

Missing Values

Missing values occurs in dataset when some of the informations is not stored for a variable There are 3 mechanisms

1 Missing Completely at Random, MCAR:

Missing completely at random (MCAR) is a type of missing data mechanism in which the probability of a value being missing is unrelated to both the observed data and the missing data. In other words, if the data is MCAR, the missing values are randomly distributed throughout the dataset, and there is no systematic reason for why they are missing.

For example, in a survey about the prevalence of a certain disease, the missing data might be MCAR if the survey participants with missing values for certain questions were selected randomly and their missing responses are not related to their disease status or any other variables measured in the survey.

2. Missing at Random MAR:

Missing at Random (MAR) is a type of missing data mechanism in which the probability of a value being missing depends only on the observed data, but not on the missing data itself. In other words, if the data is MAR, the missing values are systematically related to the observed data, but not to the missing data. Here are a few examples of missing at random:

Income data: Suppose you are collecting income data from a group of people, but some participants choose not to report their income. If the decision to report or not report income is related to the participant's age or gender, but not to their income level, then the data is missing at random.

Medical data: Suppose you are collecting medical data on patients, including their blood pressure, but some patients do not report their blood pressure. If the patients who do not report their blood pressure are more likely to be younger or have healthier lifestyles, but the missingness is not related to their actual blood pressure values, then the data is missing at random.

3. Missing data not at random (MNAR)

It is a type of missing data mechanism where the probability of missing values depends on the value of the missing data itself. In other words, if the data is MNAR, the missingness is not random and is dependent on unobserved or unmeasured factors that are associated with the missing values.

For example, suppose you are collecting data on the income and job satisfaction of employees in a company. If employees who are less satisfied with their jobs are more likely to refuse to report their income, then the data is not missing at random. In this case, the missingness is dependent on job satisfaction, which is not directly observed or measured.

In [1]:

```
import seaborn as sns
```

In [2]:

```
df=sns.load_dataset('titanic')
```

In [3]:

```
df.head()
```

Out[3]:

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alo
----------	--------	-----	-----	-------	-------	------	----------	-------	-----	------------	------	-------------	-------	-----

0	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [4]:

```
## check missing values in dataset
df.isnull().sum()
```

Out[4]:

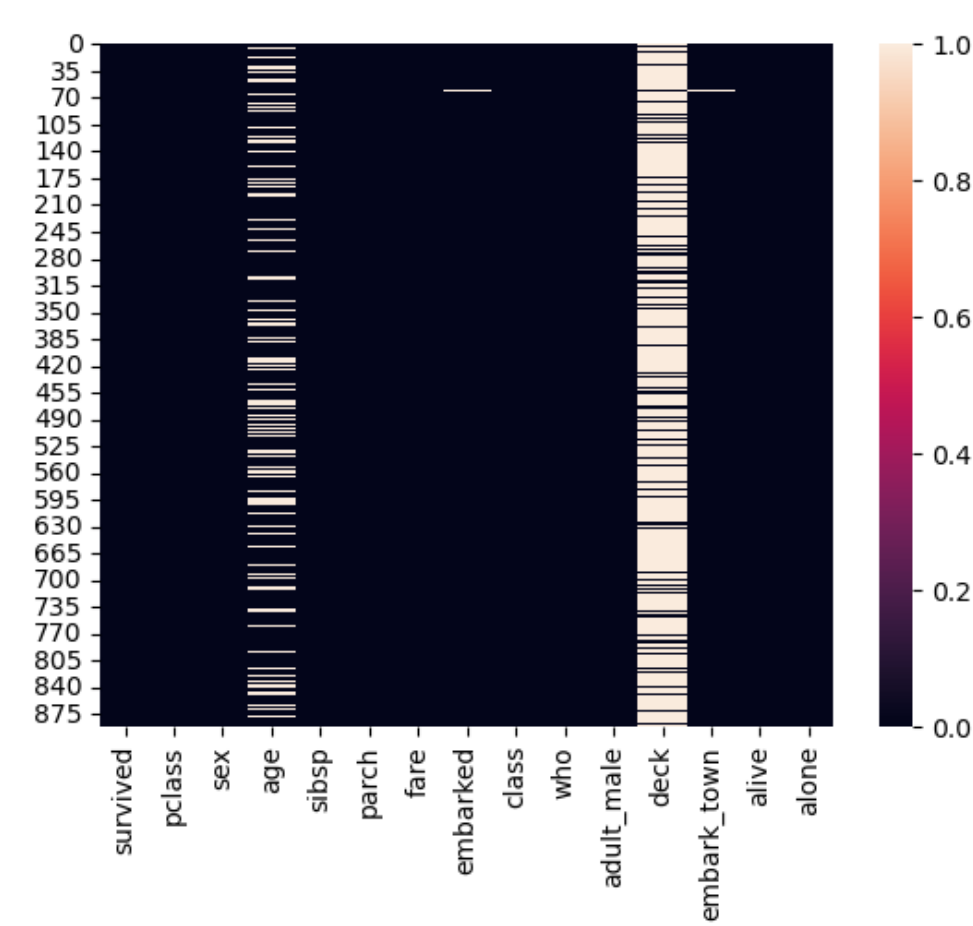
```
survived      0
pclass        0
sex           0
age          177
sibsp         0
parch         0
fare          0
embarked      2
class         0
who           0
adult_male    0
deck         688
embark_town   2
alive         0
alone         0
dtype: int64
```

In [5]:

```
sns.heatmap(df.isnull())
```

Out[5]:

<AxesSubplot:>



In [6]:

```
## HAndling missing by deleting rows
```

In [7]:

```
df.head()
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [8]:

```
## rowwise deletion
df.dropna().shape
```

Out[8]:

(182, 15)

In [9]:

```
df.shape
```

Out[9]:

(891, 15)

In [10]:

```
## Handling missing values by deleting columns
```

In [11]:

```
df.dropna(axis=1)
```

Out[11]:

	survived	pclass	sex	sibsp	parch	fare	class	who	adult_male	alive	alone
0	0	3	male	1	0	7.2500	Third	man	True	no	False
1	1	1	female	1	0	71.2833	First	woman	False	yes	False
2	1	3	female	0	0	7.9250	Third	woman	False	yes	True
3	1	1	female	1	0	53.1000	First	woman	False	yes	False
4	0	3	male	0	0	8.0500	Third	man	True	no	True
...
886	0	2	male	0	0	13.0000	Second	man	True	no	True
887	1	1	female	0	0	30.0000	First	woman	False	yes	True
888	0	3	female	1	2	23.4500	Third	woman	False	no	False
889	1	1	male	0	0	30.0000	First	man	True	yes	True
890	0	3	male	0	0	7.7500	Third	man	True	no	True

891 rows x 11 columns

Imputation Techniques

1-Mean Value Imputation

In [12]:

```
sns.distplot(df['age'])
```

C:\Users\DIPMANI\AppData\Local\Temp\ipykernel_11404\3234920688.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

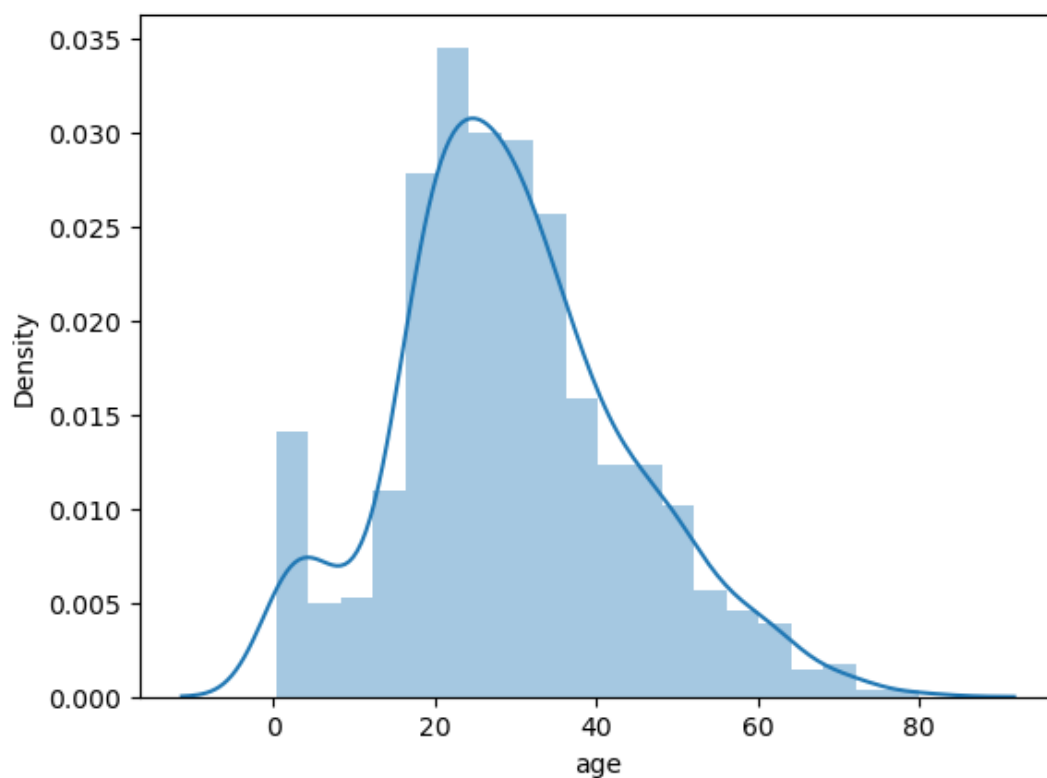
Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(df['age'])
```

Out[12]:

<AxesSubplot:xlabel='age', ylabel='Density'>



In [13]:

```
df.age.isnull().sum()
```

Out[13]:

177

In [14]:

```
df['Age_mean']=df['age'].fillna(df['age'].mean())
```

In [15]:

```
df[['Age_mean', 'age']]
```

Out[15]:

	Age_mean	age
0	22	nan
1	22	nan
2	22	nan
3	22	nan
4	22	nan
5	22	nan
6	22	nan
7	22	nan
8	22	nan
9	22	nan
10	22	nan
11	22	nan
12	22	nan
13	22	nan
14	22	nan
15	22	nan
16	22	nan
17	22	nan
18	22	nan
19	22	nan
20	22	nan
21	22	nan
22	22	nan
23	22	nan
24	22	nan
25	22	nan
26	22	nan
27	22	nan
28	22	nan
29	22	nan
30	22	nan
31	22	nan
32	22	nan
33	22	nan
34	22	nan
35	22	nan
36	22	nan
37	22	nan
38	22	nan
39	22	nan
40	22	nan
41	22	nan
42	22	nan
43	22	nan
44	22	nan
45	22	nan
46	22	nan
47	22	nan
48	22	nan
49	22	nan
50	22	nan
51	22	nan
52	22	nan
53	22	nan
54	22	nan
55	22	nan
56	22	nan
57	22	nan
58	22	nan
59	22	nan
60	22	nan
61	22	nan
62	22	nan
63	22	nan
64	22	nan
65	22	nan
66	22	nan
67	22	nan
68	22	nan
69	22	nan
70	22	nan
71	22	nan
72	22	nan
73	22	nan
74	22	nan
75	22	nan
76	22	nan
77	22	nan
78	22	nan
79	22	nan
80	22	nan
81	22	nan
82	22	nan
83	22	nan
84	22	nan
85	22	nan
86	22	nan
87	22	nan
88	22	nan
89	22	nan
90	22	nan
91	22	nan
92	22	nan
93	22	nan
94	22	nan
95	22	nan
96	22	nan
97	22	nan
98	22	nan
99	22	nan
100	22	nan
101	22	nan
102	22	nan
103	22	nan
104	22	nan
105	22	nan
106	22	nan
107	22	nan
108	22	nan
109	22	nan
110	22	nan
111	22	nan
112	22	nan
113	22	nan
114	22	nan
115	22	nan
116	22	nan
117	22	nan
118	22	nan
119	22	nan
120	22	nan
121	22	nan
122	22	nan
123	22	nan
124	22	nan
125	22	nan
126	22	nan
127	22	nan
128	22	nan
129	22	nan
130	22	nan
131	22	nan
132	22	nan
133	22	nan
134	22	nan
135	22	nan
136	22	nan
137	22	nan
138	22	nan
139	22	nan
140	22	nan
141	22	nan
142	22	nan
143	22	nan
144	22	nan
145	22	nan
146	22	nan
147	22	nan
148	22	nan
149	22	nan
150	22	nan
151	22	nan
152	22	nan
153	22	nan
154	22	nan
155	22	nan
156	22	nan
157	22	nan
158	22	nan
159	22	nan
160	22	nan
161	22	nan
162	22	nan
163	22	nan
164	22	nan
165	22	nan
166	22	nan
167	22	nan
168	22	nan
169	22	nan
170	22	nan
171	22	nan
172	22	nan
173	22	nan
174	22	nan
175	22	nan
176	22	nan
177	22	nan
178	22	nan
179	22	nan
180	22	nan
181	22	nan
182	22	nan
183	22	nan
184	22	nan
185	22	nan
186	22	nan
187	22	nan
188	22	nan
189	22	nan
190	22	nan
191	22	nan
192	22	nan
193	22	nan
194	22	nan
195	22	nan
196	22	nan
197	22	nan
198	22	nan
199	22	nan
200	22	nan
201	22	nan
202	22	nan
203	22	nan
204	22	nan
205	22	nan
206	22	nan
207	22	nan
208	22	nan
209	22	nan
210	22	nan
211	22	nan
212	22	nan
213	22	nan
214	22	nan
215	22	nan
216	22	nan
217	22	nan
218	22	nan
219	22	nan
220	22	nan
221	22	nan
222	22	nan
223	22	nan
224	22	nan
225	22	nan
226	22	nan
227	22	nan
228	22	nan
229	22	nan
230	22	nan
231	22	nan
232	22	nan
233	22	nan
234	22	nan
235	22	nan
236	22	nan
237	22	nan
238	22	nan
239	22	nan
240	22	nan
241	22	nan
242	22	nan
243	22	nan
244	22	nan
245	22	nan
246	22	nan
247	22	nan
248	22	nan
249	22	nan
250	22	nan
251	22	nan
252	22	nan
253	22	nan
254	22	nan
255	22	nan
256	22	nan
257	22	nan
258	22	nan
259	22	nan
260	22	nan
261	22	nan
262	22	nan
263	22	nan
264	22	nan
265	22	nan
266	22	nan
267	22	nan
268	22	nan
269	22	nan
270	22	nan
271	22	nan
272	22	nan
273	22	nan
274	22	nan
275	22	nan
276	22	nan
277	22	nan
278	22	nan
279	22	nan
280	22	nan
281	22	nan
282	22	nan
283	22	nan
284	22	nan
285	22	nan
286	22	nan
287	22	nan
288	22	nan
289	22	nan
290	22	nan
291	22	nan
292	22	nan
293	22	nan
294	22	nan
295	22	nan
296	22	nan
297	22	nan
298	22	nan
299	22	nan
300	22	nan
301	22	nan
302	22	nan
303	22	nan
304	22	nan
305	22	nan
306	22	nan
307	22	nan
308	22	nan
309	22	nan
310	22	nan
311	22	nan
312	22	nan
313	22	nan
314	22	nan
315	22	nan
316	22	nan
317	22	nan
318	22	nan
319	22	nan
320	22	nan
321	22	nan
322	22	nan
323	22	nan
324	22	nan
325	22	nan
326	22	nan
327	22	nan
328	22	nan
329	22	nan
330	22	nan
331	22	nan
332	22	nan
333	22	nan
334	22	nan
335	22	nan
336	22	nan
337	22	nan
338	22	nan
339	22	nan
340	22	nan
341	22	nan
342	22	nan
343	22	nan
344	22	nan
345	22	nan
346	22	nan
347	22	nan
348	22	nan
349	22	nan
350	22	nan
351	22	nan
352	22	nan
353	22	nan
354	22	nan
355	22	nan
356	22	nan
357	22	nan
358	22	nan
359	22	nan
360	22	nan
361	22	nan
362	22	nan
363	22	nan
364	22	nan
365	22	nan
366	22	nan
367	22	nan
368	22	nan
369	22	nan
370	22	nan
371	22	nan
372	22	nan
373	22	nan
374	22	nan
375	22	nan
376	22	nan
377	22	nan
378	22	nan
379	22	nan
380	22	nan
381	22	nan
382	22	nan
383	22	nan
384	22	nan
385	22	nan
386	22	nan
387	22	nan
388	22	nan
389	22	nan
390	22	nan
391	22	nan
392	22	nan
393	22	nan
394	22	nan
395	22	nan
396	22	nan
397	22	nan
398	22	nan
399	22	nan
400	22	nan
401	22	nan
402	22	nan
403	22	nan
404	22	nan
405	22	nan
406	22	nan
407	22	nan
408	22	nan
409	22	nan
410	22	nan
411	22	nan
412	22	nan
413	22	nan
414	22	nan
415	22	nan
416	22	nan
417	22	nan
418	22	nan
419	22	nan
420	22	nan
421	22	nan
422	22	nan
423	22	nan
424	22	nan
425	22	nan
426	22	nan
427	22	nan
428	22	nan
429	22	nan
430	22	nan
431	22	nan
432	22	nan
433	22	nan
434	22	nan
435	22	nan
436	22	nan
437	22	nan
438	22	nan
439	22	nan
440	22	nan
441	22	nan
442	22	nan
443	22	nan
444	22	nan
445	22	nan
446	22	nan
447	22	nan
448	22	nan
449	22	nan
450	22	nan
451	22	nan
452	22	nan
453	22	nan
454		

	Age_mean	age
1	38.000000	38.0
2	26.000000	26.0
3	35.000000	35.0
4	35.000000	35.0
...
886	27.000000	27.0
887	19.000000	19.0
888	29.699118	NaN
889	26.000000	26.0
890	32.000000	32.0

891 rows × 2 columns

In [16]:

```
## This tecnqive work well when your data is normally distributed
```

2- Median Value Imputation

If you have ooutliers in dataset use thi technique

In [17]:

```
df['Age_median']=df['age'].fillna(df['age'].median())
```

In [18]:

```
df[['Age_median', 'Age_mean', 'age']]
```

Out[18]:

	Age_median	Age_mean	age
0	22.0	22.000000	22.0
1	38.0	38.000000	38.0
2	26.0	26.000000	26.0
3	35.0	35.000000	35.0
4	35.0	35.000000	35.0
...
886	27.0	27.000000	27.0
887	19.0	19.000000	19.0
888	28.0	29.699118	NaN
889	26.0	26.000000	26.0
890	32.0	32.000000	32.0

891 rows × 3 columns

3- Mode Value Imputation-- Categorical

In [19]:

```
df[df['embarked'].isnull()]
```

Out[19]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
61	1	1	female	38.0	0	0	80.0	NaN	First	woman	False	B	NaN	yes	True
829	1	1	female	62.0	0	0	80.0	NaN	First	woman	False	B	NaN	yes	True

In [20]:

```
df['embarked'].unique()
```

Out[20]:

```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [21]:

```
df[df['age'].notna()][ 'embarked' ].mode()[0]
```

Out[21]:

```
'S'
```

In [22]:

```
mode=df[df['age'].notna()][ 'embarked' ].mode()[0]
```

In [23]:

```
mode
```

Out[23]:

```
'S'
```

In [24]:

```
df['embarked_mode']=df['embarked'].fillna(mode)
```

In [25]:

```
df[['embarked_mode','embarked']]
```

Out[25]:

	embarked_mode	embarked
0	S	S
1	C	C
2	S	S
3	S	S
4	S	S
...
886	S	S
887	S	S
888	S	S
889	C	C
890	Q	Q

891 rows x 2 columns

In [26]:

```
df['embarked_mode'].isnull().sum()
```

Out[26]:

0

In [27]:

```
df['embarked'].isnull().sum()
```

Out[27]:

2

SMOTE(Synthetic Minority Oversampling Technique)

SMOTE (Synthetic Minority Over-sampling Technique) is a technique used in machine learning to address imbalanced datasets where the minority class has significantly fewer instances than the majority class. SMOTE involves generating synthetic instances of the minority class by interpolating between existing instances.

In [28]:

```
from sklearn.datasets import make_classification
```

In [29]:

```
## X independent feature
## y dependent feature
X,y=make_classification(n_samples=1000,n_features=2,n_redundant=0,n_clusters_per_class=1,
weights=[0.90],random_state=1)
```

In [30]:

```
import pandas as pd
df1=pd.DataFrame(X,columns=['f1','f2'])
df2=pd.DataFrame(y,columns=['target'])
final_df=pd.concat([df1,df2],axis=1)
```

In [31]:

```
final_df.head()
```

Out[31]:

	f1	f2	target
0	1.536830	-1.398694	1
1	1.551108	1.810329	0
2	1.293619	1.010946	0
3	1.119889	1.632518	0
4	1.042356	1.121529	0

In [32]:

```
final_df['target'].value_counts()
```

Out[32]:

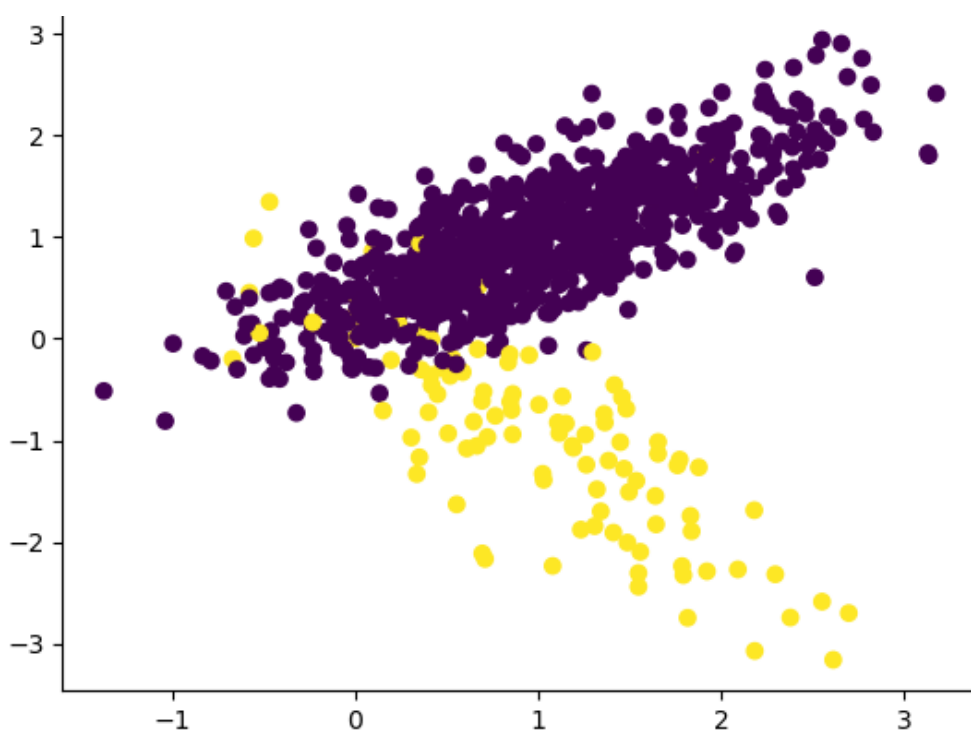
```
0      894
1      106
Name: target, dtype: int64
```

In [33]:

```
import matplotlib.pyplot as plt
plt.scatter(final_df['f1'],final_df['f2'],c=final_df['target'])
```

Out[33]:

```
<matplotlib.collections.PathCollection at 0x295b4003820>
```



In [34]:

```
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
----- 226.0/226.0 kB 3.4 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\dipmani\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\dipmani\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: scipy>=1.3.2 in c:\users\dipmani\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.9.1)
Collecting joblib>=1.1.1
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
----- 298.0/298.0 kB 6.1 MB/s eta 0:00:00
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dipmani\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Installing collected packages: joblib, imbalanced-learn, imblearn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.10.1 imblearn-0.0 joblib-1.2.0
```

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
pandas-profiling 3.2.0 requires joblib~1.1.0, but you have joblib 1.2.0 which is incompatible.

In [35]:

```
from imblearn.over_sampling import SMOTE
```

In [36]:

```
## transform the dataset
oversample=SMOTE()
X,y=oversample.fit_resample(final_df[['f1','f2']],final_df['target'])
```

In [37]:

```
X.shape
```

Out[37]:


```
(1788, 2)
```

```
In [38]:
```

```
len(y[y==0])
```

```
Out[38]:
```

```
894
```

```
In [39]:
```

```
len(y[y==1])
```

```
Out[39]:
```

```
894
```

```
In [40]:
```

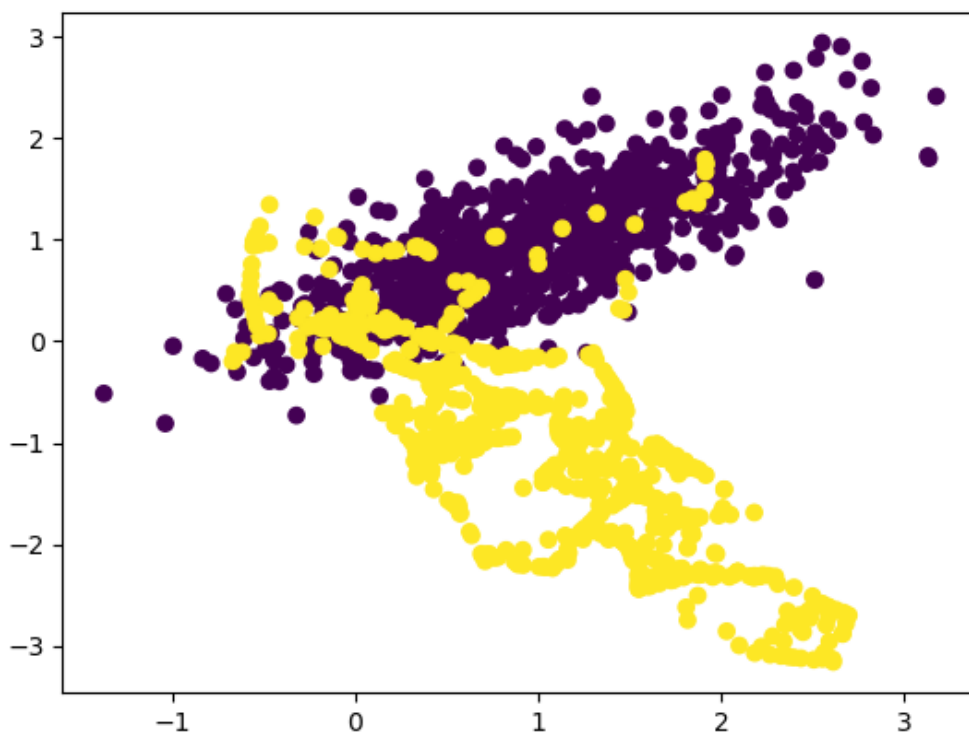
```
df1=pd.DataFrame(X,columns=['f1','f2'])  
df2=pd.DataFrame(y,columns=['target'])  
oversample_df=pd.concat([df1,df2],axis=1)
```

```
In [41]:
```

```
plt.scatter(oversample_df['f1'],oversample_df['f2'],c=oversample_df['target'])
```

```
Out[41]:
```

```
<matplotlib.collections.PathCollection at 0x295b596bf10>
```



3. Data Interpolation

Data interpolation is the process of estimating unknown values within a dataset based on the known values. In Python, there are various libraries available that can be used for data interpolation, such as NumPy, SciPy, and Pandas. Here is an example of how to perform data interpolation using the NumPy library:

1. Linear Interpolation

```
In [42]:
```

```
import numpy as np
```

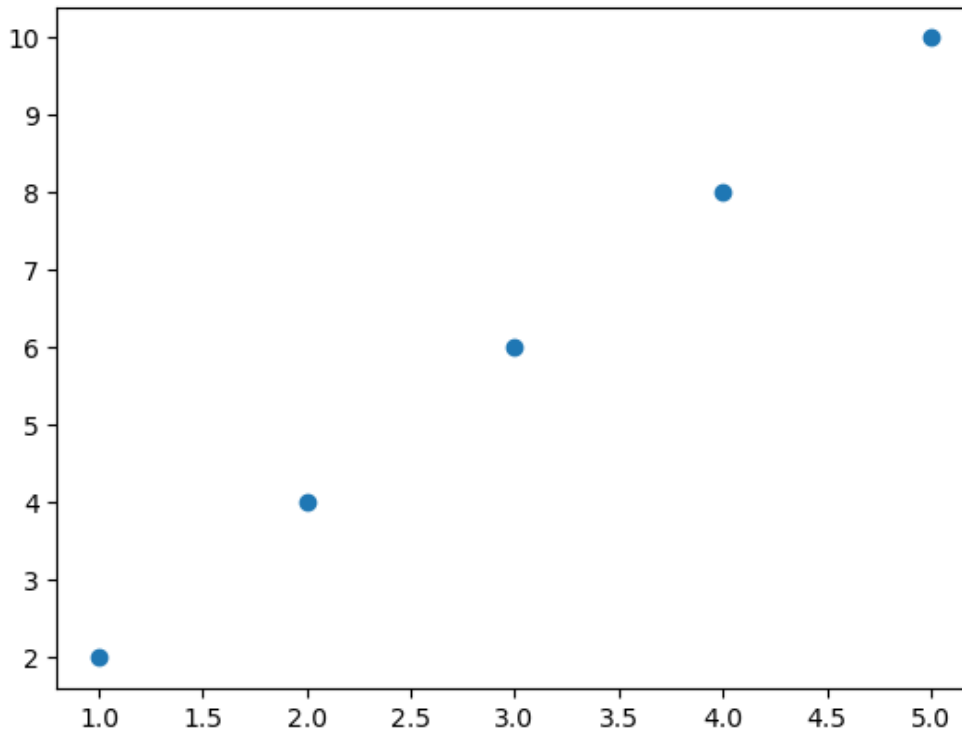
```
x=np.array([1,2,3,4,5])
y=np.array([2,4,6,8,10])
```

In [43]:

```
import matplotlib.pyplot as plt
plt.scatter(x,y)
```

Out[43]:

<matplotlib.collections.PathCollection at 0x295b59e4790>



In [44]:

```
## interpolate the data using linear interpolation
x_new=np.linspace(1,5,10) ##create new x values
y_interp=np.interp(x_new,x,y) ## interpolate y values
print(y_interp)
```

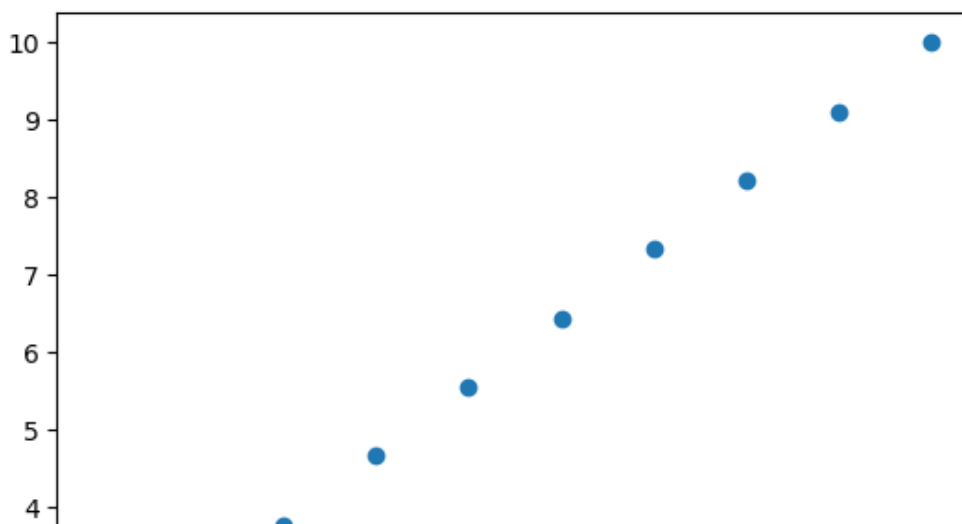
```
[ 2.          2.88888889  3.77777778  4.66666667  5.55555556  6.44444444
  7.33333333  8.22222222  9.11111111 10.          ]
```

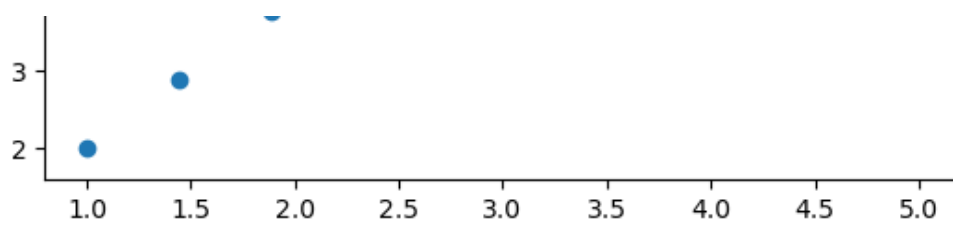
In [45]:

```
plt.scatter(x_new,y_interp)
```

Out[45]:

<matplotlib.collections.PathCollection at 0x295b5a2c190>





Cubic Interpolation With Scipy

In [46]:

```
import numpy as np
x=np.array([1,2,3,4,5])
y=np.array([1,8,27,64,125])
```

In [47]:

```
from scipy.interpolate import interp1d
```

In [48]:

```
##create a cubic interpolation function
f=interp1d(x,y,kind='cubic')
```

In [49]:

```
# interpolate the data
x_new = np.linspace(1, 5, 10)
y_interp=f(x_new)
print(y_interp)
```

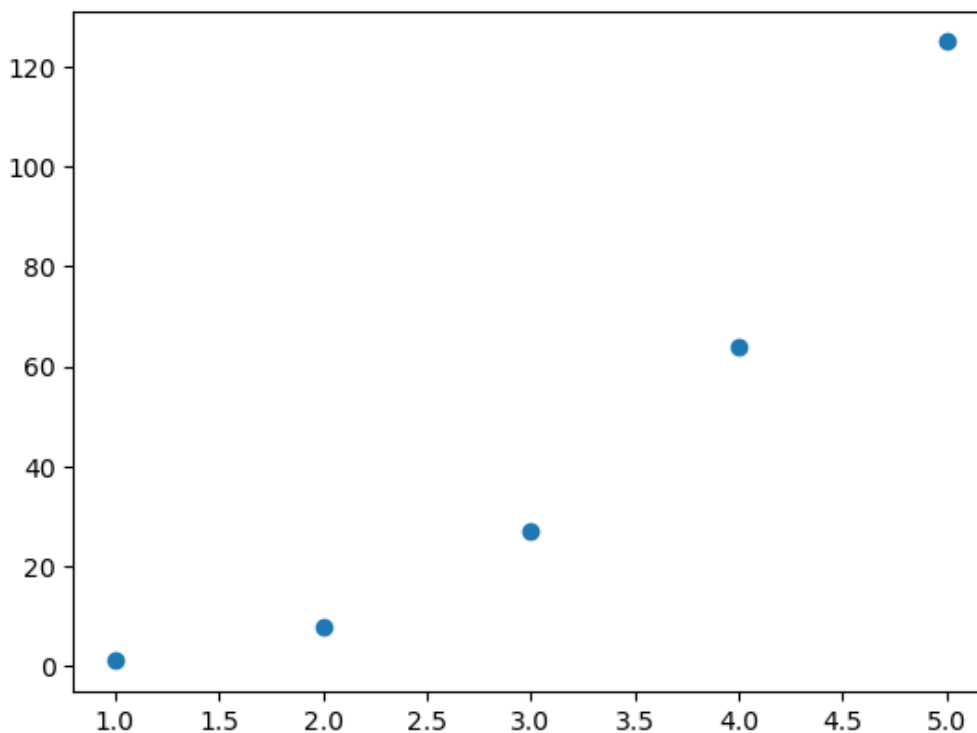
```
[ 1.          3.01371742  6.739369   12.7037037   21.43347051
 33.45541838  49.2962963   69.48285322  94.54183813 125.]
```

In [50]:

```
plt.scatter(x,y)
```

Out[50]:

<matplotlib.collections.PathCollection at 0x295b5a9c430>

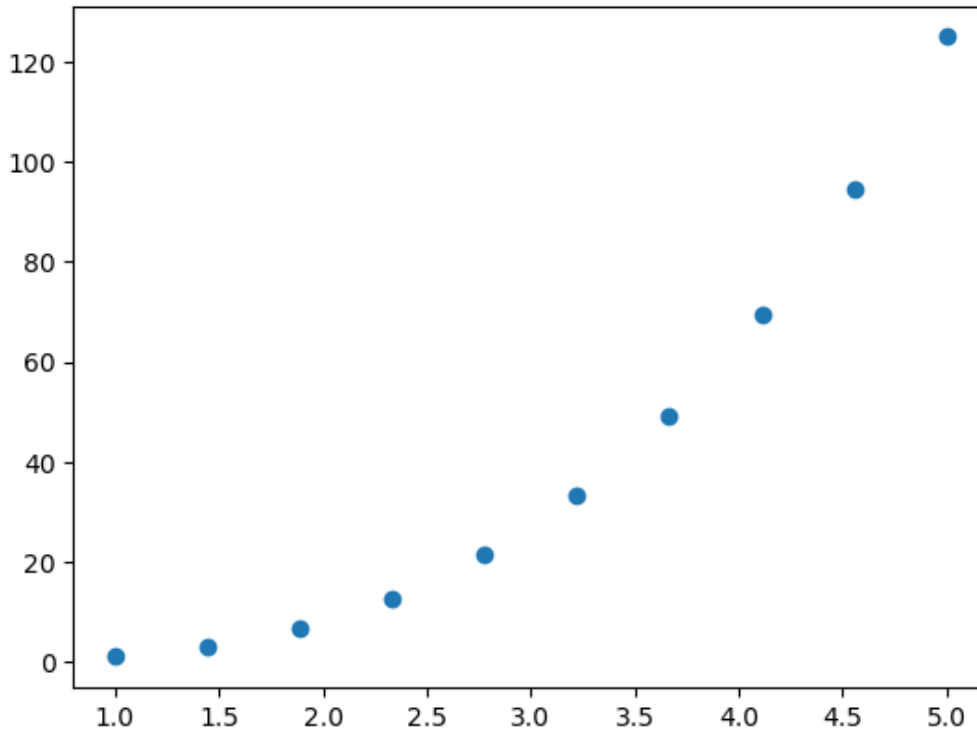


In [51]:

```
plt.scatter(x_new,y_interp)
```

Out[51]:

<matplotlib.collections.PathCollection at 0x295b5b09a00>



Polynomial Interpolation

In [52]:

```
import numpy as np

# create some sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 4, 9, 16, 25])
```

In [53]:

```
# interpolate the data using polynomial interpolation
p = np.polyfit(x, y, 2) # fit a 2nd degree polynomial to the data
```

In [54]:

```
x_new = np.linspace(1, 5, 10) # create new x values
y_interp = np.polyval(p, x_new) # interpolate y values
```

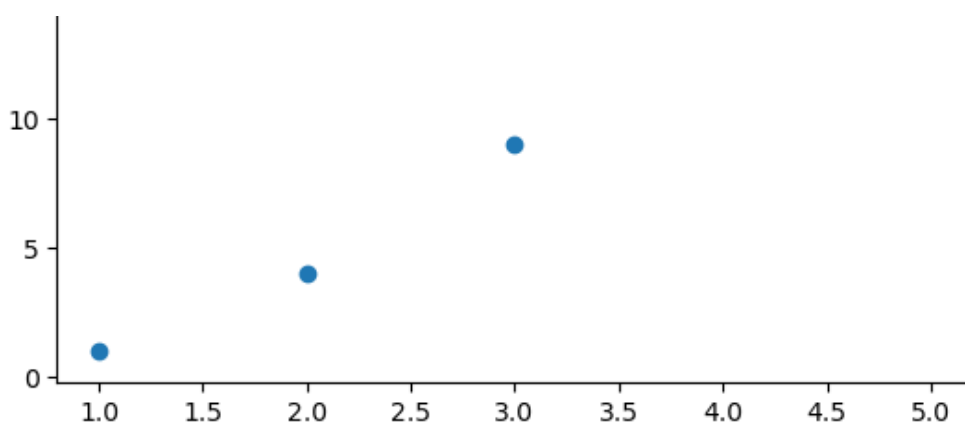
In [55]:

```
plt.scatter(x,y)
```

Out[55]:

<matplotlib.collections.PathCollection at 0x295b5b80250>



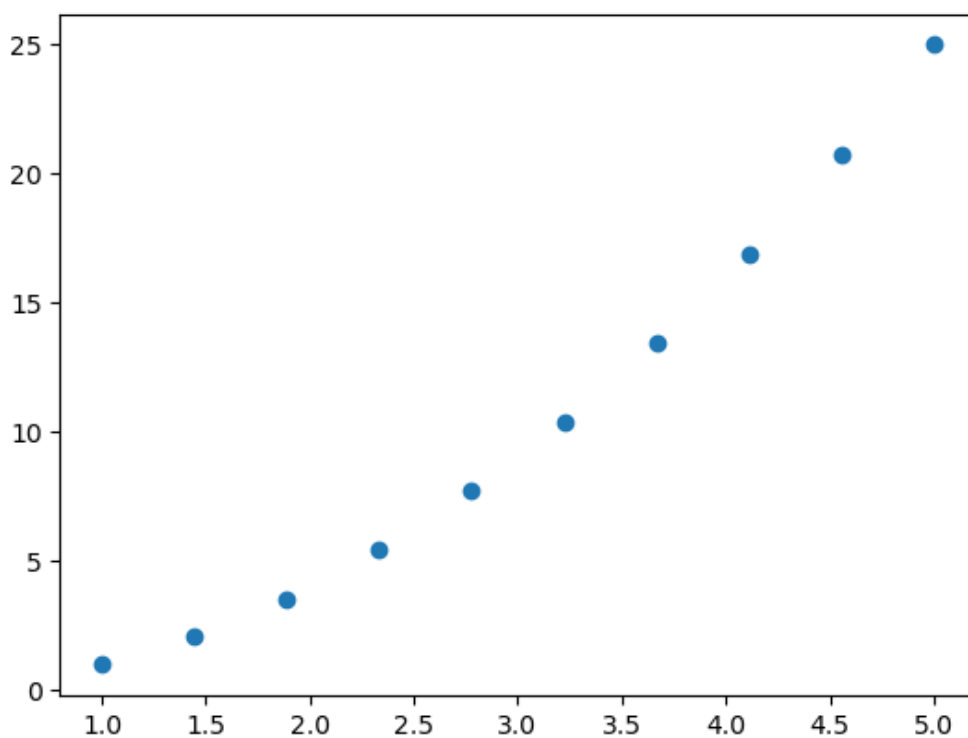


In [56]:

```
plt.scatter(x_new, y_interp)
```

Out[56]:

<matplotlib.collections.PathCollection at 0x295b5be9580>



4. Imbalanced Dataset Handling

1. Upsampling
2. Down Sampling

In [57]:

```
import numpy as np
import pandas as pd
```

In [58]:

```
# Set the random seed for reproducibility
np.random.seed(123)

# Create a dataframe with two classes
n_samples = 1000
class_0_ratio = 0.9
n_class_0 = int(n_samples * class_0_ratio)
n_class_1 = n_samples - n_class_0
```

In [59]:

```
n_class_0,n_class_1
```

Out[59]:

```
(900, 100)
```

In [60]:

```
class_0 = pd.DataFrame({
    'feature_1': np.random.normal(loc=0, scale=1, size=n_class_0),
    'feature_2': np.random.normal(loc=0, scale=1, size=n_class_0),
    'target': [0] * n_class_0
})

class_1 = pd.DataFrame({
    'feature_1': np.random.normal(loc=2, scale=1, size=n_class_1),
    'feature_2': np.random.normal(loc=2, scale=1, size=n_class_1),
    'target': [1] * n_class_1
})
```

In [61]:

```
df=pd.concat([class_0,class_1]).reset_index(drop=True)
```

In [62]:

```
df.head()
```

Out[62]:

	feature_1	feature_2	target
0	-1.085631	0.551302	0
1	0.997345	0.419589	0
2	0.282978	1.815652	0
3	-1.506295	-0.252750	0
4	-0.578600	-0.292004	0

In [63]:

```
df['target'].value_counts()
```

Out[63]:

```
0    900
1    100
Name: target, dtype: int64
```

Upsampling

In [64]:

```
df_minority=df[df['target']==1]
df_majority=df[df['target']==0]
```

In [65]:

```
df_minority.head()
```

Out[65]:

	feature_1	feature_2	target
900	1.699768	2.139033	1
901	1.367739	2.025577	1

902	feature_1	feature_2	target
903	2.213696	3.312255	1
904	3.033878	3.187417	1

In [66]:

```
df_majority.head()
```

Out[66]:

	feature_1	feature_2	target
0	-1.085631	0.551302	0
1	0.997345	0.419589	0
2	0.282978	1.815652	0
3	-1.506295	-0.252750	0
4	-0.578600	-0.292004	0

In [67]:

```
##Upsampling perform
from sklearn.utils import resample
```

In [68]:

```
df_minority_upsample=resample(df_minority,
                               replace=True, ## Sample With replacement
                               n_samples=len(df_majority), # to match the majority class)
                               random_state=42
                               )
```

In [69]:

```
df_minority_upsample.shape
```

Out[69]:

(900, 3)

In [70]:

```
df_minority_upsample.shape
```

Out[70]:

(900, 3)

In [71]:

```
df_minority_upsample['target'].value_counts()
```

Out[71]:

1 900
Name: target, dtype: int64

In [72]:

```
df_upsampled= pd.concat([df_majority,df_minority_upsample])
```

In [73]:

```
df_upsampled['target'].value_counts()
```

Out[73]:

0 900
1 900

Name: target, dtype: int64

In [74]:

```
df_upsampled.shape
```

Out[74]:

```
(1800, 3)
```

DownSampling

In [75]:

```
class_0 = pd.DataFrame({
    'feature_1': np.random.normal(loc=0, scale=1, size=n_class_0),
    'feature_2': np.random.normal(loc=0, scale=1, size=n_class_0),
    'target': [0] * n_class_0
})

class_1 = pd.DataFrame({
    'feature_1': np.random.normal(loc=2, scale=1, size=n_class_1),
    'feature_2': np.random.normal(loc=2, scale=1, size=n_class_1),
    'target': [1] * n_class_1
})
```

In [76]:

```
df=pd.concat([class_0,class_1]).reset_index(drop=True)
```

In [77]:

```
df_minority=df[df['target']==1]
df_majority=df[df['target']==0]
```

In [78]:

```
df_majority_downsample=resample(df_majority,
                                replace=False, ## Sample Without replacement
                                n_samples=len(df_minority), # to match the minority class)
                                random_state=42
                                )
```

In [79]:

```
df_majority_downsample.shape
```

Out[79]:

```
(100, 3)
```

In [80]:

```
df_downsample=pd.concat([df_minority,df_majority_downsample])
```

In [81]:

```
df_downsample['target'].value_counts()
```

Out[81]:

```
1    100
0    100
Name: target, dtype: int64
```

Handling Outliers

1. Minimum Value
2. Q1- 25 percentile
3. Median
4. Q3- 75 percentile
5. MAXimum

In [82]:

```
import numpy as np
1st_marks=[45,32,56,75,89,54,32,89,90,87,67,54,45,98,99,67,74,1000,1100]
```

In [83]:

```
## [Lower Fence<---> Higher Fence]
Q1=np.percentile(lst_marks,[25])
print(Q1)
```

[54.]

In [84]:

```
minimum,Q1,Q2,Q3,maximum=np.quantile(lst_marks,[0,0.25,0.50,0.75,1.0])
```

In [85]:

maximum

Out[85]:

1100.0

In [86]:

```
IQR=Q3-Q1  
print(IQR)
```

35.5

In [87]:

```
lower_fence=Q1-1.5*(IQR)
higher_fence=Q3+1.5*(IQR)
```

In [88]:

lower_fence, higher_fence

Out[88]:

 $(0.75, 142.75)$

In [89]:

```
outliers=[]
for i in lst_marks:
    if i>=0.75 and i<=142.75:
        print("This element is not an outlier")
    else:
        outliers.append(i)
```

[illegible]

This element is not an outlier
This element is not an outlier
This element is not an outlier
This element is not an outlier
This element is not an outlier
This element is not an outlier
This element is not an outlier
This element is not an outlier

In [90]:

```
outliers
```

Out[90]:

```
[1000, 1100]
```

In [91]:

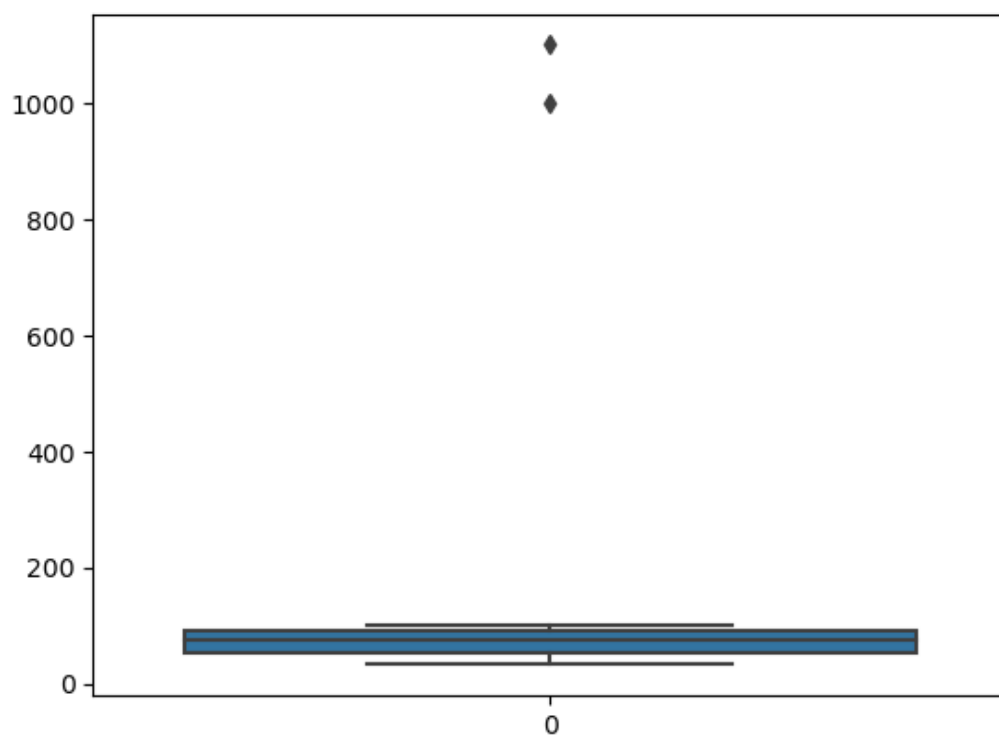
```
import seaborn as sns
```

In [92]:

```
sns.boxplot(lst_marks)
```

Out[92]:

<AxesSubplot:>



In [93]:

```
lst_marks=[45,32,56,75,89,54,32,89,90,87,67,54,45,98,99,67,74]
```

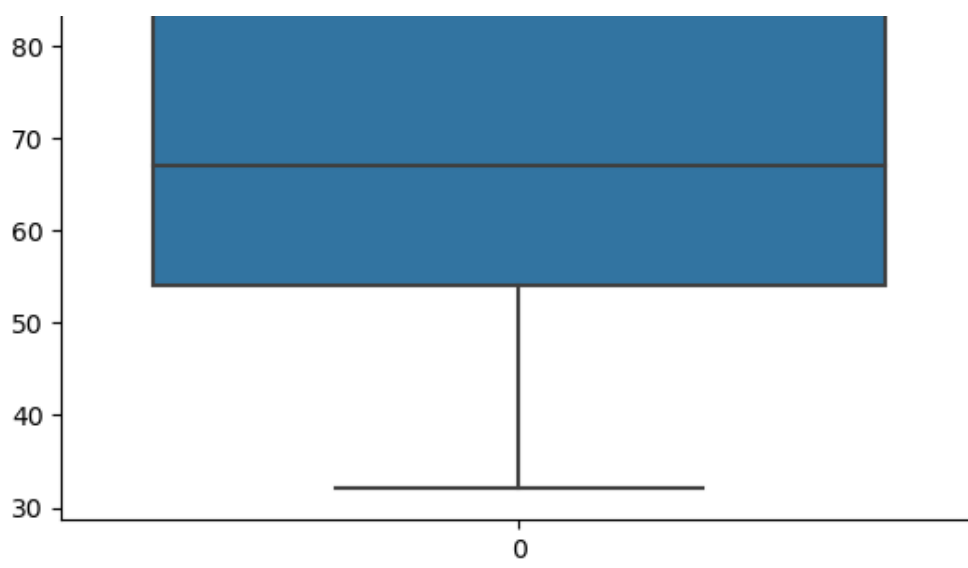
In [94]:

```
sns.boxplot(lst_marks)
```

Out[94]:

<AxesSubplot:>





In []: