SWE485: Selected Topics in Software Engineering
Software Engineering Department
King Saud University
3rd term 1444

Heart Attack Analysis & Prediction using Machine Learning Algorithms

Group Members		
#	**Name**	**ID**
1	Najd Awadh Almutairi	441200520
2	Ruyuf Albarrak	439200293
3	Khawlah Alghanim	441201130

Group Number	3
Section Number	74984
Supervisor	Dr. Manal BinKhonain

Table of Contents

Phase 1:

1. Introduction

The dataset we have chosen is a heart attack analysis & prediction dataset, We have chosen this dataset since the correct prediction of heart attacks can prevent life threats, and incorrect prediction can prove to be fatal at the same time.

2. The goal of Choosing the dataset

The dataset provides a list of values such as: age, sex, blood pressure, cholesterol level, chest pain and some other attributes. The goal of choosing this dataset is to predict the chance of heart attack by analyzing the relationship between the patient attributes and the target variable, which is binary outcome, so: 0 = less chance of heart attack and 1= more chance of heart attack by applying machine learning techniques.

3. Machine learning Tasks

Since the class label in the dataset "output" is known, therefore our problem is a supervised machine learning problem. And since some values of the class label are binary values (zero or one), therefore, our problem is a classification problem because the problem requires predicting a target. For that, we will use a supervised machine learning classification algorithm to predict whether it has a chance of a heart attack or not based on the values of some attributes.

Supervised learning

To predict whether the there is a chance of heart attack or not, we will use the following machine learning algorithms:

• Logistic Regression algorithm
• Decision Tree algorithm

4. Data

a. Kind of data:

- Heart Attack Analysis & Prediction Dataset contains information indicate if the person has more chance of heart attack compared with normal person.

b. Data source:

- We got the dataset from Kaggle. Dataset URL: https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

c. Data exploration:

1) Number of observations: our data set contains 303 rows and 14 columns
2) Describe the meaning of each variable
   a) Age : Age of the patient
   b) Sex : Sex of the patient (1 = male; 0 = female)
   c) exang: exercise induced angina (1 = yes; 0 = no)
   d) caa: number of major vessels (0-3)
   e) cp : Chest Pain type chest pain type
      i) Value 1: typical angina
      ii) Value 2: atypical angina
      iii) Value 3: non-anginal pain
      iv) Value 4: asymptomatic
   f) trtbps : resting blood pressure (in mm Hg)
   g) chol : cholesterol in mg/dl fetched via BMI sensor
   h) fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
   i) rest_ecg : resting electrocardiographic results
      i) Value 0: normal.
      ii) Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
      iii) Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria.
   j) thalach : maximum heart rate achieved
   k) old peak: ST depression induced by exercise relative to rest
   l) thall : thalassemia which is an inherited blood disorder that causes your body to have less hemoglobin than normal.
      i) 0 = null
      ii) 1 = fixed defect
      iii) 2 = normal
      iv) 3 = reversable defect
   m) output: 0= less chance of heart attack 1= more chance of heart attack

3) Number of variables and data types:

By using **dtypes** which we can find it in **panda** library we can see the data type of each variable in dataset.

```
data.dtypes
```

```
[9]: age           int64
     sex           int64
     cp            int64
     trtbps        int64
     chol          int64
     fbs           int64
     restecg       int64
     thalachh      int64
     exng          int64
     oldpeak     float64
     slp           int64
     caa           int64
     thall         int64
     output        int64
     dtype: object
```

Figure 1: Data type of variable in dataset

d. Sample of raw dataset:

Here we extract some sample of the dataset by using **head()** function in **panda** library

```
[12]: data.head()
```

[12]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 2: some rows of dataset

e. Variables distribution:

i. Distribution plot

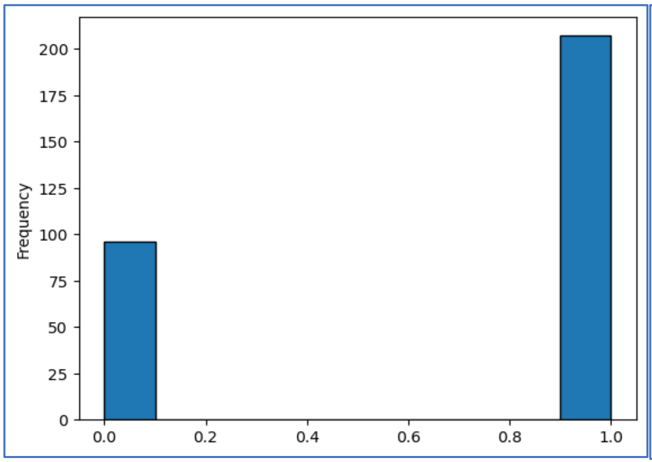


Figure 3: Sex Variable Distribution Plot

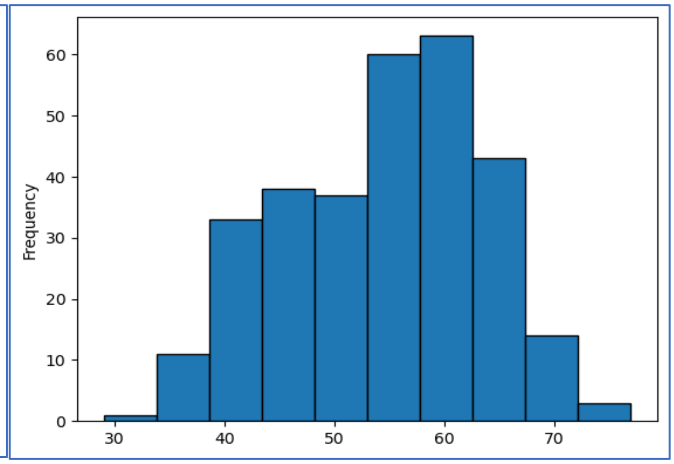


Figure 4: Age Variable Distribution plot

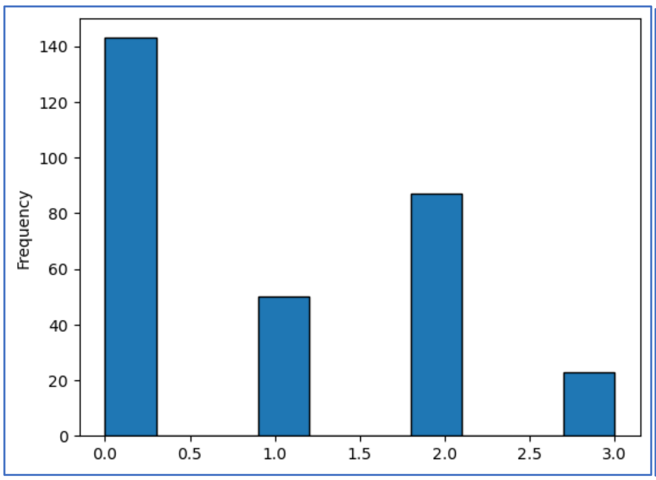


Figure 5: Cp Variable Distribution Plot

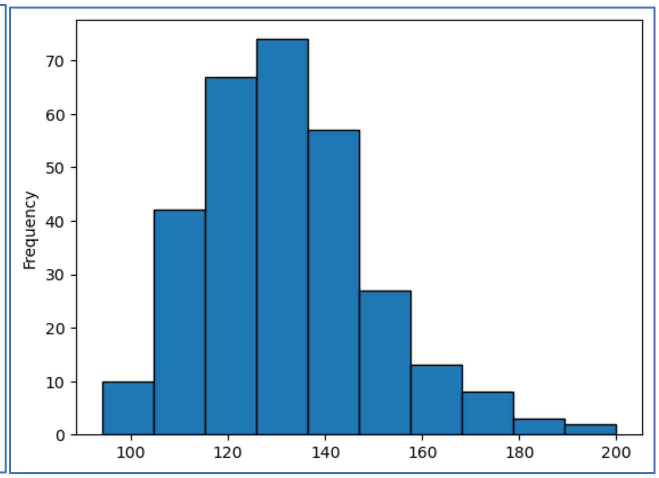


Figure 6: Trtbps Variable Distribution plot

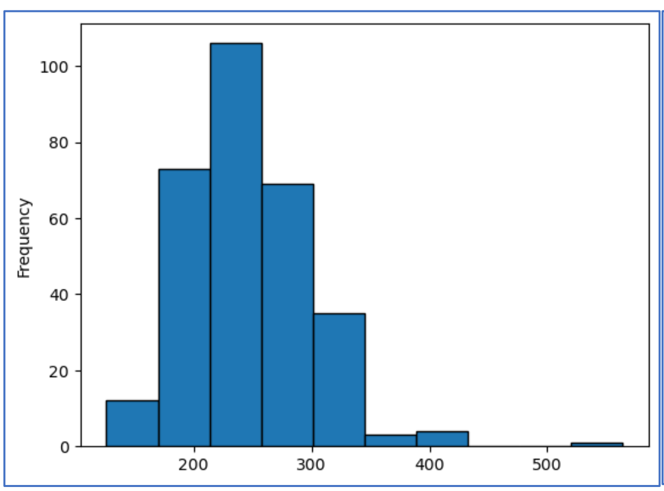


Figure 7: Chol Variable Distribution Plot

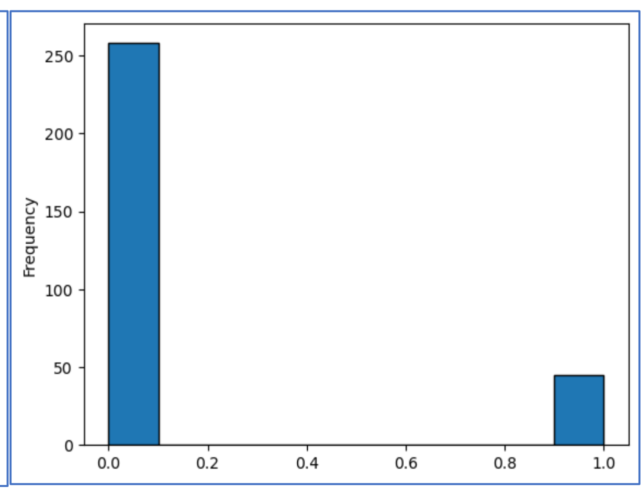


Figure 8: Fbs Variable Distribution Plot

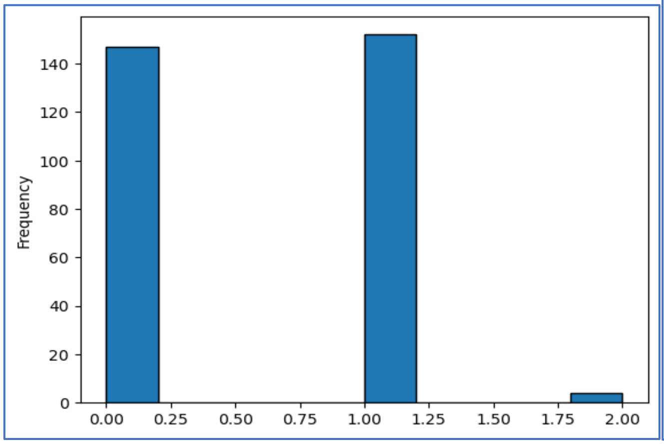


Figure 9: restecg Variable Distribution Plot

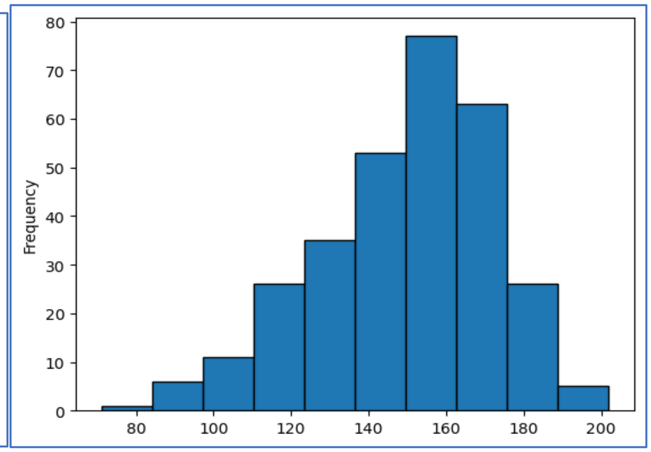


Figure 10: thalachh Variable Distribution Plot

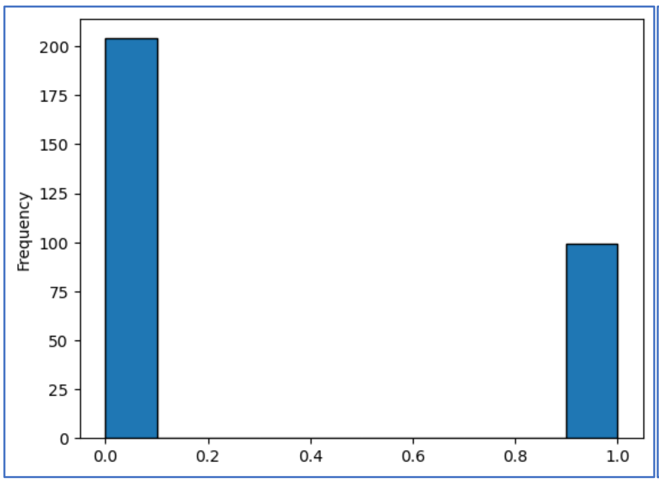


Figure 11: exng Variable Distribution Plot

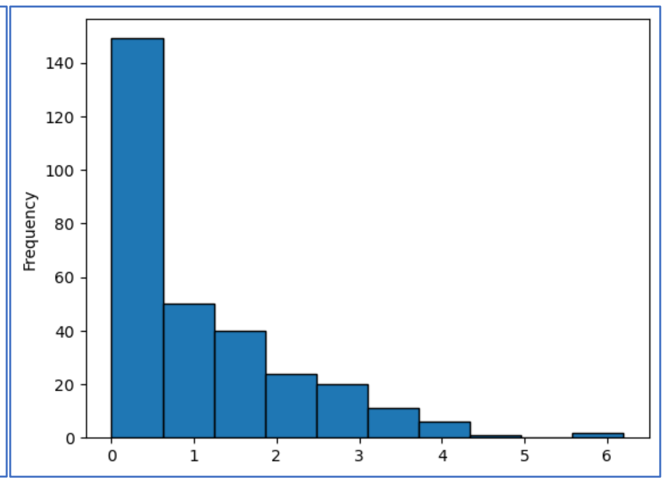


Figure 12:old peak Variable Distribution Plot

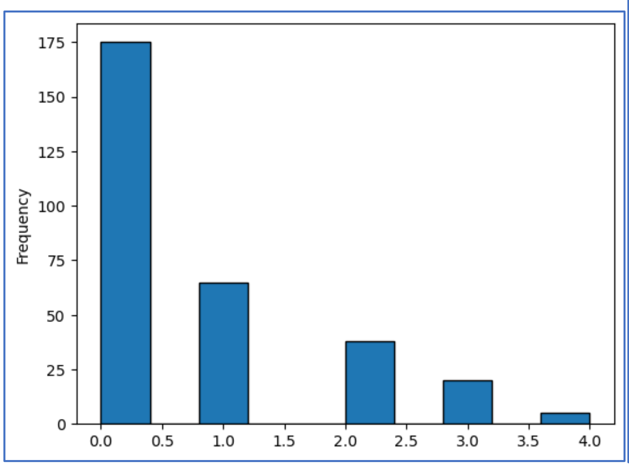


Figure 13: Caa Variable Distribution Plot

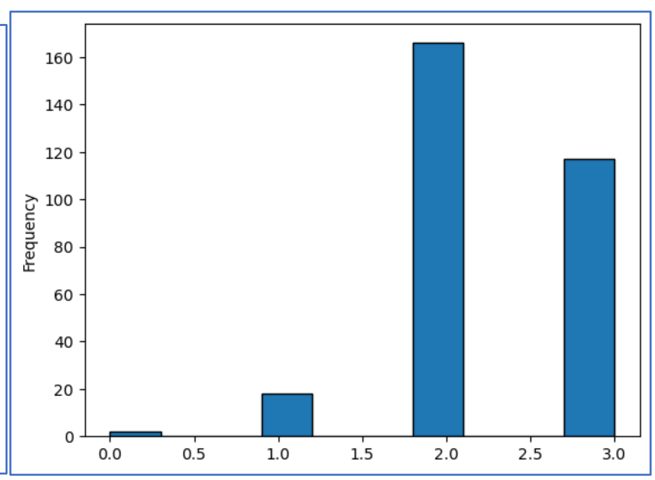


Figure 14: thall Variable Distribution Plot

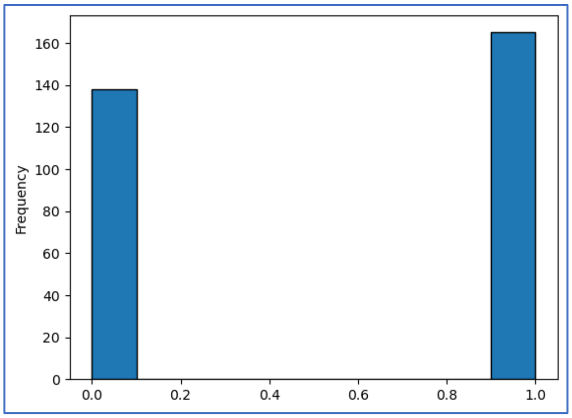


Figure 15: Output Variable Distribution Plot

ii. Pie chart of output types:

***output: 0= less chance of heart attack 1= more chance of heart attack**

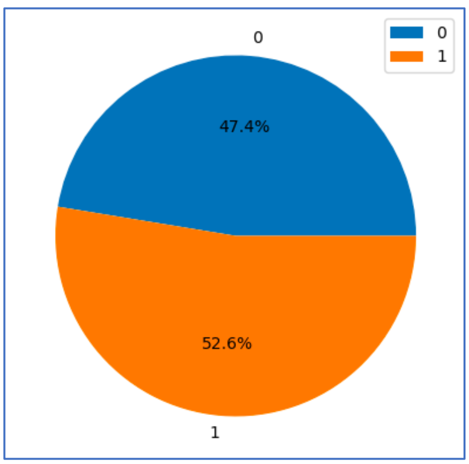


Figure 16: output pie chart

f. Missing values:

We check if the dataset have null or missing values by using isnull() function which is returns the number of missing values in the dataset.

```
data.isnull()
```

[14]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	False	False	False	False	False	False	False	False
299	False	False	False	False	False	False	False	False	False	False	False	False	False	False
300	False	False	False	False	False	False	False	False	False	False	False	False	False	False
301	False	False	False	False	False	False	False	False	False	False	False	False	False	False
302	False	False	False	False	False	False	False	False	False	False	False	False	False	False

303 rows × 14 columns

Figure 17: missing values

g. Statistical summaries:

Here we can see the statistical summaries we use describe function is used to get a descriptive statistics summary. This includes mean, count, std deviation, percentiles, and min-max values of all the features. We discover that mean of age is ~ 54 year old.

[15]:
```
data.describe()
```

[15]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

Figure 18: statistical summaries of all except variation

```
data.var()
```

```
[4]: age            81.865757
     sex             0.217553
     cp              1.065114
     trtbps        308.472817
     chol         2678.423588
     fbs             0.127225
     restecg         0.276705
     thalachh      524.571561
     exng            0.221084
     oldpeak         1.348971
     slp             0.379794
     caa             1.013542
     thall           0.375800
     output          0.248971
     dtype: float64
```

Figure 19: statistical summaries of variation

5. Data preprocessing

We deeply check our dataset to decide what techniques we need to apply. Because all variables in our data are numeral, we didn't need to do the variable transformation. Also, because our data was already classified into categorical attributes, we didn't need to do the discretization. Moreover. Because most of the variables in our data are of type integer, we didn't need to do the normalization.

Data cleaning:

The dataset didn't contain a null value but there is one duplicate in row 164 so we removed it.

```
duplic= data.duplicated()

print(data[duplic])
```

```
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
164   38    1   2     138   175    0        1       173     0      0.0    2

     caa  thall  output
164    4      2       1
```

+ Code   + Markdown

Figure 20: Check the duplicate in dataset

We use this code to remove duplicate row.

```
data= data.drop_duplicates()
```

Figure 21: remove duplicate row code

And this our data after remove row 164.

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
10	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
12	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
13	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
15	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
16	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
19	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
20	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
21	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
22	42	1	0	140	226	0	1	178	0	0.0	2	0	2	1
23	61	1	2	150	243	1	1	137	1	1.0	1	0	2	1
24	40	1	3	140	199	0	1	178	1	1.4	2	0	3	1
25	71	0	1	160	302	0	1	162	0	0.4	2	2	2	1

26	59	1	2	150	212	1	1	157	0	1.6	2	0	2	1
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1
28	65	0	2	140	417	1	0	157	0	0.8	2	1	2	1
29	53	1	2	130	197	1	0	152	0	1.2	0	0	2	1
30	41	0	1	105	198	0	1	168	0	0.0	2	1	2	1
31	65	1	0	120	177	0	1	140	0	0.4	2	0	3	1
32	44	1	1	130	219	0	0	188	0	0.0	2	0	2	1
33	54	1	2	125	273	0	0	152	0	0.5	0	1	2	1
34	51	1	3	125	213	0	0	125	1	1.4	2	1	2	1
35	46	0	2	142	177	0	0	160	1	1.4	0	0	2	1
36	54	0	2	135	304	1	1	170	0	0.0	2	0	2	1
37	54	1	2	150	232	0	0	165	0	1.6	2	0	3	1
38	65	0	2	155	269	0	1	148	0	0.8	2	0	2	1
39	65	0	2	160	360	0	0	151	0	0.8	2	0	2	1
40	51	0	2	140	308	0	0	142	0	1.5	2	1	2	1
41	48	1	1	130	245	0	0	180	0	0.2	1	0	2	1
42	45	1	0	104	208	0	0	148	1	3.0	1	0	2	1
43	53	0	0	130	264	0	0	143	0	0.4	1	0	2	1
44	39	1	2	140	321	0	0	182	0	0.0	2	0	2	1
45	52	1	1	120	325	0	1	172	0	0.2	2	0	2	1
46	44	1	2	140	235	0	0	180	0	0.0	2	0	2	1
47	47	1	2	138	257	0	0	156	0	0.0	2	0	2	1
48	53	0	2	128	216	0	0	115	0	0.0	2	0	0	1
49	53	0	0	138	234	0	0	160	0	0.0	2	0	2	1
50	51	0	2	130	256	0	0	149	0	0.5	2	0	2	1
51	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1
52	62	1	2	130	231	0	1	146	0	1.8	1	3	3	1
53	44	0	2	108	141	0	1	175	0	0.6	1	0	2	1
54	63	0	2	135	252	0	0	172	0	0.0	2	0	2	1
55	52	1	1	134	201	0	1	158	0	0.8	2	1	2	1
56	48	1	0	122	222	0	0	186	0	0.0	2	0	2	1
57	45	1	0	115	260	0	0	185	0	0.0	2	0	2	1
58	34	1	3	118	182	0	0	174	0	0.0	2	0	2	1
59	57	0	0	128	303	0	0	159	0	0.0	2	1	2	1
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2	1
61	54	1	1	108	309	0	1	156	0	0.0	2	0	3	1
62	52	1	3	118	186	0	0	190	0	0.0	1	0	1	1
63	41	1	1	135	203	0	1	132	0	0.0	1	0	1	1
64	58	1	2	140	211	1	0	165	0	0.0	2	0	2	1
65	35	0	0	138	183	0	1	182	0	1.4	2	0	2	1
66	51	1	2	100	222	0	1	143	1	1.2	1	0	2	1
67	45	0	1	130	234	0	0	175	0	0.6	1	0	2	1
68	44	1	1	120	220	0	1	170	0	0.0	2	0	2	1
69	62	0	0	124	209	0	1	163	0	0.0	2	0	2	1
70	54	1	2	120	258	0	0	147	0	0.4	1	0	3	1
71	51	1	2	94	227	0	1	154	1	0.0	2	1	3	1
72	29	1	1	130	204	0	0	202	0	0.0	2	0	2	1
73	51	1	0	140	261	0	0	186	1	0.0	2	0	2	1
74	43	0	2	122	213	0	1	165	0	0.2	1	0	2	1
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2	1
76	51	1	2	125	245	1	0	166	0	2.4	1	0	2	1
77	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
78	52	1	1	128	205	1	1	184	0	0.0	2	0	2	1
79	58	1	2	105	240	0	0	154	1	0.6	1	0	3	1
80	41	1	2	112	250	0	1	179	0	0.0	2	0	2	1
81	45	1	1	128	308	0	0	170	0	0.0	2	0	2	1
82	60	0	2	102	318	0	1	160	0	0.0	2	1	2	1
83	52	1	3	152	298	1	1	178	0	1.2	1	0	3	1
84	42	0	0	102	265	0	0	122	0	0.6	1	0	2	1
85	67	0	2	115	564	0	0	160	0	1.6	1	0	3	1
86	68	1	2	118	277	0	1	151	0	1.0	2	1	3	1
87	46	1	1	101	197	1	1	156	0	0.0	2	0	3	1
88	54	0	2	110	214	0	1	158	0	1.6	1	0	2	1
89	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
90	48	1	2	124	255	1	1	175	0	0.0	2	2	2	1
91	57	1	0	132	207	0	1	168	1	0.0	2	0	3	1
92	52	1	2	138	223	0	1	169	0	0.0	2	4	2	1
93	54	0	1	132	288	1	0	159	1	0.0	2	1	2	1
94	45	0	1	112	160	0	1	138	0	0.0	1	0	2	1
95	53	1	0	142	226	0	0	111	1	0.0	2	0	3	1
96	62	0	0	140	394	0	0	157	0	1.2	1	0	2	1
97	52	1	0	108	233	1	1	147	0	0.1	2	3	3	1
98	43	1	2	130	315	0	1	162	0	1.9	2	1	2	1
99	53	1	2	130	246	1	0	173	0	0.0	2	3	2	1
100	42	1	3	148	244	0	0	178	0	0.8	2	2	2	1
101	59	1	3	178	270	0	0	145	0	4.2	0	0	3	1
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2	1
103	42	1	2	120	240	1	1	194	0	0.8	0	0	3	1
104	50	1	2	129	196	0	1	163	0	0.0	2	0	2	1
105	68	0	2	120	211	0	0	115	0	1.5	1	0	2	1
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2	1

107	45	0	0	138	236	0	0	152	1	0.2	1	0	2	1
108	50	0	1	120	244	0	1	162	0	1.1	2	0	2	1
109	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
110	64	0	0	180	325	0	1	154	1	0.0	2	0	2	1
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3	1
112	64	0	2	140	313	0	1	133	0	0.2	2	0	3	1
113	43	1	0	110	211	0	1	161	0	0.0	2	0	3	1
114	55	1	1	130	262	0	1	155	0	0.0	2	0	2	1
115	37	0	2	120	215	0	1	170	0	0.0	2	0	2	1
116	41	1	2	130	214	0	0	168	0	2.0	1	0	2	1
117	56	1	3	120	193	0	0	162	0	1.9	1	0	3	1
118	46	0	1	105	204	0	1	172	0	0.0	2	0	2	1
119	46	0	0	138	243	0	0	152	1	0.0	1	0	2	1
120	64	0	0	130	303	0	1	122	0	2.0	1	2	2	1
121	59	1	0	138	271	0	0	182	0	0.0	2	0	2	1
122	41	0	2	112	268	0	0	172	1	0.0	2	0	2	1
123	54	0	2	108	267	0	0	167	0	0.0	2	0	2	1
124	39	0	2	94	199	0	1	179	0	0.0	2	0	2	1
125	34	0	1	118	210	0	1	192	0	0.7	2	0	2	1
126	47	1	0	112	204	0	1	143	0	0.1	2	0	2	1
127	67	0	2	152	277	0	1	172	0	0.0	2	1	2	1
128	52	0	2	136	196	0	0	169	0	0.1	1	0	2	1
129	74	0	1	120	269	0	0	121	1	0.2	2	1	2	1
130	54	0	2	160	201	0	1	163	0	0.0	2	1	2	1
131	49	0	1	134	271	0	1	162	0	0.0	1	0	2	1
132	42	1	1	120	295	0	1	162	0	0.0	2	0	2	1
133	41	1	1	110	235	0	1	153	0	0.0	2	0	2	1
134	41	0	1	126	306	0	1	163	0	0.0	2	0	2	1
135	49	0	0	130	269	0	1	163	0	0.0	2	0	2	1
136	60	0	2	120	178	1	1	96	0	0.0	2	0	2	1
137	62	1	1	128	208	1	0	140	0	0.0	2	0	2	1
138	57	1	0	110	201	0	1	126	1	1.5	1	0	1	1
139	64	1	0	128	263	0	1	105	1	0.2	1	1	3	1
140	51	0	2	120	295	0	0	157	0	0.6	2	0	2	1
141	43	1	0	115	303	0	1	181	0	1.2	1	0	2	1
142	42	0	2	120	209	0	1	173	0	0.0	1	0	2	1
143	67	0	0	106	223	0	1	142	0	0.3	2	2	2	1
144	76	0	2	140	197	0	2	116	0	1.1	1	0	2	1
145	70	1	1	156	245	0	0	143	0	0.0	2	0	2	1
146	44	0	2	118	242	0	1	149	0	0.3	1	1	2	1
147	60	0	3	150	240	0	1	171	0	0.9	2	0	2	1
148	44	1	2	120	226	0	1	169	0	0.0	2	0	2	1
149	42	1	2	130	180	0	1	150	0	0.0	2	0	2	1
150	66	1	0	160	228	0	0	138	0	2.3	2	0	1	1
151	71	0	0	112	149	0	1	125	0	1.6	1	0	2	1
152	64	1	3	170	227	0	0	155	0	0.6	1	0	3	1
153	66	0	2	146	278	0	0	152	0	0.0	1	1	2	1
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1
155	58	0	0	130	197	0	1	131	0	0.6	1	0	2	1
156	47	1	2	130	253	0	1	179	0	0.0	2	0	2	1
157	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1
158	58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
159	56	1	1	130	221	0	0	163	0	0.0	2	0	3	1
160	56	1	1	120	240	0	1	169	0	0.0	0	0	2	1
161	55	0	1	132	342	0	1	166	0	1.2	2	0	2	1
162	41	1	1	120	157	0	1	182	0	0.0	2	0	2	1
163	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1
165	67	1	0	160	286	0	0	108	1	1.5	1	3	2	0
166	67	1	0	120	229	0	0	129	1	2.6	1	2	3	0
167	62	0	0	140	268	0	0	160	0	3.6	0	2	2	0
168	63	1	0	130	254	0	0	147	0	1.4	1	1	3	0
169	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
170	56	1	2	130	256	1	0	142	1	0.6	1	1	1	0
171	48	1	1	110	229	0	1	168	0	1.0	0	0	3	0
172	58	1	1	120	284	0	0	160	0	1.8	1	0	2	0
173	58	1	2	132	224	0	0	173	0	3.2	2	2	3	0
174	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0
175	40	1	0	110	167	0	0	114	1	2.0	1	0	3	0
176	60	1	0	117	230	1	1	160	1	1.4	2	2	3	0
177	64	1	2	140	335	0	1	158	0	0.0	2	0	2	0
178	43	1	0	120	177	0	0	120	1	2.5	1	0	3	0
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1	0
180	55	1	0	132	353	0	1	132	1	1.2	1	1	3	0
181	65	0	0	150	225	0	0	114	0	1.0	1	3	3	0
182	61	0	0	130	330	0	0	169	0	0.0	2	0	2	0
183	58	1	2	112	230	0	0	165	0	2.5	1	1	3	0
184	50	1	0	150	243	0	0	128	0	2.6	1	0	3	0
185	44	1	0	112	290	0	0	153	0	0.0	2	1	2	0
186	60	1	0	130	253	0	1	144	1	1.4	2	1	3	0
187	54	1	0	124	266	0	0	109	1	2.2	1	1	3	0
188	50	1	2	140	233	0	1	163	0	0.6	1	1	3	0

189	41	1	0	110	172	0	0	158	0	0.0	2	0	3	0
190	51	0	0	130	305	0	1	142	1	1.2	1	0	3	0
191	58	1	0	128	216	0	0	131	1	2.2	1	3	3	0
192	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3	0
194	60	1	2	140	185	0	0	155	0	3.0	1	0	2	0
195	59	1	0	170	326	0	0	140	1	3.4	0	0	3	0
196	46	1	2	150	231	0	1	147	0	3.6	1	0	2	0
197	67	1	0	125	254	1	1	163	0	0.2	1	2	3	0
198	62	1	0	120	267	0	1	99	1	1.8	1	2	3	0
199	65	1	0	110	248	0	0	158	0	0.6	2	2	1	0
200	44	1	0	110	197	0	0	177	0	0.0	2	1	2	0
201	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
202	58	1	0	150	270	0	0	111	1	0.8	2	0	3	0
203	68	1	2	180	274	1	0	150	1	1.6	1	0	3	0
204	62	0	0	160	164	0	0	145	0	6.2	0	3	3	0
205	52	1	0	128	255	0	1	161	1	0.0	2	1	3	0
206	59	1	0	110	239	0	0	142	1	1.2	1	1	3	0
207	60	0	0	150	258	0	0	157	0	2.6	1	2	3	0
208	49	1	2	120	188	0	1	139	0	2.0	1	3	3	0
209	59	1	0	140	177	0	1	162	1	0.0	2	1	3	0
210	57	1	2	128	229	0	0	150	0	0.4	1	1	3	0
211	61	1	0	120	260	0	1	140	1	3.6	1	1	3	0
212	39	1	0	118	219	0	1	140	0	1.2	1	0	3	0
213	61	0	0	145	307	0	0	146	1	1.0	1	0	3	0
214	56	1	0	125	249	1	0	144	1	1.2	1	1	2	0
215	43	0	0	132	341	1	0	136	1	3.0	1	0	3	0
216	62	0	2	130	263	0	1	97	0	1.2	1	1	3	0
217	63	1	0	130	330	1	0	132	1	1.8	2	3	3	0
218	65	1	0	135	254	0	0	127	0	2.8	1	1	3	0
219	48	1	0	130	256	1	0	150	1	0.0	2	2	3	0
220	63	0	0	150	407	0	0	154	0	4.0	1	3	3	0
221	55	1	0	140	217	0	1	111	1	5.6	0	0	3	0
222	65	1	3	138	282	1	0	174	0	1.4	1	1	2	0
223	56	0	0	200	288	1	0	133	1	4.0	0	2	3	0
224	54	1	0	110	239	0	1	126	1	2.8	1	1	3	0
225	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
226	62	1	1	120	281	0	0	103	0	1.4	1	1	3	0
227	35	1	0	120	198	0	1	130	1	1.6	1	0	3	0
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3	0
229	64	1	2	125	309	0	1	131	1	1.8	1	0	3	0
230	47	1	2	108	243	0	1	152	0	0.0	2	0	2	0
231	57	1	0	165	289	1	0	124	0	1.0	1	3	3	0
232	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
233	64	1	0	120	246	0	0	96	1	2.2	0	1	2	0
234	70	1	0	130	322	0	0	109	0	2.4	1	3	2	0
235	51	1	0	140	299	0	1	173	1	1.6	2	0	3	0
236	58	1	0	125	300	0	0	171	0	0.0	2	2	3	0
237	60	1	0	140	293	0	0	170	0	1.2	1	2	3	0
238	77	1	0	125	304	0	0	162	1	0.0	2	3	2	0
239	35	1	0	126	282	0	0	156	1	0.0	2	0	3	0
240	70	1	2	160	269	0	1	112	1	2.9	1	1	3	0
241	59	0	0	174	249	0	1	143	1	0.0	1	0	2	0
242	64	1	0	145	212	0	0	132	0	2.0	1	2	1	0
243	57	1	0	152	274	0	1	88	1	1.2	1	1	3	0
244	56	1	0	132	184	0	0	105	1	2.1	1	1	1	0
245	48	1	0	124	274	0	0	166	0	0.5	1	0	3	0
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3	0
247	66	1	1	160	246	0	1	120	1	0.0	1	3	1	0
248	54	1	1	192	283	0	0	195	0	0.0	2	1	3	0
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3	0
250	51	1	0	140	298	0	1	122	1	4.2	1	3	3	0
251	43	1	0	132	247	1	0	143	1	0.1	1	4	3	0
252	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
253	67	1	0	100	299	0	0	125	1	0.9	1	2	2	0
254	59	1	3	160	273	0	0	125	0	0.0	2	0	2	0
255	45	1	0	142	309	0	0	147	1	0.0	1	3	3	0
256	58	1	0	128	259	0	0	130	1	3.0	1	2	3	0
257	50	1	0	144	200	0	0	126	1	0.9	1	0	3	0
258	62	0	0	150	244	0	1	154	1	1.4	1	0	2	0
259	38	1	3	120	231	0	1	182	1	3.8	1	0	3	0
260	66	0	0	178	228	1	1	165	1	1.0	1	2	3	0
261	52	1	0	112	230	0	1	160	0	0.0	2	1	2	0
262	53	1	0	123	282	0	1	95	1	2.0	1	2	3	0
263	63	0	0	108	269	0	1	169	1	1.8	1	2	2	0
264	54	1	0	110	206	0	0	108	1	0.0	1	1	2	0
265	66	1	0	112	212	0	0	132	1	0.1	2	1	2	0
266	55	0	0	180	327	0	2	117	1	3.4	1	0	2	0
267	49	1	2	118	149	0	0	126	0	0.8	2	3	2	0
268	54	1	0	122	286	0	0	116	1	3.2	1	2	2	0
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3	0

270	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
271	61	1	3	134	234	0	1	145	0	2.6	1	2	2	0
272	67	1	0	120	237	0	1	71	0	1.0	1	0	2	0
273	58	1	0	100	234	0	1	156	0	0.1	2	1	3	0
274	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
275	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
276	58	1	0	146	218	0	1	105	0	2.0	1	1	3	0
277	57	1	1	124	261	0	1	141	0	0.3	2	0	3	0
278	58	0	1	136	319	1	0	152	0	0.0	2	2	2	0
279	61	1	0	138	166	0	0	125	1	3.6	1	1	2	0
280	42	1	0	136	315	0	1	125	1	1.8	1	0	1	0
281	52	1	0	128	204	1	1	156	1	1.0	1	0	0	0
282	59	1	2	126	218	1	1	134	0	2.2	1	1	1	0
283	40	1	0	152	223	0	1	181	0	0.0	2	0	3	0
284	61	1	0	140	207	0	0	138	1	1.9	2	1	3	0
285	46	1	0	140	311	0	1	120	1	1.8	1	2	3	0
286	59	1	3	134	204	0	1	162	0	0.8	2	2	2	0
287	57	1	1	154	232	0	0	164	0	0.0	2	1	2	0
288	57	1	0	110	335	0	1	143	1	3.0	1	1	3	0
289	55	0	0	128	205	0	2	130	1	2.0	1	1	3	0
290	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
291	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
292	58	0	0	170	225	1	0	146	1	2.8	1	2	1	0
293	67	1	2	152	212	0	0	150	0	0.8	1	0	3	0
294	44	1	0	120	169	0	1	144	1	2.8	0	0	1	0
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2	0
297	59	1	0	164	176	1	0	90	0	1.0	1	2	1	0
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

Phase 2:

1. Supervised Learning

To predict whether it has a chance of a heart attack or not, we will use the following machine learning algorithms:

a. Logistic Regression algorithm

The Logistic Regression is a supervised linear classification algorithm. It can be used to classify objects of binary and multi-class problems.

b. Decision Tree algorithm

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.[1]

To evaluate the performance of logistic regression and decision tree algorithms in predicting the chance of a heart attack, we will divide the dataset into two sets:

Training set: this data will constitute 70% of the dataset and it will be used for training the machine learning models.

Test set: This data constitutes 30% of the data and it will be used for evaluating the performance of the machine learning algorithms. We will use the accuracy, precision, sensitivity, and specificity performance metrics to evaluate the performance of the logistic regression and decision tree algorithms.

2. Implementation of the Algorithms

We implemented the logistic regression and decision tree algorithms in this project using Python programming language. We used Google Colab. [2] Colab, or "Collaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Loading the dataset

The following figure shows the python code for loading the dataset. We used the function read csv in Panda's library [3] to load the dataset.

```
import pandas as pd
from sklearn import preprocessing as pre
import numpy as np
data = pd.read_csv('heartpreprocessed.csv')
```

Figure 22: import and load the dataset

Preparing the dataset to implement algorithms:

1- Separate the dataset into features ('age' , 'sex' , 'cp' , 'trtbps' , 'chol' , 'fbs' , 'restecg' , 'thalachh' ,'exng' , 'oldpeak' , 'slp' ,'caa' , 'thall') and output and then we store the **target variable in variable Y** and store the **features in the matrix X.**
2- Divide the dataset into training/testing with ratio 70/30

```
[16] from sklearn.model_selection import train_test_split
     # target Y , features X
     X= data[['age' , 'sex' , 'cp' , 'trtbps' , 'chol' , 'fbs' , 'restecg' , 'thalachh' ,
              'exng' , 'oldpeak' , 'slp' ,'caa' , 'thall']]
     Y = data['output']
     # train_test splitting that data to 80/20 train/test
     x_train , x_test , y_train , y_test = train_test_split(X ,Y , test_size=0.3, random_state = 2)
```

Figure 23: divide the data set to train and test

Implementation Logistic Regression

In this algorithm, we divided our dataset into training and testing data as shown in the code snapshot where 70% of the data set is training and 30% is testing:

```
[16] from sklearn.model_selection import train_test_split
     # target Y , features X
     X= data[['age' , 'sex' , 'cp' , 'trtbps' , 'chol' , 'fbs' , 'restecg' , 'thalachh' ,
              'exng' , 'oldpeak' , 'slp' ,'caa' , 'thall']]
     Y = data['output']
     # train_test splitting that data to 80/20 train/test
     x_train , x_test , y_train , y_test = train_test_split(X ,Y , test_size=0.3, random_state = 2)
```

Figure 24: divide dataset for Logistic Regression

Then, we plotted our graph comparing our training and testing sets' accuracies over many iterations to make more accurate result :

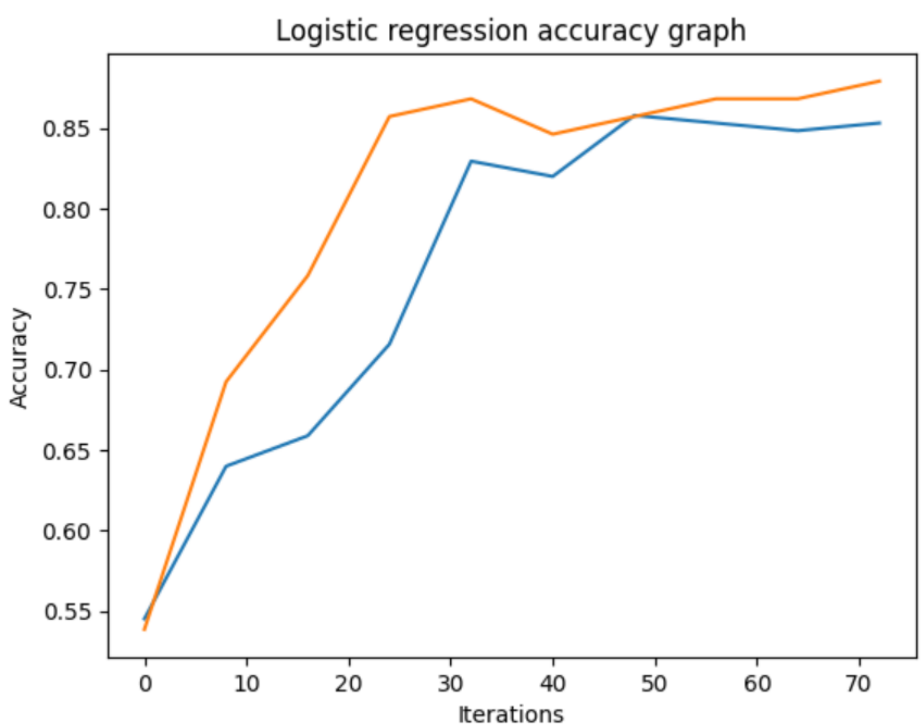
```
[44] # Train and validation scores initialized as empty list
     train_score = []
     test_score = []
     #Loop for taking average result
     Array=8*(np.arange(10))
     for i in Array:
         # Create LogisticRegression object and fit
         lr = LogisticRegression( max_iter=i)
         lr.fit(x_train, y_train)

         # Evalueate scores and append to lists
         train_score.append( lr.score(x_train, y_train))
         test_score.append( lr.score(x_test, y_test))

     # Plot results
     plt.ylabel('Accuracy')
     plt.xlabel('Iterations')
     plt.title('Logistic regression accuracy graph')
     plt.plot(Array,train_score)
     plt.plot(Array,test_score)
```

Figure 25: build a Logistic Regression model

In orange is our training accuracy and in blue is our testing accuracy, we can conclude that as the iterations increases both accuracies increases but the training data will have higher accuracy than testing data.



Next, we plotted a confusion matrix using test data to asses the accuracy of the classification. The rows correspond to the actual label for which the results were intended. And the columns correspond to predicted label.

```
from sklearn import metrics
y_pred=lr.predict(x_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
```

```
[42] class_names=[0,1] # name  of classes
     fig, ax = plt.subplots()
     tick_marks = np.arange(len(class_names))
     plt.xticks(tick_marks, class_names)
     plt.yticks(tick_marks, class_names)
     # create heatmap
     sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
     ax.xaxis.set_label_position("top")
     plt.tight_layout()
     plt.title('Confusion matrix', y=1.1)
     plt.ylabel('Actual label')
     plt.xlabel('Predicted label')
```

The output:

**output: 0= less chance of heart attack 1= more chance of heart attack*

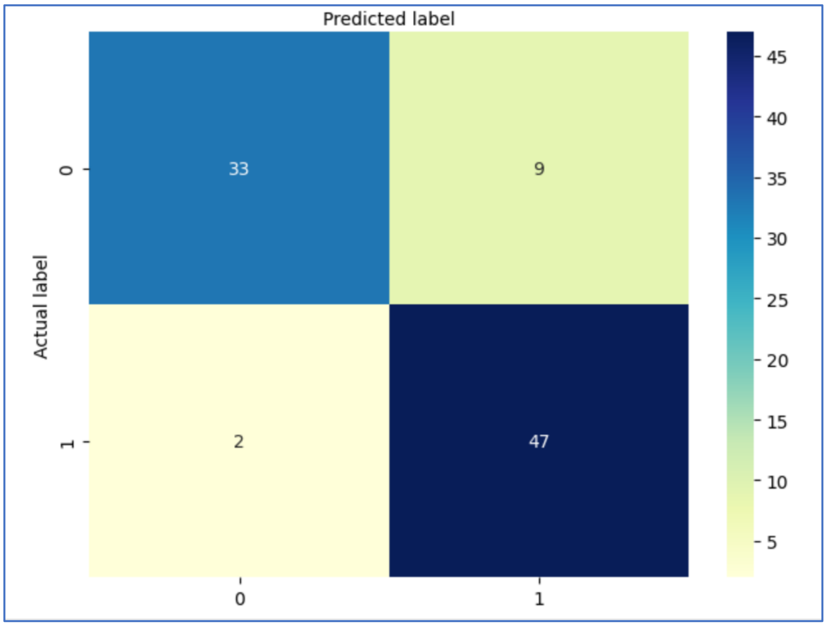


Figure 26: confusion matrix for Logistic Regression

Finally we calculated the CFM evaluation metrics Accuracy, Recall, f1-score using classification_report:

**output: 0= less chance of heart attack 1= more chance of heart attack*

```
[27] from sklearn.metrics import classification_report
     print(classification_report(y_pred,y_test))

              precision    recall  f1-score   support

           0       0.79      0.94      0.86        35
           1       0.96      0.84      0.90        56

    accuracy                           0.88        91
   macro avg       0.87      0.89      0.88        91
weighted avg       0.89      0.88      0.88        91
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Accuracy: 0.8791208791208791
```

Figure 27: report for Logistic Regression

Implementation Decision Tree

We create the decision tree with entropy and we pass two parameters random_state=100 and max_depth=5 then we trained the model using fit() function then, we performed the test to predict the output(the chance of heart attack) using predict() function passing the test data (x_test) to it. Finally, we calculate the accuracy using accuracy_score() method and generate a report using classification_report() method and cross validation score using cross_val_score() which runs 5-folds cross validation on our dataset to see if the model generalizes to the entire dataset to produce an array of estimator scores for each cross validation run.

**output: 0= less chance of heart attack 1= more chance of heart attack*

```
from sklearn.metrics import accuracy_score , classification_report
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
#  decision tree with entropy
clf_entropy = DecisionTreeClassifier(criterion="entropy" , random_state = 100 , max_depth=5)
# performing training
clf_entropy.fit(x_train , y_train )
# predicton on test with entropy
y_pred = clf_entropy.predict(x_test)

print("Accuracy :" ,accuracy_score(y_test , y_pred)*100)

print("Report :" ,classification_report(y_test , y_pred))

score= cross_val_score(clf_entropy , X , Y , cv=5)

print('Cross validation score :' , score)
```

Figure 28: build the decision tree model

```
Accuracy : 86.81318681318682
Report :               precision    recall  f1-score   support

           0       0.92      0.79      0.85        42
           1       0.84      0.94      0.88        49

    accuracy                           0.87        91
   macro avg       0.88      0.86      0.87        91
weighted avg       0.87      0.87      0.87        91

Cross validation score : [0.73770492 0.86885246 0.73333333 0.78333333 0.63333333]
```

Figure 29: report of decision tree model

After that we pass (x_train, y_train) and (x_test, y_test) to score() after training the model on the data.

```
# Train Accuracy
data_train_accurcy= clf_entropy.score(x_train , y_train)
print("Training accuracy =" , data_train_accurcy)
# Test Accuracy
data_test_accurcy = clf_entropy.score(x_test , y_test)
print("Testing accuracy =" , data_test_accurcy)

Training accuracy = 0.909952606635071
Testing accuracy = 0.8681318681318682
```

Figure 30: training and test decision tree model with the result of accuracy

Decision tree visual representation:

We represent Decision tree representation using plot_tree() function.

```python
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(12,12))
plot_tree(clf_entropy ,
                 feature_names = ['age' , 'sex' , 'cp' , 'trtbps' , 'chol' , 'fbs' , 'restecg' , 'thalachh' ,
          'exng' , 'oldpeak' , 'slp' ,'caa' , 'thall'],
          class_names=['0' , '1'],
          filled= True )
```

Figure 31: represent Decision tree code

And this the output:

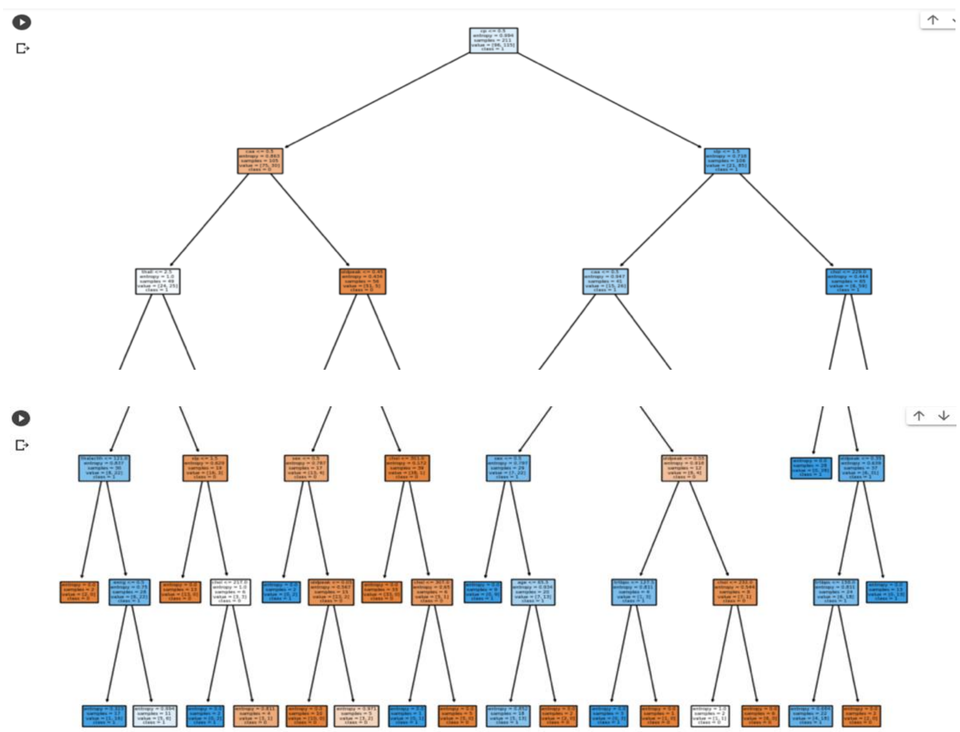


Figure 32: decision tree diagram

Finally, we use confusion_matrix() on the test data to assess the classification's accuracy and identify any model flaws. The rows correspond to the actual courses for which the results were intended. And the columns correspond to the predictions we've made.

```python
import seaborn as sns
# confusion matrix
from sklearn.metrics import confusion_matrix
y_pred = clf_entropy.predict(x_test)
y_true = y_test
ab = ['#3b5f91' , '#7daa6a'  ]
cm_dt = confusion_matrix(y_true , y_pred)
f , ax = plt.subplots(figsize=(10 , 5))
sns.heatmap(cm_dt , annot = True , linewidth = 0.1 , fmt =".0f" , cmap=ab , ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Predicted vs actual confusion matrix ")
plt.show()
```

Figure 33: plot the confutation matrix code

The output:

**output: 0= less chance of heart attack 1= more chance of heart attack*

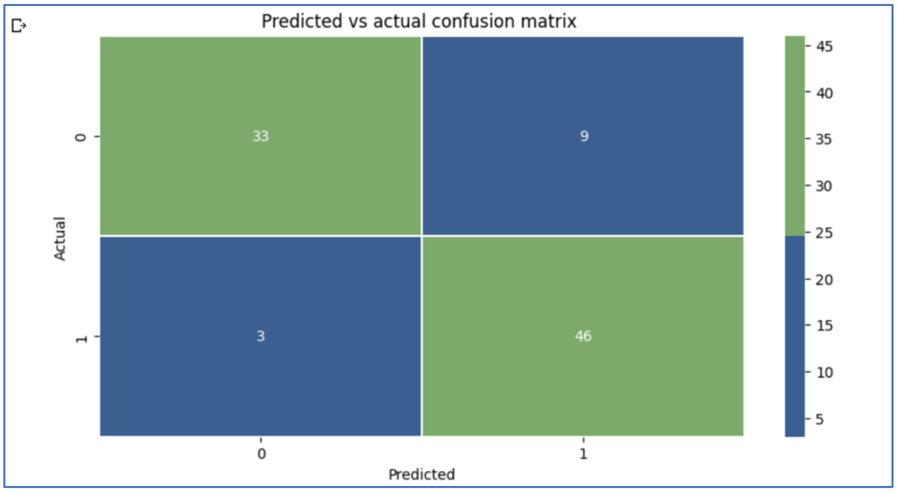


Figure 34: confusion matrix of decision tree

3. Result of Algorithms

Comparison between Logistic Regression and Decision tree in predicting the chance of a heart attack.

Performance Metric	Logistic Regression	Decision tree
Accuracy	87%	86.81%
Precision	89 %	87%
Sensitivity	88%	87%
F1 score	88 %	87%

Based on what is shown in the table, we can conclude that the logistic regression yielded better results than the decision tree.

Phase 3:

1. Unsupervised Learning

In this section, we use unsupervised learning algorithm which is k-means to group the dataset into clusters. To cluster the dataset using k-means algorithm, we perform the following steps:
Step 1: drop the class label
Step 2: Build the K-means model.
Step 3: Train the K-means model.
Step 4: Evaluate the K-means model.

We evaluate the k-means model for different number of clusters: from 2 clusters to 9 clusters. To evaluate the k-means model, we used two evaluation metrics: total within-cluster sum of square and Silhouette coefficient. The following figure shows python code for clustering using K-means algorithm and evaluating the performance of the algorithm.

Step 1:

While clustering is an unsupervised machine learning task, so we are going to give the algorithm a lot of input data with no class labels, so the first step will be dropping the class label, we also drop the first column, which is the index of the rows (not beneficial data).

```
import pandas as pd
from sklearn import preprocessing as pre
import numpy as np
data = pd.read_csv('heartpreprocessed.csv')
data.drop(['output', data.columns[0]], axis=1)
```

Figure 35: drop the class label

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	57	0	0	140	241	0	1	123	1	0.2	1	0	3
298	45	1	3	110	264	0	1	132	0	1.2	1	0	3
299	68	1	0	144	193	1	1	141	0	3.4	1	2	3
300	57	1	0	130	131	0	1	115	1	1.2	1	1	3
301	57	0	1	130	236	0	0	174	0	0.0	1	1	2

302 rows × 13 columns

Figure 36: drop the class label result

Step 2, 3, 4:

In this step we will build a k-means model which is technique used to identify clusters of data objects in a dataset. [4]

First, we will import the needed library for build, train, evaluate the k-means model.

```
from traitlets.config import List
from sklearn.metrics.cluster import silhouette_score
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from yellowbrick.cluster import SilhouetteVisualizer
```

Figure 37: library used for build, Train, Evaluate the k-means model

Second, we build our model then we train and evaluate the model

```
# Extract features: Feature extraction is one of the crucial
# steps to obtain an efficient representation of input patterns
X = data.iloc[:, :].values
WCSS=[]
for k in range(2,10):
  #create K-means model
  Kmeans=KMeans(n_clusters=k, max_iter=1000)
  #train the model using the dataset
  Kmeans.fit(X)
  #evaluate the model
  lables =Kmeans.predict(X)
  #claculate Silhouette coefficient score for each number of clustrs
  score =silhouette_score(X,lables, metric='euclidean')
  print("K"+str(k)+" Silhouette coefficient score: " +str(score))
  #claculate WCSS for each number of clusters
  WCSS.append(Kmeans.inertia_)
```

Figure 38: build, train and evaluate the model

Third, as we see in above code figure 38, we calculate the Silhouette score for each number of clusters, and as we know the silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate.

The following figure shows the Silhouette score for different number of clusters. The results show that the highest Silhouette score was 0.4328835 for two clusters. This implies that the optimal number of clusters are two clusters

```
K2 Silhouette coefficient score: 0.4328835170484906
K3 Silhouette coefficient score: 0.3072623091388496
K4 Silhouette coefficient score: 0.295036719835046
K5 Silhouette coefficient score: 0.3037831612983324
K6 Silhouette coefficient score: 0.2830645031989993
K7 Silhouette coefficient score: 0.2757697191237553
K8 Silhouette coefficient score: 0.2503332935005701
K9 Silhouette coefficient score: 0.23767388915462362
```

Figure 39: Silhouette score for each cluster

```
K2 WCSS: 1630264.624818801
K3 WCSS: 1288613.1833022707
K4 WCSS: 1068553.1340408612
K5 WCSS: 871773.8790552414
K6 WCSS: 779569.3397454319
K7 WCSS: 711998.9920311256
K8 WCSS: 655760.4541068628
K9 WCSS: 602258.2916314325
```

Figure 40: calculate SSE for each cluster

to justify why we choice [2, 3, 5] size for K, for that we will plot the elbow diagram to find the best value of K, by using number of cluster and WCSS which we have collect it in Figure 38 and displayed in figure 40. [6]

```
# plot the elbow diagram by using number of cluster and WCSS
plt.figure()
plt.plot(range(1, 10), WCSS)
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```

Figure 41: plot the elbow diagram code

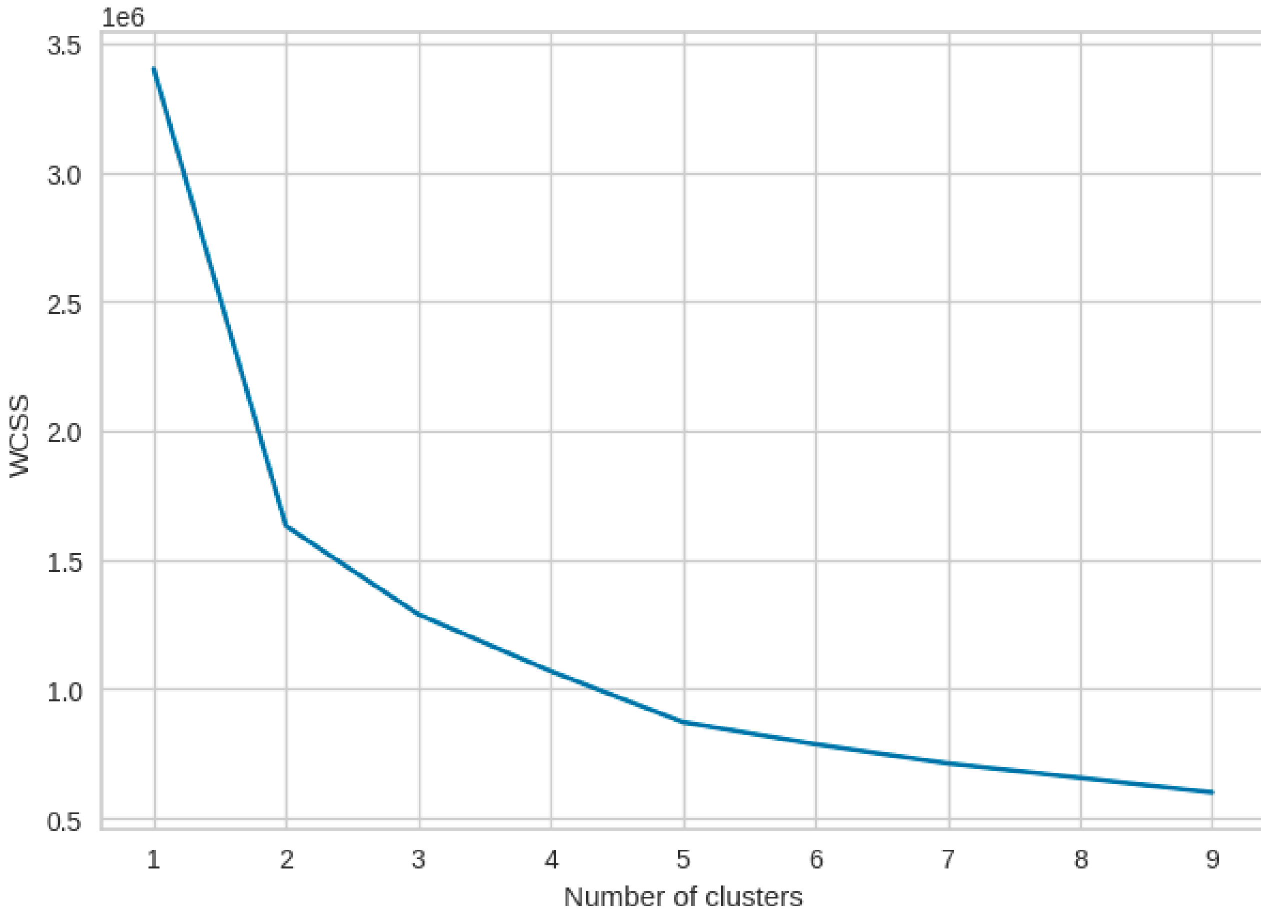


Figure 42: elbow diagram

So, after that we have find the optimal value of clusters which is looks like an Elbow we can find it clearly in cluster 2 and 5 but 3 is difficult to find, and now we will find plot of the optimal values of the Silhouette Visualizer. [5]

```
fig, ax = plt.subplots(2,2, figsize=(6,4))
for i in [2,3,5]:
  #create KMeans instanc for different number of clusters
  km= KMeans(n_clusters=i, init="k-means++", n_init=10, max_iter=1000, random_state=1)
  q, mod= divmod(i, 2)
  #create SilhouetteVisualizer instance with KMeans instane and Fit the visualizer
  visualizer= SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
  visualizer.fit(X)
```

Figure 43: plot the Silhouette Visualizer code

*x-axis: represent the Silhouette score and y-axis: cluster label
The result of following figure is the best result we have implemented.

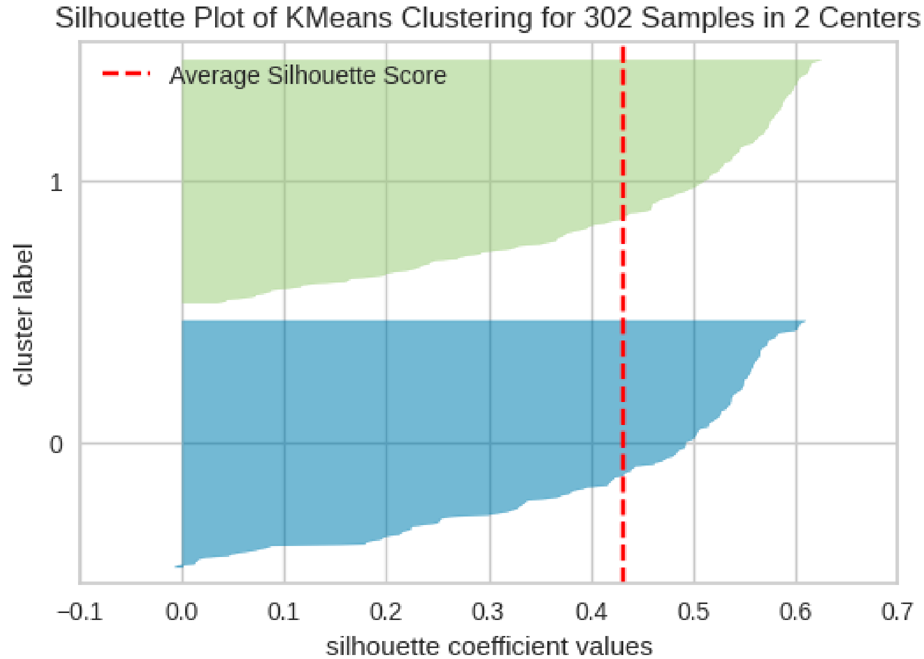


Figure 44: Silhouette Visualizer for two cluster

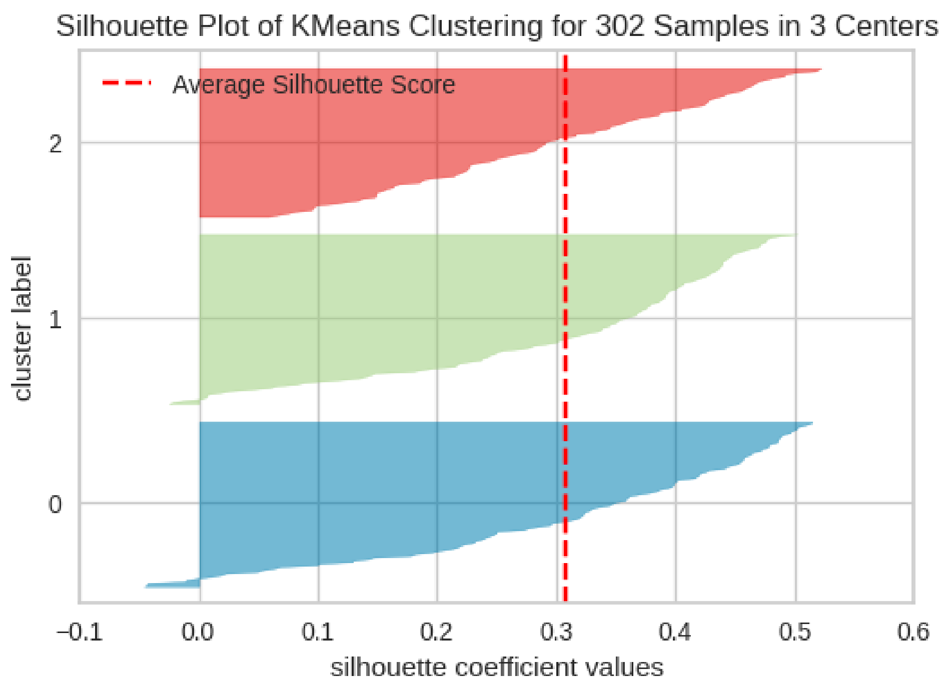


Figure 45: Silhouette Visualizer for three cluster

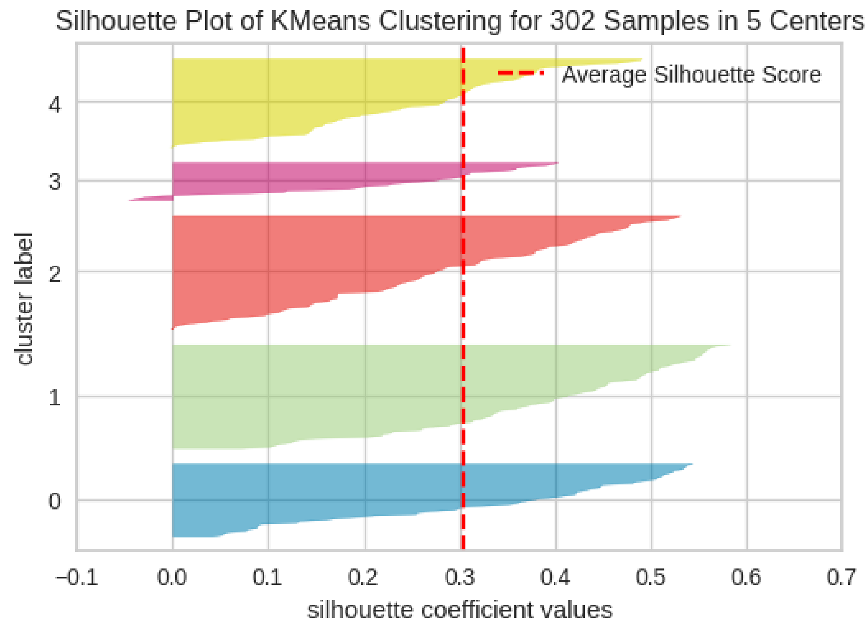


Figure 46: Silhouette Visualizer for five cluster

```
#plot the result of each clustr
plt.scatter(X[:, 0], X[:, 1], c= lables, s=50, cmap='viridis')
plt.scatter(Kmeans.cluster_centers_[:, 0], Kmeans.cluster_centers_[:, 1], c='red', s=200, alpha=0.5)
plt.title('Clustering Results')
plt.show()
```

Figure 47: plot the cluster

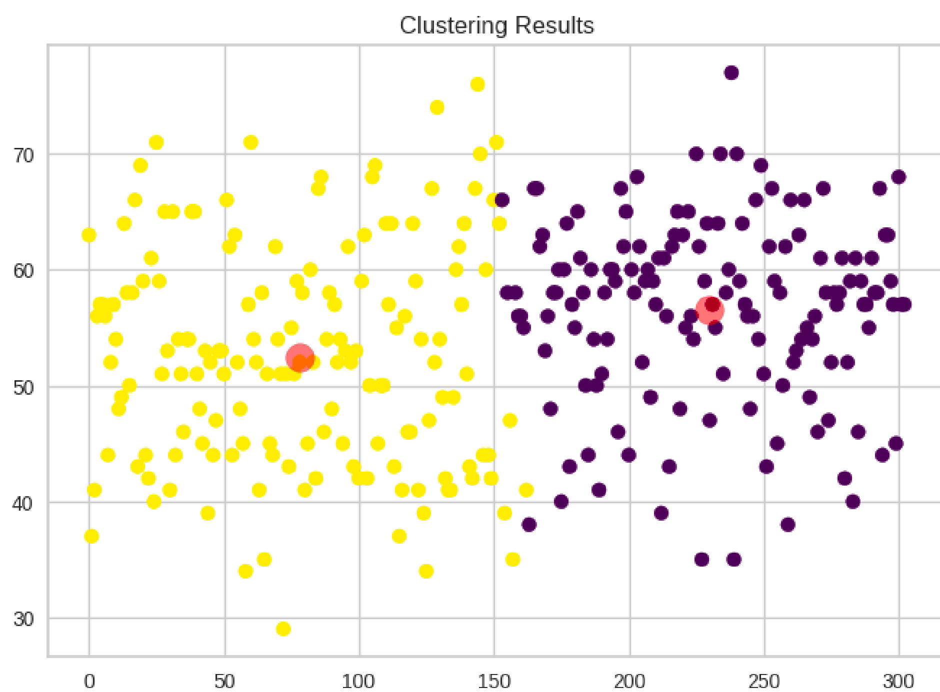


Figure 48: two cluster plot

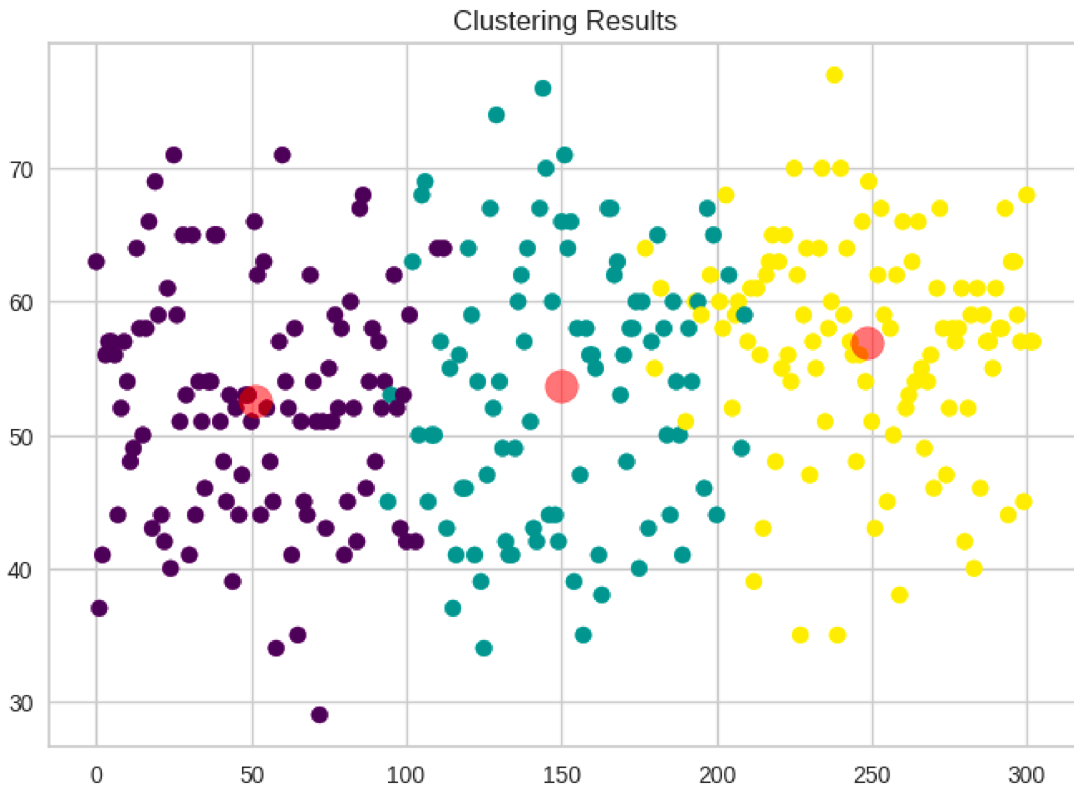


Figure 49: three cluster plot

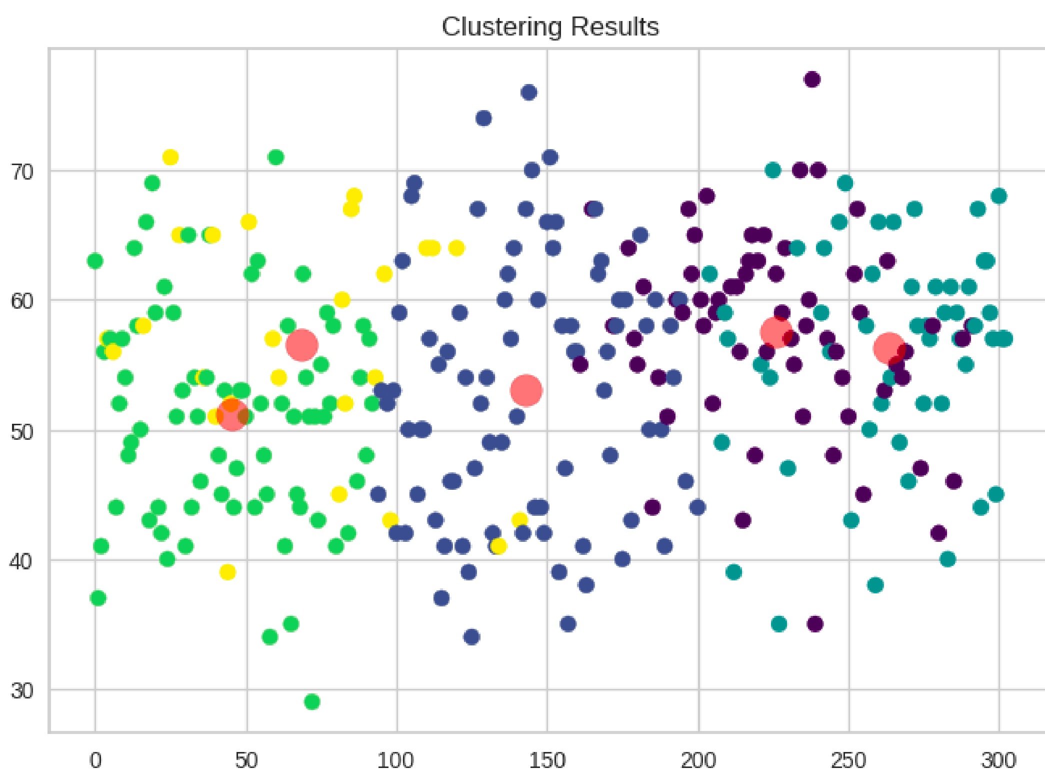


Figure 50: five cluster plot

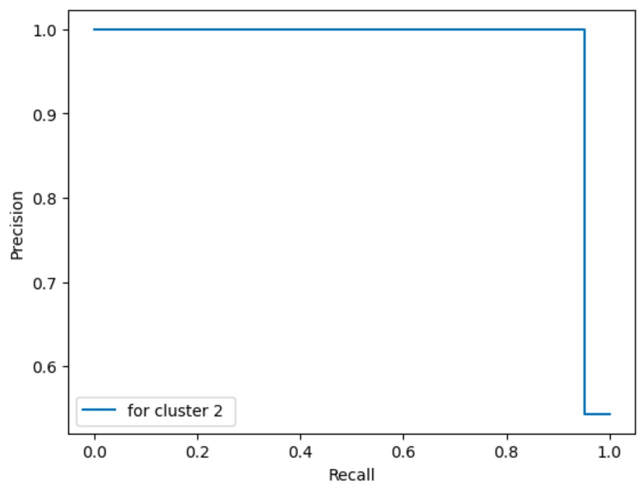
Plot Precision Recall Curve

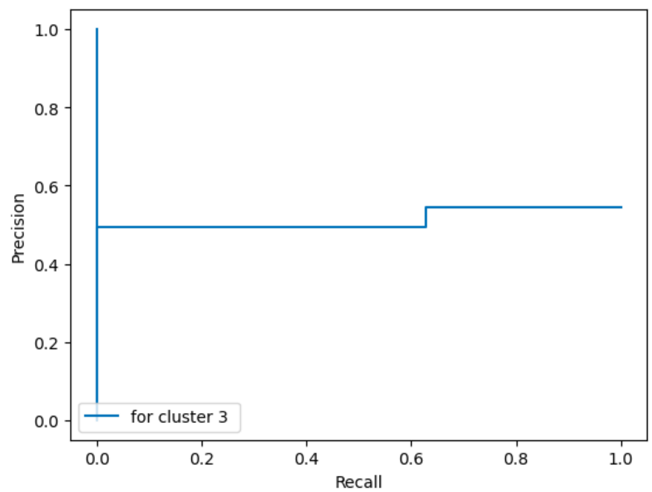
Precision can be seen as a measure of quality and recall as a measure of quantity. Higher precision means that an algorithm returns more relevant results

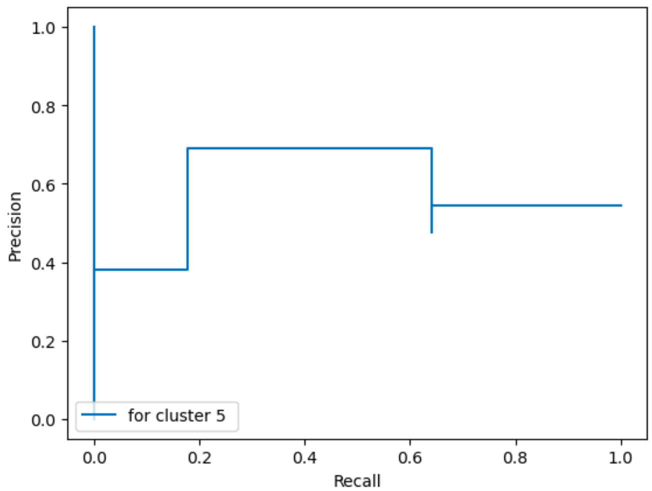
Precision and Recall Formulas

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \qquad \text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

Plot the Precision and Recall







And to see the different between the optimal values of K and bad ones we have taken k=9 as example which is the worst value we have implement it, we can see that the Silhouette score have value of 0.23779 which considered not good score, also in the figure 51 we can clearly see the unacceptable result.

```
K9 Silhouette coefficient score: 0.23779195797564767
```

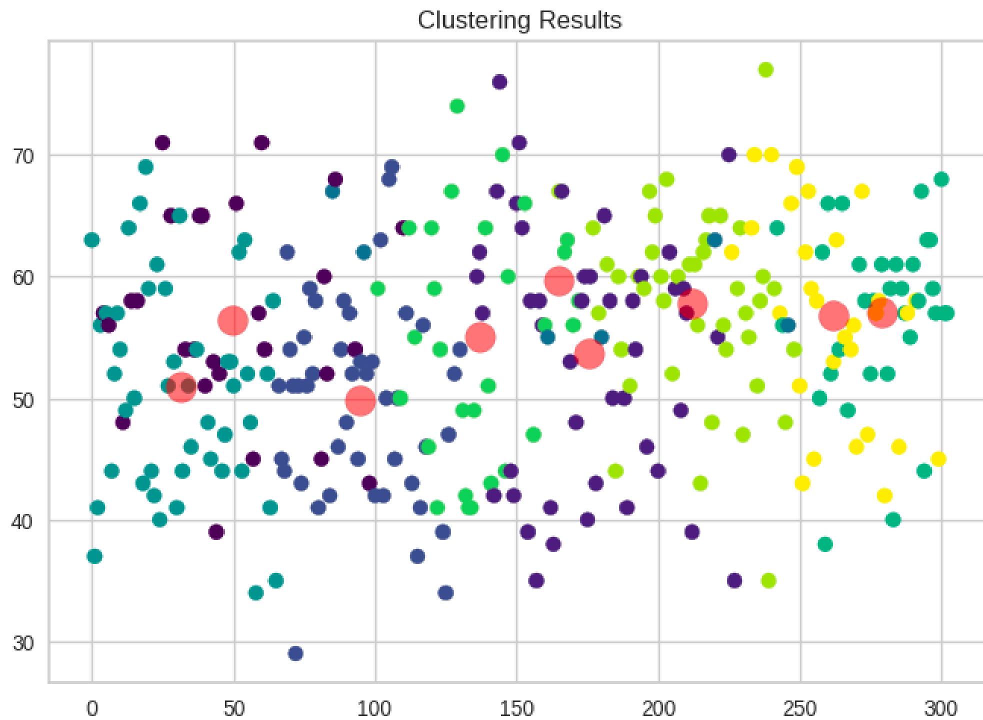


Figure 51: nine cluster plot

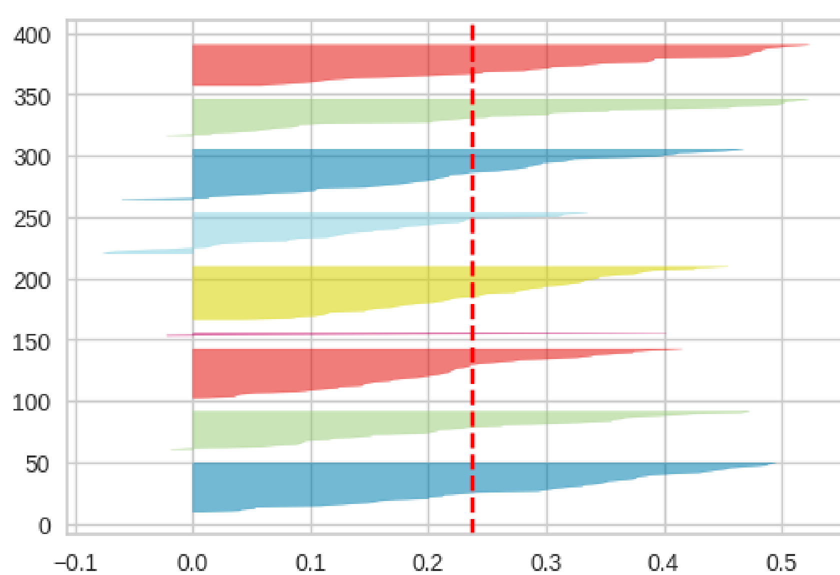


Figure 52: Silhouette Visualizer for nine cluster

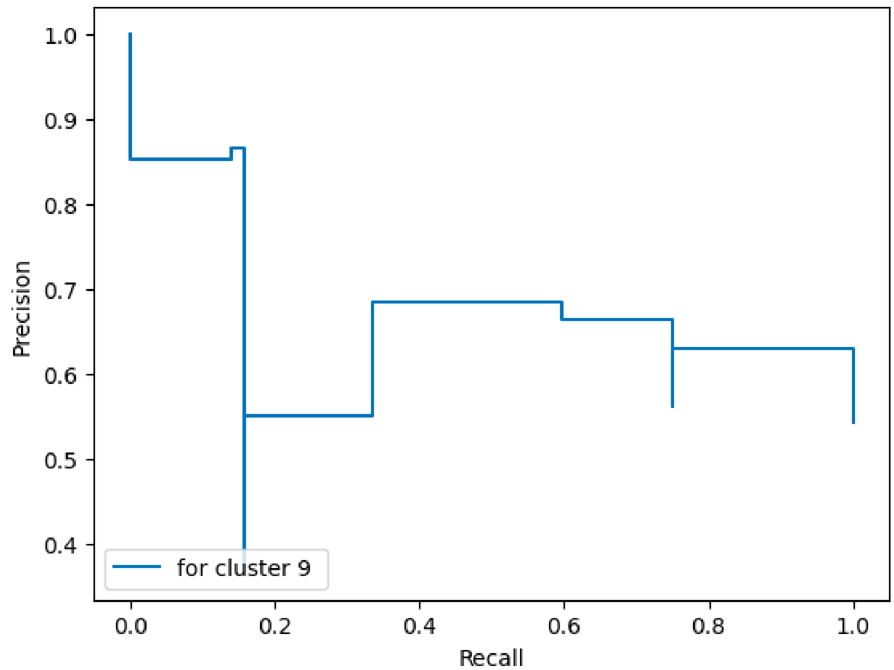
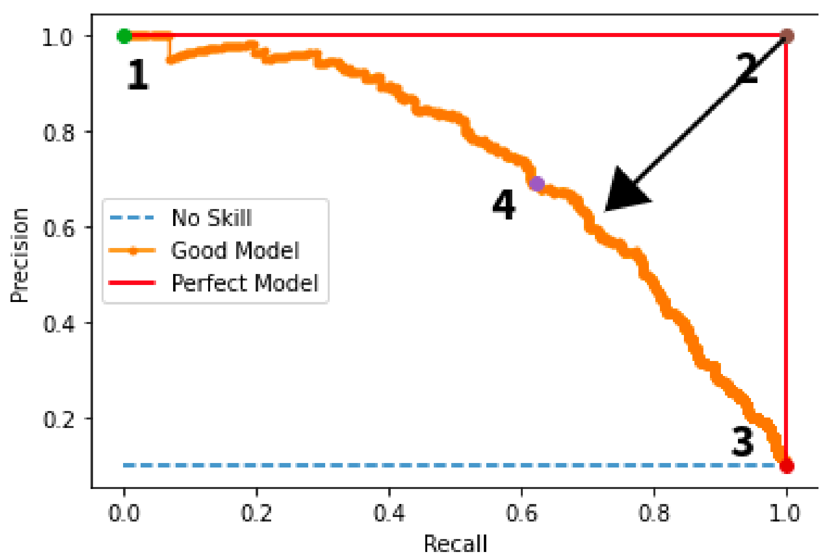


Figure 53: Precision and Recall for nine cluster

So, as we know the perfict fit of Precision and Recall is to fit the border of right and up. Ex:



We see that the k=3,5 have a good fit while the k=2 have properly perfect fit.

Conclusion

In this project, we applied two supervised machine learning classification algorithms, namely Logistic Regression and DT to predicting the chance of a heart attack using measurements of mineral elements. We also used K-means clustering algorithm to group the dataset into different number of clusters. The dataset was cleaned before data mining tasks. Results showed that data mining classification algorithms can successfully predict the chance of a heart attack with a classification accuracy near to 90%. Results of unsupervised learning showed that the optimal number of clusters were two clusters.

References

[1] IBM, "What is a Decision Tree | IBM," *www.ibm.com*. https://www.ibm.com/topics/decision-trees#:~:text=A%20decision%20tree%20is%20a(accessed May 14, 2023).

[2] Google, "Google Colaboratory," *Google.com*, 2019. https://colab.research.google.com/(accessed May 14, 2023).

[3] Pandas, "Python Data Analysis Library — pandas: Python Data Analysis Library," *Pydata.org*, 2018. https://pandas.pydata.org/(accessed May 14, 2023).

[4]K. Arvai, "K-Means Clustering in Python: A Practical Guide – Real Python," realpython.com. https://realpython.com/k-means-clustering-python/(accessed May 20, 2023).

[5]"Silhouette Visualizer — Yellowbrick v1.5 documentation," *www.scikit-yb.org*. https://www.scikit-yb.org/en/latest/api/cluster/silhouette.html(accessed May 21, 2023).

[6]Zach, "How to Use the Elbow Method in Python to Find Optimal Clusters," *Statology*, Jan. 03, 2023. https://www.statology.org/elbow-method-in-python/#:~:text=One%20of%20the%20most%20common%20ways%20to%20choose%20a%20value(accessed May 21, 2023).

[7]C. Rajendran, "Unsupervised Machine Learning (KMeans Clustering) with Scikit-Learn," *Ascentic Technology*, May 10, 2020. https://medium.com/ascentic-technology/unsupervised-machine-learning-kmeans-clustering-with-scikit-learn-bc8895cd66a8 (accessed May 21, 2023).

[8]"sklearn.metrics.PrecisionRecallDisplay," *scikit-learn*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.PrecisionRecallDisplay.html#sklearn.metrics.PrecisionRecallDisplay (accessed May 22, 2023).