Assignment 1:

Question 2 Import the key libraries needed and then read the Haberman csv file.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('~/Desktop/Assignment 1/Haberman.csv')
df.head(10)
```

Out[1]:

	Age	YearofOperation	Number_Auxillary	SurvivalStatus
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1
5	33	58	10	1
6	33	60	0	1
7	34	59	0	2
8	34	66	9	2
9	34	58	30	1

i. Using binning by distance

Grouping the ages into three categories: young, mid-age, and elderly. First pull out the smallest and largest value, find their difference (range) and split it into three equal parts.

In [2]:
```
min_age = df['Age'].min()
max_age = df['Age'].max()
print(min_age)
print(max_age)
```

```
30
83
```

The linspace() function is used to calculate the 4 bins equally, it give three equal splits.

In [3]:
```
bins = np.linspace(min_age,max_age,4)
bins
```

Out[3]:
```
array([30.        , 47.66666667, 65.33333333, 83.        ])
```

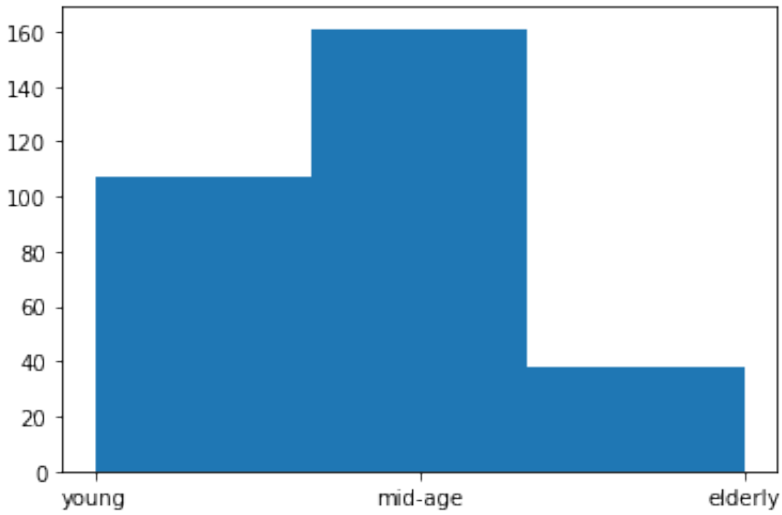
I then define the categorical labels and then convert the numeric values of the column 'Age' into the categorical values using the cut() function.

In [4]:
```
labels = ['young', 'mid-age', 'elderly']

df['cut_bin'] = pd.cut(df['Age'], bins=bins, labels=labels, include_lowest=Tr
```

Plot the distribution

In [5]:
```
plt.hist(df['cut_bin'], bins=3)
plt.show()
```



Majority of the age category are in the mid-age category which is between 48-65, followed by the young category (30-47) and then the elderly 66 and older.

ii. Correlation Analysis

Using Age and Number_Auxilliary for our correlation analysis, we shall find the covariance of each column, calculate standard deviations and then Pearson correlation

In [6]:
```
from numpy import cov

covariance = cov(df['Age'], df['Number_Auxillary'])
print(covariance)
```

```
[[116.71458266  -4.9070824 ]
 [ -4.9070824   51.69111754]]
```

From the covariance matrix above, the covariance is -4.91 which is negative suggesting age and number_auxillary change in different direction

Pearson's Correlation

In [7]:
```
from scipy.stats import pearsonr

corr, _ = pearsonr(df['Age'], df['Number_Auxillary'])
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.063
```

The coefficient shows a very weak negative correlation. I will redo another correlation but this time using age and survival status

In [8]:
```
corr, _ = pearsonr(df['Age'], df['SurvivalStatus'])
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.068
```

Age and Survival rate tends to have a weak but positive relationship.

Correlation Matrix and Correlation Heatmap

The correlation matrix give a pairwise correlation coefficients between two or more (numeric) variables. The heatmap replaces numbers with colors of varying shades, with lighter cells having higher values of r (correlation coefficient).

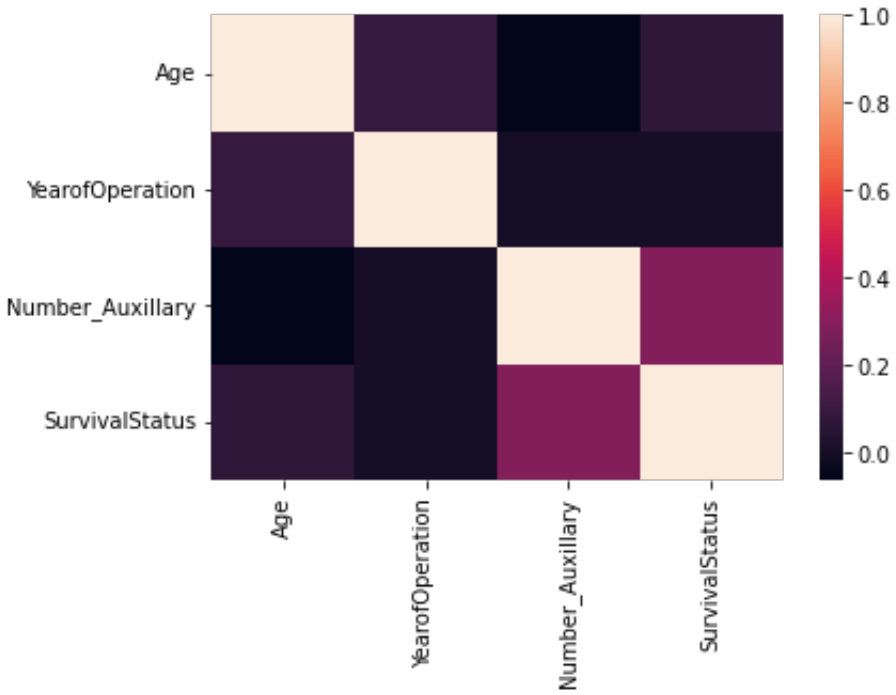
In [9]:
```
cor_mat = df.corr()
round(cor_mat,2)
```

Out[9]:

	Age	YearofOperation	Number_Auxillary	SurvivalStatus
Age	1.00	0.09	-0.06	0.07
YearofOperation	0.09	1.00	-0.00	-0.00
Number_Auxillary	-0.06	-0.00	1.00	0.29
SurvivalStatus	0.07	-0.00	0.29	1.00

In [10]:
```
sns.heatmap(cor_mat)
```

Out[10]:
```
<AxesSubplot:>
```



iii. Dimensionality Reduction

We will do PCA, Principal component analysis importing it from Scikit Learn. Before applying PCA, we will use StandardScaler to help standardize the dataset's features onto a normal scale

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
features = ['Age', 'YearofOperation', 'Number_Auxillary']
# Separating out the features
x = df.loc[:, features].values

# Separating out the target
y = df.loc[:,['SurvivalStatus']].values

# Standardizing the features
x = StandardScaler().fit_transform(x)
```

The feature columns (which are 3: 'Age', 'YearofOperation', 'Number_Auxillary') are projected into 2 dimensions.

After the dimensionality reduction, there isn't a particular meaning assigned to each principal component,they are just the two main dimensions of variation.

In [13]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf
```

Out[13]:

	principal component 1	principal component 2
0	1.079952	-0.164819
1	1.552517	-0.292351
2	0.843669	-0.101052
3	1.963759	-0.937989
4	1.011230	0.355427
...	...	...
301	-1.489786	-0.471916
302	-2.503637	0.303394
303	-2.037949	0.290783
304	-2.219263	0.064149
305	-1.239273	-1.059980

306 rows × 2 columns

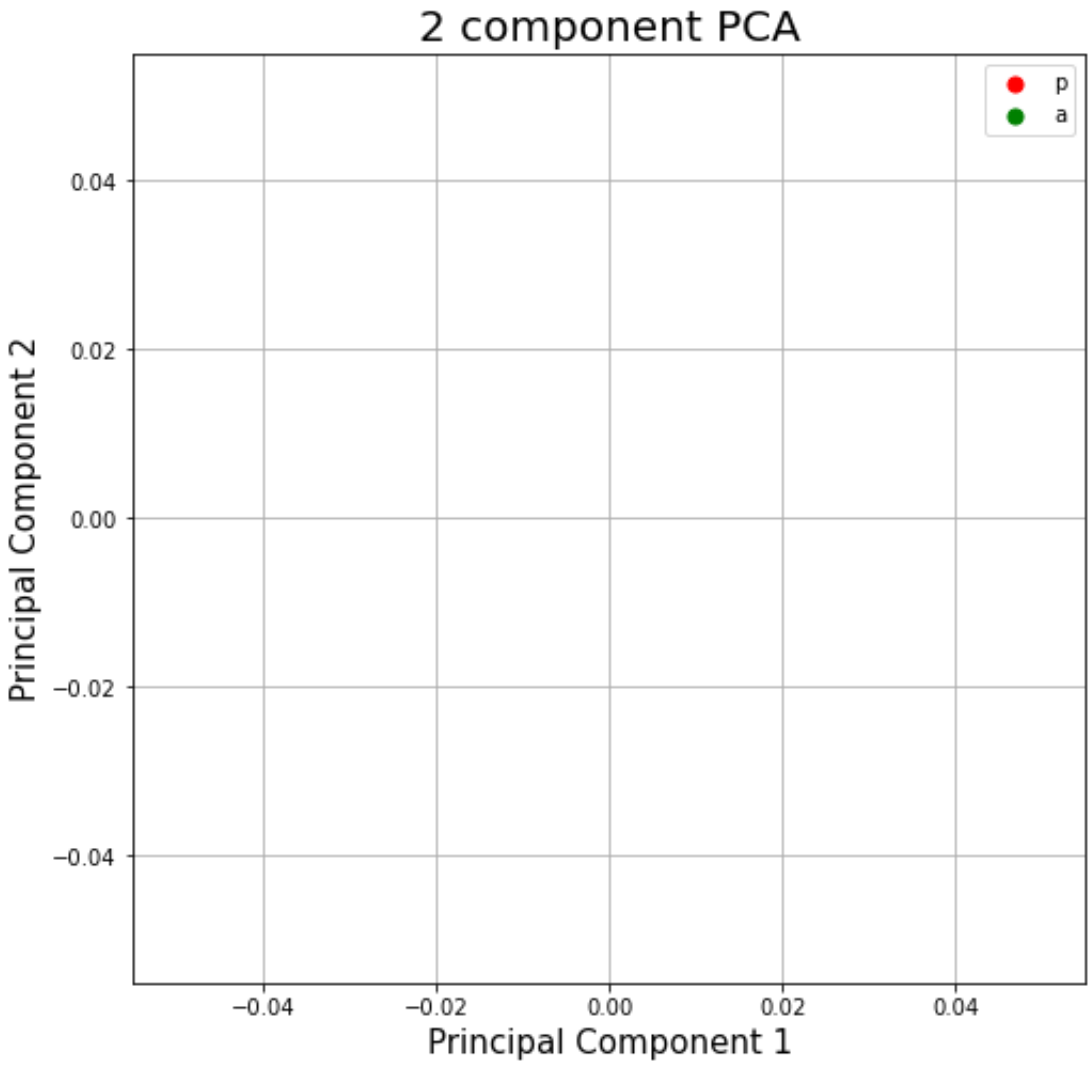
In [14]:
```
finalDf = pd.concat([principalDf, df[['SurvivalStatus']]], axis = 1)
finalDf
```

Out[14]:

	principal component 1	principal component 2	SurvivalStatus
0	1.079952	-0.164819	1
1	1.552517	-0.292351	1
2	0.843669	-0.101052	1
3	1.963759	-0.937989	1
4	1.011230	0.355427	1
...	...	...	...
301	-1.489786	-0.471916	1
302	-2.503637	0.303394	1
303	-2.037949	0.290783	1
304	-2.219263	0.064149	2
305	-1.239273	-1.059980	2

306 rows × 3 columns

In [15]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
SurvivalStatus = ['patient-survived-5years-or-longer', 'patient-died-within-5
colors = ['r', 'g']
for SurvivalStatus, color in zip(SurvivalStatus,colors):
    indicesToKeep = finalDf['SurvivalStatus'] == SurvivalStatus
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(SurvivalStatus)
ax.grid()
```



Not sure why the code above isn't giving any output but I experimented with a better tool below and it worked. Will touch more on the variance explained when we get the output.

In [16]:
```
pca.explained_variance_ratio_
```

Out[16]:
```
array([0.37045668, 0.33215125])
```

Visualizing PCA using plotly

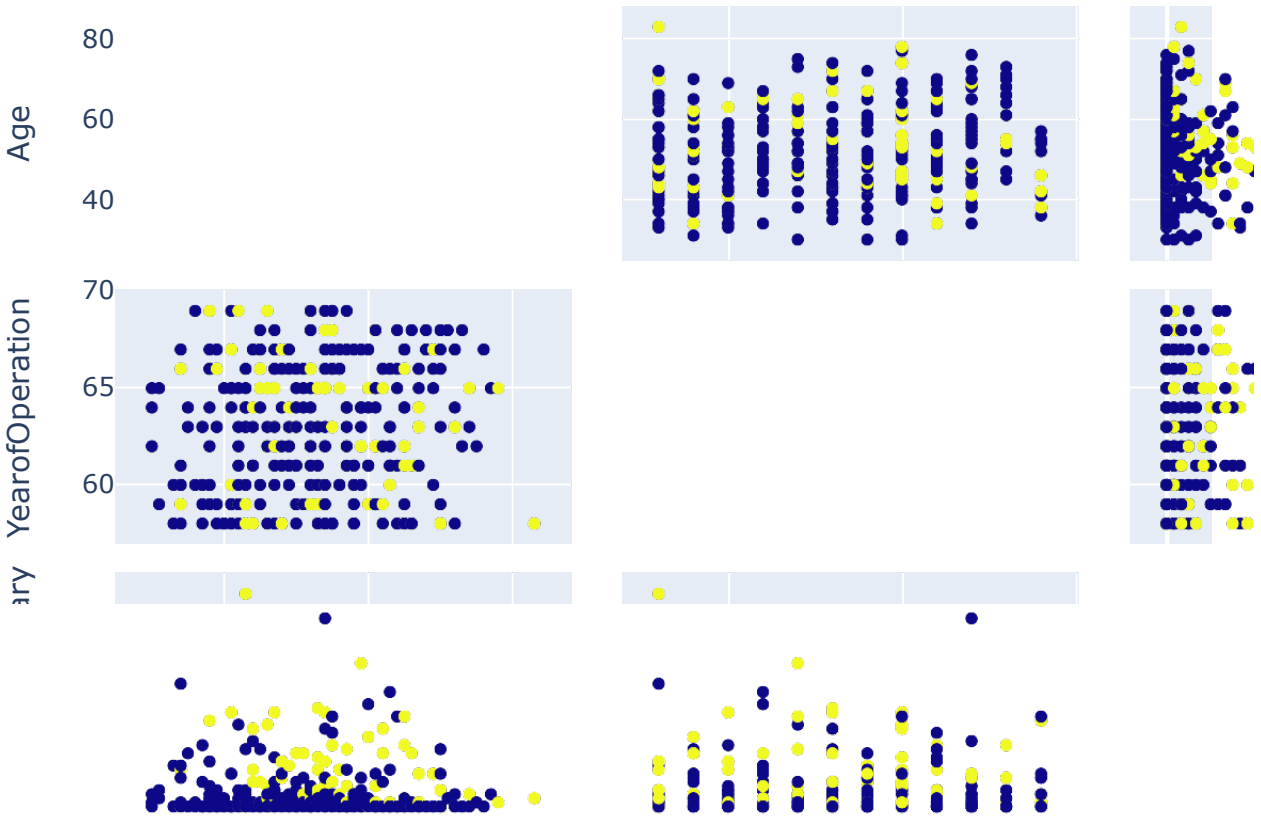
I came across this library online and so I tried experimenting with it. The code below graphs all the original dimensions

In [17]:

```
import plotly.express as px

df = pd.read_csv('~/Desktop/Assignment 1/Haberman.csv')
features = ['Age', 'YearofOperation', 'Number_Auxillary']

fig = px.scatter_matrix(
    df,
    dimensions=features,
    color="SurvivalStatus"
)
fig.update_traces(diagonal_visible=False)
fig.show()
```

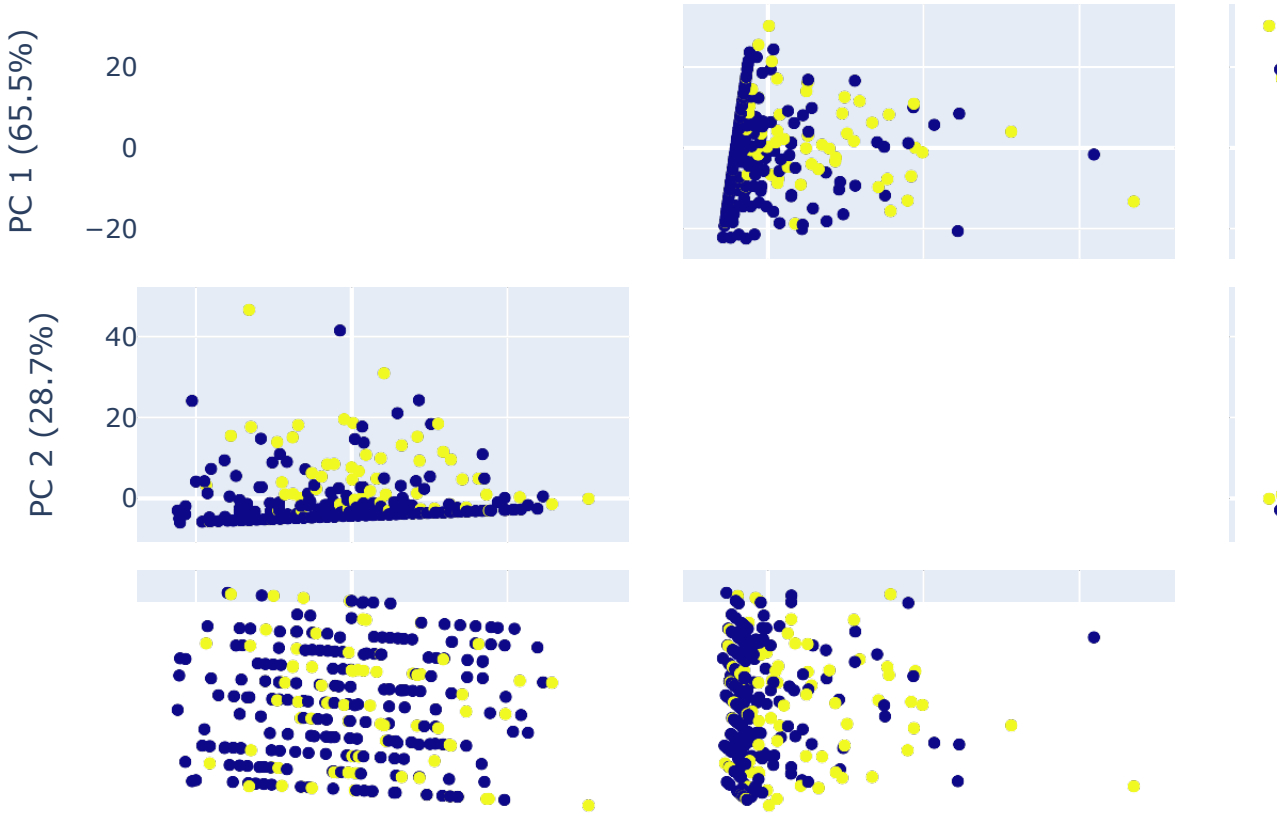


PCA is applied again to retrieve all components and then the same px.scatter_matrix is used to trace and display the results, but this time our features are the resulting principal components, ordered by how much variance they are able to explain

In [18]:

```
pca = PCA()
components = pca.fit_transform(df[features])
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(3),
    color=df["SurvivalStatus"]
)
fig.update_traces(diagonal_visible=False)
fig.show()
```



The code below help visualize the first two principal components of a PCA, by reducing a dataset of 3 dimensions to 2 dimensions.

In [20]:

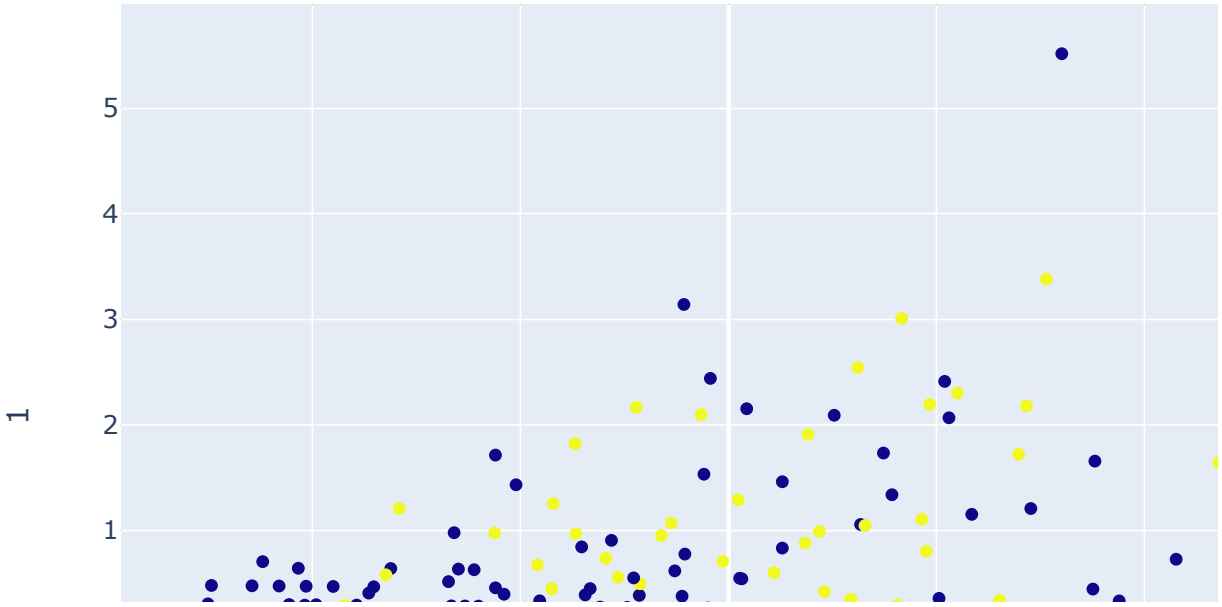
```
features = ['Age', 'YearofOperation', 'Number_Auxillary']
# Separating out the features
x = df.loc[:, features].values

# Separating out the target
y = df.loc[:,['SurvivalStatus']].values

# Standardizing the features
X = StandardScaler().fit_transform(x)


pca = PCA(n_components=2)
components = pca.fit_transform(X)

fig = px.scatter(components, x=0, y=1, color=df['SurvivalStatus'])
fig.show()
```



iv Normalization.

Using z-score normalization on 'YearofOperation' column, which is calculated by finding the difference between the current value and the average value, divided by the standard deviation.

In [21]:
```
df['YearofOperation'] = (df['YearofOperation'] - df['YearofOperation'].mean()
```

In [22]:
```
df['YearofOperation'].min()
```

Out[22]:
```
-1.493486247310241
```

In [23]:
```
df['YearofOperation'].max()
```

Out[23]:
```
1.8917492465929728
```

With the z-score normalization, the new range (min & max) of the column 'YearofOperation' is (-1.49, 1.89)

v. Sampling

In [24]:
```
df.shape
```

Out[24]:
```
(306, 4)
```

Simple Random Sampling: Knowning the exact number of samples to return

In [25]:
```
subset = df.sample(n=250)
subset.shape
```

Out[25]:
```
(250, 4)
```

Manually entered the percentage of samples to return, 65% of the total.

In [26]:
```
subset = df.sample(frac=0.65)
subset.shape
```

Out[26]:
```
(199, 4)
```

Sampling with condition

To return random sample where Age is less than 55 years.

In [27]:
```
condition = df['Age'] < 55
condition
```

Out[27]:
```
0       True
1       True
2       True
3       True
4       True
       ...
301    False
302    False
303    False
304    False
305    False
Name: Age, Length: 306, dtype: bool
```

In [28]:
```
true_index = condition[condition == True].index
len(true_index)
```

Out[28]:
```
180
```

Thus, there are 180 elements (or individuals) that satisfy the condition above (with Age < 55).

In [ ]:
```

```