width. We will try to predict the weight of the fish based on their characteristic. We will use Linear Regression Method to see whether the weight of the fish related to their characteristic.

* Species: Species name of fish
* Weight: Weight of fish in gram
* Length1: Vertical length in cm
* Length2: Diagonal length in cm
* Length3: Cross length in cm
* Height: Height in cm
* Width: Diagonal width in cm

[**Watch Video Tutorial**](https://www.youtube.com/c/YBIFoundation?sub_confirmation=1)

[**Ask Doubt in FREE Live QnA Session**](https://www.ybifoundation.org/session/live-qna-and-doubt-support)

In [ ]:

*# Step 1 : import library*

import pandas as pd

In [ ]:

*# Step 2 : import data*

fish = pd.read\_csv('https://github.com/ybifoundation/Dataset/raw/main/Fish.csv')

In [ ]:

fish.head()

Out[ ]:

|  | Category | Species | Weight | Height | Width | Length1 | Length2 | Length3 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | Bream | 242.0 | 11.5200 | 4.0200 | 23.2 | 25.4 | 30.0 |
| 1 | 1 | Bream | 290.0 | 12.4800 | 4.3056 | 24.0 | 26.3 | 31.2 |
| 2 | 1 | Bream | 340.0 | 12.3778 | 4.6961 | 23.9 | 26.5 | 31.1 |
| 3 | 1 | Bream | 363.0 | 12.7300 | 4.4555 | 26.3 | 29.0 | 33.5 |
| 4 | 1 | Bream | 430.0 | 12.4440 | 5.1340 | 26.5 | 29.0 | 34.0 |

In [ ]:

fish.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 159 entries, 0 to 158

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Category 159 non-null int64

1 Species 159 non-null object

2 Weight 159 non-null float64

3 Height 159 non-null float64

4 Width 159 non-null float64

5 Length1 159 non-null float64

6 Length2 159 non-null float64

7 Length3 159 non-null float64

dtypes: float64(6), int64(1), object(1)

memory usage: 10.1+ KB

In [ ]:

fish.describe()

Out[ ]:

|  | Category | Weight | Height | Width | Length1 | Length2 | Length3 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| count | 159.000000 | 159.000000 | 159.000000 | 159.000000 | 159.000000 | 159.000000 | 159.000000 |
| mean | 3.264151 | 398.326415 | 8.970994 | 4.417486 | 26.247170 | 28.415723 | 31.227044 |
| std | 1.704249 | 357.978317 | 4.286208 | 1.685804 | 9.996441 | 10.716328 | 11.610246 |
| min | 1.000000 | 0.000000 | 1.728400 | 1.047600 | 7.500000 | 8.400000 | 8.800000 |
| 25% | 2.000000 | 120.000000 | 5.944800 | 3.385650 | 19.050000 | 21.000000 | 23.150000 |
| 50% | 3.000000 | 273.000000 | 7.786000 | 4.248500 | 25.200000 | 27.300000 | 29.400000 |
| 75% | 4.500000 | 650.000000 | 12.365900 | 5.584500 | 32.700000 | 35.500000 | 39.650000 |
| max | 7.000000 | 1650.000000 | 18.957000 | 8.142000 | 59.000000 | 63.400000 | 68.000000 |

In [ ]:

*# Step 3 : define target (y) and features (X)*

In [ ]:

fish.columns

Out[ ]:

Index(['Category', 'Species', 'Weight', 'Height', 'Width', 'Length1',

'Length2', 'Length3'],

dtype='object')

In [ ]:

y = fish['Weight']

In [ ]:

X = fish[['Category','Height', 'Width', 'Length1',

'Length2', 'Length3']]

In [ ]:

*# Step 4 : train test split*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

In [ ]:

*# check shape of train and test sample*

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

Out[ ]:

((111, 6), (48, 6), (111,), (48,))

In [ ]:

*# Step 5 : select model*

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

In [ ]:

*# Step 6 : train or fit model*

model.fit(X\_train,y\_train)

Out[ ]:

LinearRegression

LinearRegression()

In [ ]:

model.intercept\_

Out[ ]:

-684.4235918478537

In [ ]:

model.coef\_

Out[ ]:

array([ 35.19634977, 52.19372157, -37.13869125, 11.2218449 ,

78.11233002, -59.11783139])

In [ ]:

*# Step 7 : predict model*

y\_pred = model.predict(X\_test)

In [ ]:

y\_pred

Out[ ]:

array([ 475.93351307, 525.81910195, 77.63275849, 881.10235121,

160.9685664 , 255.94371856, 361.87029932, 358.87068094,

499.83411068, -150.07834151, -115.91810869, 428.65470115,

114.67533404, 812.51385122, 586.5071178 , 273.38510858,

579.63900729, 225.18126845, 639.26068037, 85.00820599,

136.92159041, -87.7778087 , 629.97231046, 732.63097812,

859.8720695 , -166.76928607, 342.04209934, 722.92198147,

321.44827179, 787.98248357, 486.93194673, 541.89982795,

376.74813045, 624.81211202, -170.11945033, 917.76513801,

792.26439518, -21.15655005, 300.24921659, 914.07325473,

621.05636286, 934.17373986, 676.85479574, 653.92304403,

615.51226767, 336.61090622, 505.75519147, -33.53283763])

In [ ]:

*# Step 8 : model accuracy*

from sklearn.metrics import mean\_absolute\_error, r2\_score

In [ ]:

mean\_absolute\_error(y\_test,y\_pred)

Out[ ]:

99.58910366731813

In [ ]:

r2\_score(y\_test,y\_pred)

Out[ ]:

0.83982461599445

Customer Purchase Prediction & Effect of Micro-Numerosity

Watch Video Tutorial

Ask Doubt in FREE Live QnA Session

# Step 1 : import library

import pandas as pd

# Step 2 : import data

purchase = pd.read\_csv('https://github.com/YBIFoundation/Dataset/raw/main/Customer%20Purchase.csv')

purchase.head()

Customer ID Age Gender Education Review Purchased

0 1021 30 Female School Average No

1 1022 68 Female UG Poor No

2 1023 70 Female PG Good No

3 1024 72 Female PG Good No

4 1025 16 Female UG Average No

purchase.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 50 entries, 0 to 49

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Customer ID 50 non-null int64

1 Age 50 non-null int64

2 Gender 50 non-null object

3 Education 50 non-null object

4 Review 50 non-null object

5 Purchased 50 non-null object

dtypes: int64(2), object(4)

memory usage: 2.5+ KB

purchase.describe()

Customer ID Age

count 50.00000 50.000000

mean 1045.50000 54.160000

std 14.57738 25.658161

min 1021.00000 15.000000

25% 1033.25000 30.250000

50% 1045.50000 57.000000

75% 1057.75000 74.000000

max 1070.00000 98.000000

# Step 3 : define target (y) and features (X)

purchase.columns

Index(['Customer ID', 'Age', 'Gender', 'Education', 'Review', 'Purchased'], dtype='object')

y = purchase['Purchased']

X = purchase.drop(['Purchased','Customer ID'],axis=1)

# encoding categorical variable

X.replace({'Review':{'Poor':0,'Average':1,'Good':2}},inplace=True)

X.replace({'Education':{'School':0,'UG':1,'PG':2}},inplace=True)

X.replace({'Gender':{'Male': 0,'Female':1}},inplace=True)

# display first 5 rows

X.head()

Age Gender Education Review

0 30 1 0 1

1 68 1 1 0

2 70 1 2 2

3 72 1 2 2

4 16 1 1 1

# Step 4 : train test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.8, random\_state=2529)

# check shape of train and test sample

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

((40, 4), (10, 4), (40,), (10,))

# Step 5 : select model

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

# Step 6 : train or fit model

model.fit(X\_train,y\_train)

RandomForestClassifier()

# Step 7 : predict model

y\_pred = model.predict(X\_test)

y\_pred

array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes'],

dtype=object)

# Step 8 : model accuracy

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

confusion\_matrix(y\_test,y\_pred)

array([[2, 1],

[4, 3]])

accuracy\_score(y\_test,y\_pred)

0.5

print(classification\_report(y\_test,y\_pred))

precision recall f1-score support

No 0.33 0.67 0.44 3

Yes 0.75 0.43 0.55 7

accuracy 0.50 10

macro avg 0.54 0.55 0.49 10

weighted avg 0.62 0.50 0.52 10

Credit Card Default Prediction

The data set consists of 2000 samples from each of two categories. Five variables are

Income

Age

Loan

Loan to Income (engineered feature)

Default

Watch Video Tutorial

Ask Doubt in FREE Live QnA Session

# Step 1 : import library

import pandas as pd

# Step 2 : import data

default = pd.read\_csv('https://github.com/ybifoundation/Dataset/raw/main/Credit%20Default.csv')

default.head()

Income Age Loan Loan to Income Default

0 66155.92510 59.017015 8106.532131 0.122537 0

1 34415.15397 48.117153 6564.745018 0.190752 0

2 57317.17006 63.108049 8020.953296 0.139940 0

3 42709.53420 45.751972 6103.642260 0.142911 0

4 66952.68885 18.584336 8770.099235 0.130990 1

default.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2000 entries, 0 to 1999

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Income 2000 non-null float64

1 Age 2000 non-null float64

2 Loan 2000 non-null float64

3 Loan to Income 2000 non-null float64

4 Default 2000 non-null int64

dtypes: float64(4), int64(1)

memory usage: 78.2 KB

default.describe()

Income Age Loan Loan to Income Default

count 2000.000000 2000.000000 2000.000000 2000.000000 2000.000000

mean 45331.600018 40.927143 4444.369695 0.098403 0.141500

std 14326.327119 13.262450 3045.410024 0.057620 0.348624

min 20014.489470 18.055189 1.377630 0.000049 0.000000

25% 32796.459720 29.062492 1939.708847 0.047903 0.000000

50% 45789.117310 41.382673 3974.719418 0.099437 0.000000

75% 57791.281670 52.596993 6432.410625 0.147585 0.000000

max 69995.685580 63.971796 13766.051240 0.199938 1.000000

# Count of each category

default['Default'].value\_counts()

0 1717

1 283

Name: Default, dtype: int64

# Step 3 : define target (y) and features (X)

default.columns

Index(['Income', 'Age', 'Loan', 'Loan to Income', 'Default'], dtype='object')

y = default['Default']

X = default.drop(['Default'],axis=1)

# Step 4 : train test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

# check shape of train and test sample

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

((1400, 4), (600, 4), (1400,), (600,))

# Step 5 : select model

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

# Step 6 : train or fit model

model.fit(X\_train,y\_train)

LogisticRegression

LogisticRegression()

model.intercept\_

array([9.39569095])

model.coef\_

array([[-2.31410016e-04, -3.43062682e-01, 1.67863323e-03,

1.51188530e+00]])

# Step 7 : predict model

y\_pred = model.predict(X\_test)

y\_pred

array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0])

# Step 8 : model accuracy

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

confusion\_matrix(y\_test,y\_pred)

array([[506, 13],

[ 17, 64]])

accuracy\_score(y\_test,y\_pred)

0.95

print(classification\_report(y\_test,y\_pred))

precision recall f1-score support

0 0.97 0.97 0.97 519

1 0.83 0.79 0.81 81

accuracy 0.95 600

macro avg 0.90 0.88 0.89 600

weighted avg 0.95 0.95 0.95 600