Ram Lal Anand College

(University of Delhi)

Department of Computer Science

Session :- January to May 2021

Practical File

Submitted to Ms. Dikscha Sapra

Program Name: B.Sc(H) Computer Science

Semester: 6

Title of the paper: Data Mining

Name of the Student: Ajit Singh

Examination Roll no: 19058570034

Practical 1

> Name - Ajit singh
> Roll No. - 19058570034

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
```

In [7]:
```
data = pd.read_csv("p1.csv")
data.head()
```

Out[7]:

	Age	agegroup	height	status	yearsmarried
0	21	adult	6.0	single	-1
1	2	child	3.0	married	0
2	18	adult	5.7	married	20
3	221	elderly	5.0	widowed	2
4	34	child	-7.0	married	3

Creatinf custom function to check ruleset

In [16]:
```
def x(r):
    return x>0 and x<151
```

In [17]:
```
def check_age(data):
    print('Checking age between 0 and 150')
    x = lambda r: r in range(0,151)
    a = np.array(data['Age'].apply(x))
    a = sum(a)
    return a
```

In [18]:
```
check_age(data)
```

Out[18]:
```
Checking age between 0 and 150
4
```

In [19]:
```
def age_years_married(data):
    print("Checking if age>years married")
    x = lambda r: r[0]>r[1]
    a = np.array(data[['Age', 'yearsmarried']].apply(x, axis =1))
    a = sum(a)
    return a
```

In [20]:
```
sum([True,False, True, True])
```

Out[20]: 3

In [21]:
```
age_years_married(data)
```

```
Checking if age>years married
```

Out[21]: 4

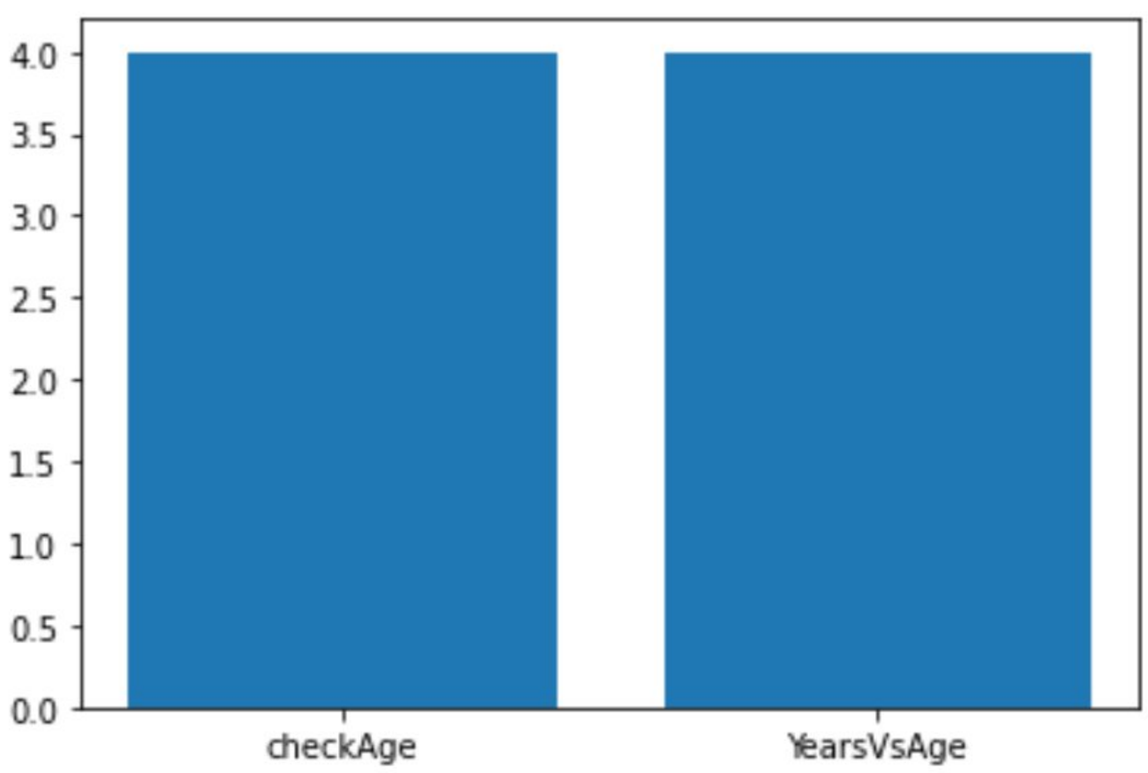
Visualisation

In [22]:
```
import matplotlib.pyplot as plt
```

In [23]:
```
E={'checkAge':check_age,'YearsVsAge':age_years_married}
x=[]
for i in E.keys():
    x.append(E[i](data))
```

```
Checking age between 0 and 150
Checking if age>years married
```

In [25]:
```
plt.bar(E.keys(),x)
plt.show()
```



Practical 2

> Name - Ajit Singh
> Roll No. - 19058570034

Importing Libararies

In [112…

```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_theme(style="whitegrid")
```

Loading data

In [80]:

```
data = pd.read_csv('dirty_iris.csv')
print("Data loaded with {} rows".format(len(data)))
data
```

```
Data loaded with 150 rows
```

Out[80]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	NaN	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor
...	...	...	...	...	...
145	6.7	3.1	5.6	2.4	virginica
146	5.6	3.0	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa
148	6.4	3.1	NaN	1.8	virginica
149	5.8	2.6	4.0	NaN	versicolor

150 rows × 5 columns

Data Cleaning

In [81]:

```
# Data Info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
```

```
 0   Sepal.Length  140 non-null    float64
 1   Sepal.Width   133 non-null    float64
 2   Petal.Length  131 non-null    float64
 3   Petal.Width   137 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

There are many null null values in our dataset

In [82]:
```
# Checking null values in each columns
for i in data.columns:
    sum_na=data[i].isna().sum()
    percent_na=(sum_na/len(data))*100
    print("{} has {} null values of {} % of dataset".format(i,sum_na,round(percent_na,2)))
```

```
Sepal.Length has 10 null values of 6.67 % of dataset
Sepal.Width has 17 null values of 11.33 % of dataset
Petal.Length has 19 null values of 12.67 % of dataset
Petal.Width has 13 null values of 8.67 % of dataset
Species has 0 null values of 0.0 % of dataset
```

This dataset has a lot of outliers, so it would be best to replace it with median

In [83]:
```
df=data.fillna(data.median())
```

In [84]:
```
df
```

Out[84]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	3.0	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor
...	...	...	...	...	...
145	6.7	3.1	5.6	2.4	virginica
146	5.6	3.0	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa
148	6.4	3.1	4.5	1.8	virginica
149	5.8	2.6	4.0	1.3	versicolor

150 rows × 5 columns

In [85]:
```
# Checking null values in each columns
for i in df.columns:
    sum_na=df[i].isna().sum()
    percent_na=(sum_na/len(df))*100
    print("{} has {} null values of {} % of dataset".format(i,sum_na,round(percent_na,2)))
```

```
Sepal.Length has 0 null values of 0.0 % of dataset
Sepal.Width has 0 null values of 0.0 % of dataset
Petal.Length has 0 null values of 0.0 % of dataset
```

```
Petal.Width has 0 null values of 0.0 % of dataset
Species has 0 null values of 0.0 % of dataset
```

In [86]:

```
# More Data Exploration
for i in data.columns:
    print("Value counts of",i)
    print(data[i].value_counts())
    print('-'*30)
```

```
Value counts of Sepal.Length
5.0     10
5.1      9
6.3      9
6.4      7
5.5      7
5.7      7
6.7      7
5.8      6
5.6      6
6.5      5
4.9      5
6.0      5
4.8      5
6.1      4
7.7      4
5.4      4
4.6      4
6.2      4
5.2      4
6.9      3
4.4      3
5.9      3
7.2      3
6.6      2
4.7      2
6.8      2
7.9      1
7.6      1
4.5      1
73.0     1
4.3      1
7.4      1
7.0      1
0.0      1
5.3      1
49.0     1
Name: Sepal.Length, dtype: int64
------------------------------
Value counts of Sepal.Width
 3.0     23
 3.2     12
 2.8     12
 3.1     11
 3.4     10
 2.9      9
 2.7      8
 2.5      7
 3.5      6
 3.3      6
 2.6      5
 3.8      5
 2.3      3
 2.2      3
 3.6      3
 3.7      2
```

```
 29.0     1
 4.0      1
 4.2      1
-3.0      1
 3.9      1
 4.1      1
 30.0     1
 0.0      1
Name: Sepal.Width, dtype: int64
------------------------------
Value counts of Petal.Length
1.400     10
1.500      9
4.500      8
5.100      8
1.300      6
1.600      6
5.600      5
4.700      5
5.000      4
1.700      4
4.000      4
4.400      4
4.900      4
3.900      3
5.700      3
4.200      3
4.800      3
5.800      3
4.600      2
6.000      2
6.700      2
3.300      2
1.900      2
5.500      2
5.200      2
5.900      2
4.100      2
5.300      2
3.500      2
5.400      2
6.900      1
14.000     1
0.925      1
4.300      1
3.600      1
1.200      1
6.400      1
0.000      1
6.600      1
1.100      1
6.100      1
23.000     1
0.820      1
63.000     1
3.800      1
Name: Petal.Length, dtype: int64
------------------------------
Value counts of Petal.Width
0.2    26
1.3    13
1.5    12
2.3     8
1.8     8
1.4     8
2.0     7
```

```
2.1     6
0.4     6
1.0     6
0.3     6
0.1     5
1.6     4
1.2     4
1.9     4
2.5     3
1.1     3
2.4     3
2.2     2
0.6     1
1.7     1
0.5     1
Name: Petal.Width, dtype: int64
------------------------------
Value counts of Species
versicolor    50
virginica     50
setosa        50
Name: Species, dtype: int64
------------------------------
```

There are no special values either

In [87]:
```
# Describing
df.describe()
```

Out[87]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	6.505333	3.346667	4.456300	1.215333
std	6.571880	3.122932	5.388944	0.731069
min	0.000000	-3.000000	0.000000	0.100000
25%	5.100000	2.800000	1.700000	0.400000
50%	5.750000	3.000000	4.500000	1.300000
75%	6.400000	3.275000	5.100000	1.800000
max	73.000000	30.000000	63.000000	2.500000

Custom ruleset

In [88]:
```
def species(data):
    print("Checking Species should be one of the following values: setosa, versicolor or v
    x=lambda x:x in ['setosa','versicolor','virginica']
    a=np.array(data['Species'].apply(x))
    a=sum(a)
    return a
```

In [89]:
```
species(df)
```

Out[89]:
```
Checking Species should be one of the following values: setosa, versicolor or virginica
150
```

Out of 150, all 150 species are one of setosa, versicolor or virginica

In [90]:
```
def iris_positive(data):
    print("Checking measured numerical properties of an iris should be positive.")
    x=lambda x: x<0
    a=np.array(data.loc[:,data.columns!='Species'].apply(x))
    a=sum(a)
    return a
```

In [91]:
```
iris_positive(df)
```

```
Checking measured numerical properties of an iris should be positive.
```

Out[91]:
```
array([0, 1, 0, 0])
```

In [92]:
```
df[df['Sepal.Width']<0]
```

Out[92]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
15	5.0	-3.0	3.5	1.0	versicolor

There is 1 negative value in Sepal width rest are positive

In [93]:
```
print("petal length is atleast twice the petal width")
a=np.array(df['Petal.Length']>=2*df['Petal.Width'])
print(sum(a))
```

```
petal length is atleast twice the petal width
141
```

There are 141 records in our data which follows that petal length is atleast twice the petal width and 9 which dont

In [94]:
```
def sepal_length_check(data):
    print("The sepal length of an iris cannot exceed 30 cm.")
    x=lambda x:x>30
    a=np.array(df['Sepal.Length'].apply(x))
    a=sum(a)
    return a
```

In [95]:
```
sepal_length_check(data)
```

```
The sepal length of an iris cannot exceed 30 cm.
```

Out[95]:
```
2
```

There are 2 records in our data which sepal length exceeds 30 cm

In [96]:
```
print("The sepals of an iris are longer than its petals.")
a=df[df['Sepal.Length']>df['Petal.Length']]
```

```
The sepals of an iris are longer than its petals.
```

There are 146 instances where sepal length is greater than petal length and in 4 they don't

---

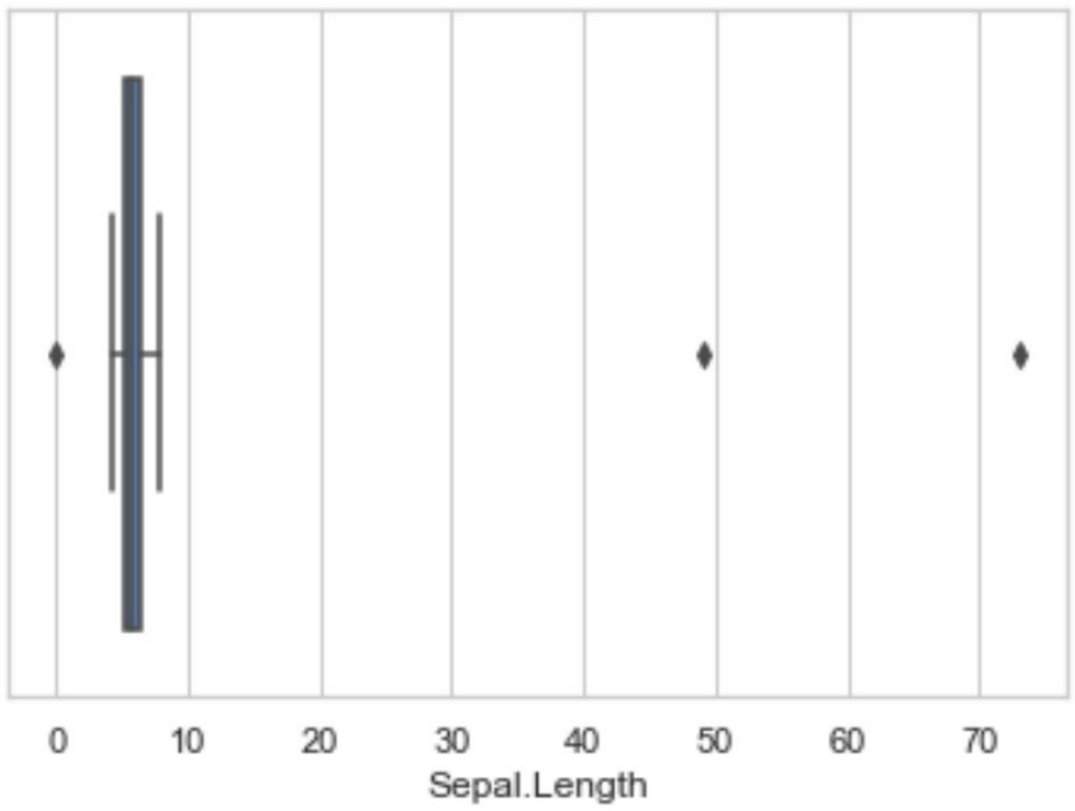
Outlier Detection using boxplot

In [97]:
```
data.columns
```

Out[97]:
```
Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width',
       'Species'],
      dtype='object')
```

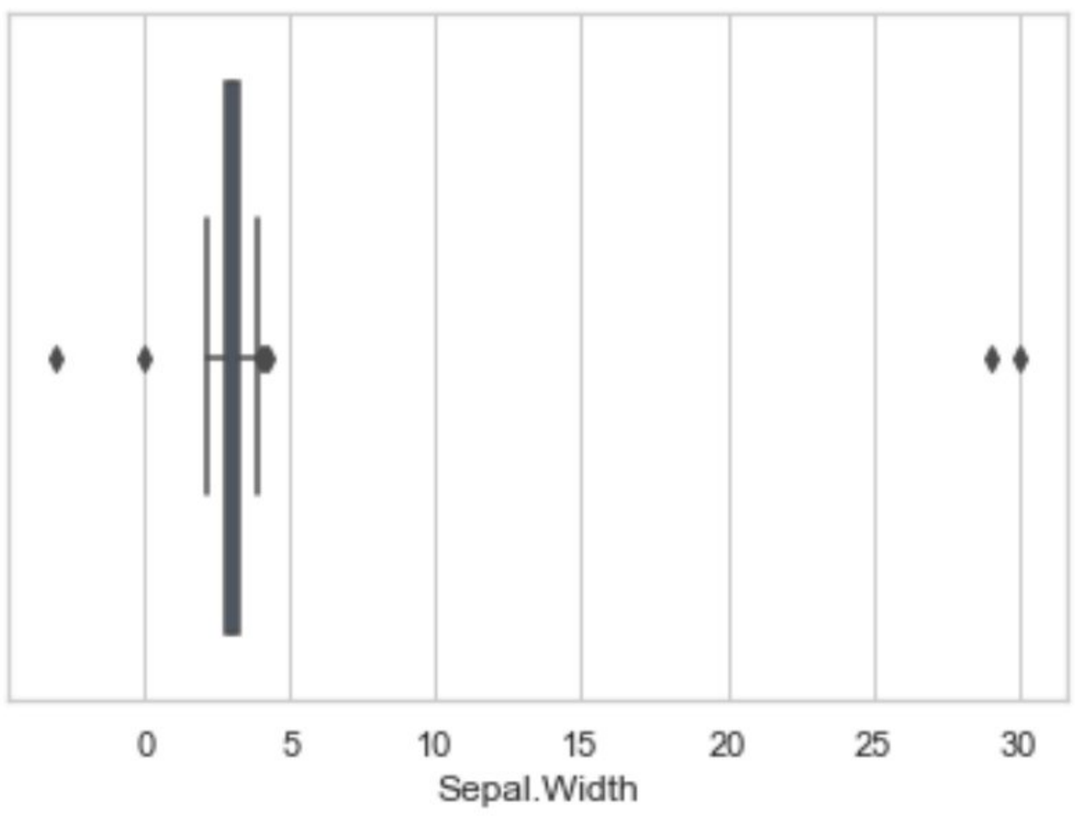
In [116…
```
sns.boxplot(x=df['Sepal.Length'])
```

Out[116…
```
<AxesSubplot:xlabel='Sepal.Length'>
```



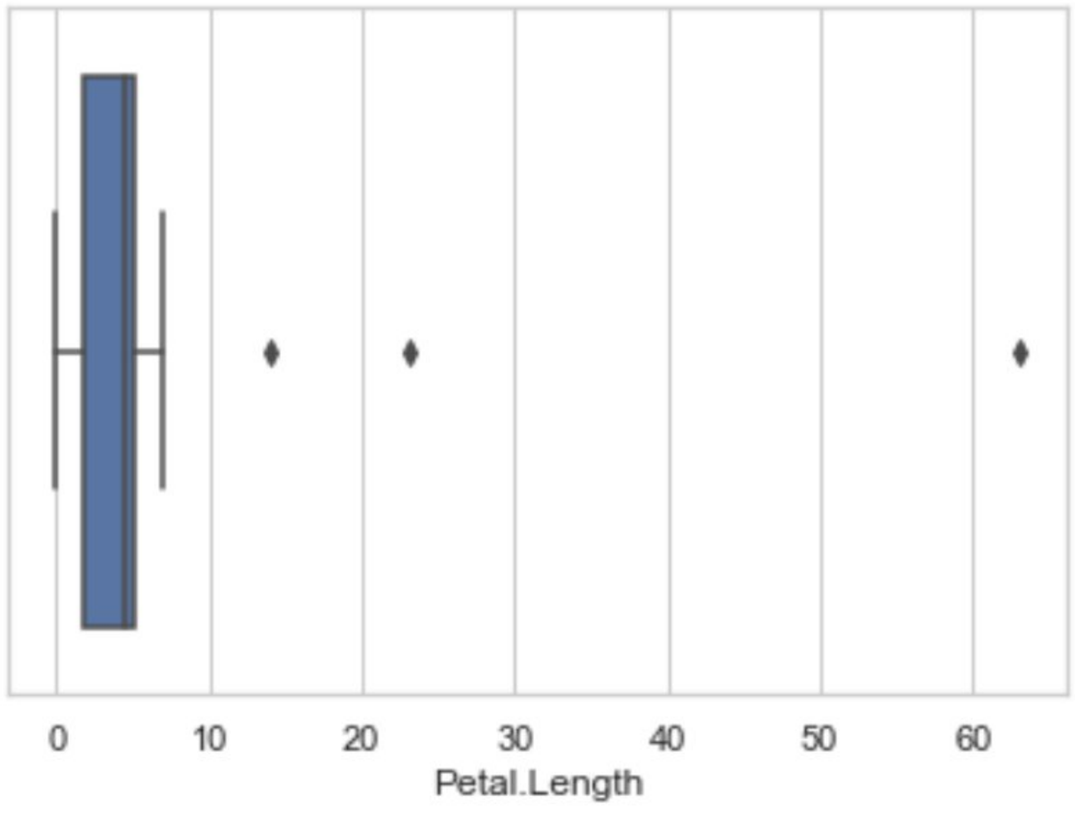
In [117…
```
sns.boxplot(df["Sepal.Width"])
```

Out[117…
```
<AxesSubplot:xlabel='Sepal.Width'>
```



In [118…
```
sns.boxplot(df["Petal.Length"])
```

Out[118…
```
<AxesSubplot:xlabel='Petal.Length'>
```

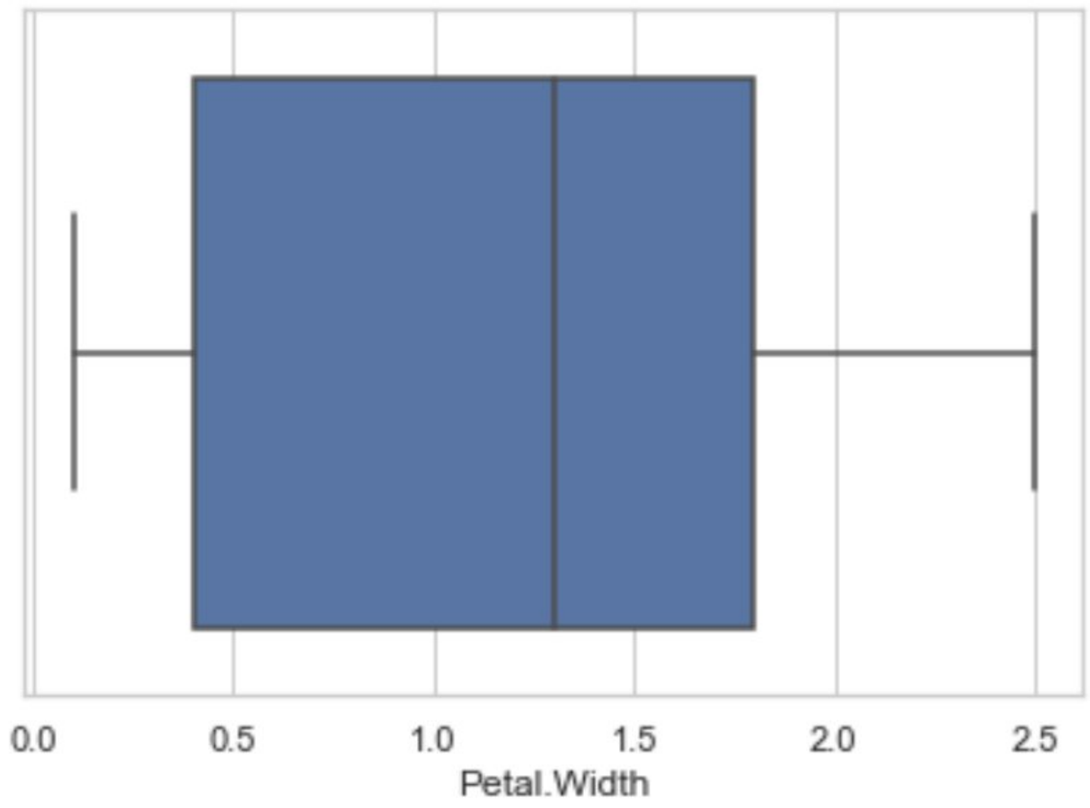


In [119…

```
sns.boxplot(df["Petal.Width"])
```

Out[119…

```
<AxesSubplot:xlabel='Petal.Width'>
```



Outliers in Sepal.Length

In [122…

```
det=df['Sepal.Length'].describe()
det
```

Out[122…

```
count    150.000000
mean       6.505333
std        6.571880
min        0.000000
25%        5.100000
50%        5.750000
75%        6.400000
max       73.000000
Name: Sepal.Length, dtype: float64
```

By looking sepal width we can see there are 3 outliers which are different from rest of data

Five Number Summary

In [124…

```
maxx=det[7]
minn=det[3]
q2=det[5]
```

```
q1=det[4]
q3=det[6]
```

In [126…

```
iqr=q3-q1
print(iqr)
```

```
1.3000000000000007
```

In [130…

```
downn=q1-1.5*iqr
downn
```

Out[130…

```
3.1499999999999986
```

In [129…

```
upp=q3+1.5*iqr
upp
```

Out[129…

```
8.350000000000001
```

In [136…

```
df[(df['Sepal.Length']>upp) | (df['Sepal.Length']<downn)]
```

Out[136…

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
27	73.0	29.0	63.0	1.3	virginica
42	0.0	3.0	1.3	0.4	setosa
124	49.0	30.0	14.0	2.0	setosa

by IQR we have found too 3 outliers in Sepal.Length columns

Practical 3

Name - Ajit singh
Roll No. - 19058570034

Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

In [66]:
```
# Loading the wine dataset
from sklearn.datasets import load_wine # Wine Dataset
from sklearn.preprocessing import StandardScaler # Standardization Library
import pandas as pd
import numpy as np
import seaborn as sns
```

In [67]:
```
wine=load_wine()
print(wine.feature_names)
print(wine.target_names)
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'f
lavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od28
0/od315_of_diluted_wines', 'proline']
['class_0' 'class_1' 'class_2']
```

In [68]:
```
features=wine.feature_names
target=wine.target_names
df = pd.DataFrame(wine.data,columns=wine.feature_names)
df['target']=pd.Categorical.from_codes(wine.target,wine.target_names)
df
```

Out[68]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_ph
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

178 rows × 14 columns

In [69]:
```
df[features].describe()
```

Out[69]:

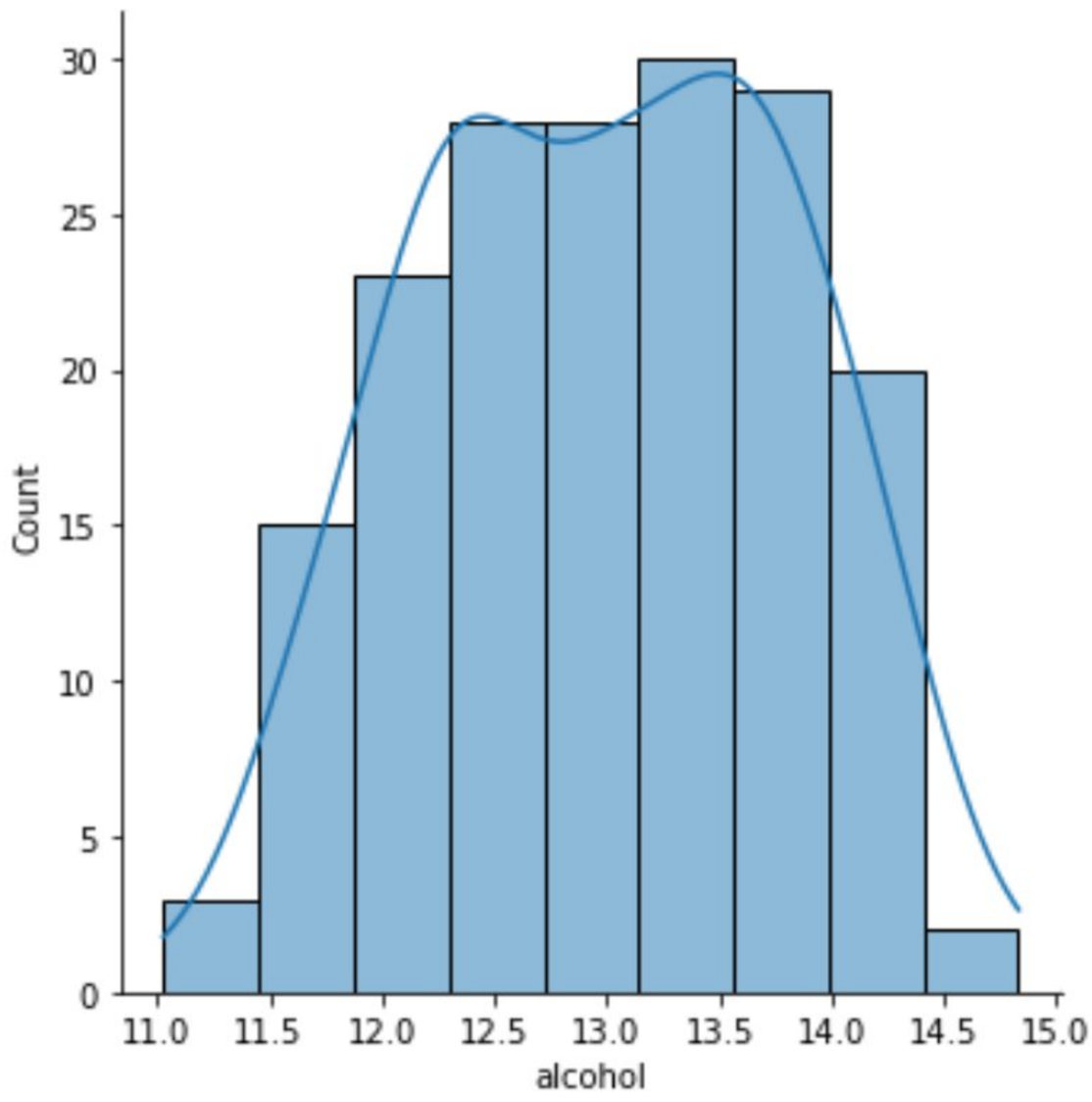
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000

In [70]:
```
new_df=df.copy()
```

By looking mean and standard deviation we can tell some attributes are not standardized.

In [71]:
```
sns.displot(df['alcohol'],kde=True)
```

Out[71]:
```
<seaborn.axisgrid.FacetGrid at 0x1b4af854640>
```



Standardized attribute follow normal distribution which Alcohol distribution does not.

In [72]:
```
# Standardizing all features
scaler=StandardScaler()
for i in features:
    scale=scaler.fit(new_df[[i]])
    new_df[i]=scale.transform(new_df[[i]])
new_df
```

Out[72]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflav
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	
...	...	...	...	...	...	...	...	
173	0.876275	2.974543	0.305159	0.301803	-0.332922	-0.985614	-1.424900	
174	0.493343	1.412609	0.414820	1.052516	0.158572	-0.793334	-1.284344	
175	0.332758	1.744744	-0.389355	0.151661	1.422412	-1.129824	-1.344582	
176	0.209232	0.227694	0.012732	0.151661	1.422412	-1.033684	-1.354622	
177	1.395086	1.583165	1.365208	1.502943	-0.262708	-0.392751	-1.274305	

178 rows × 14 columns

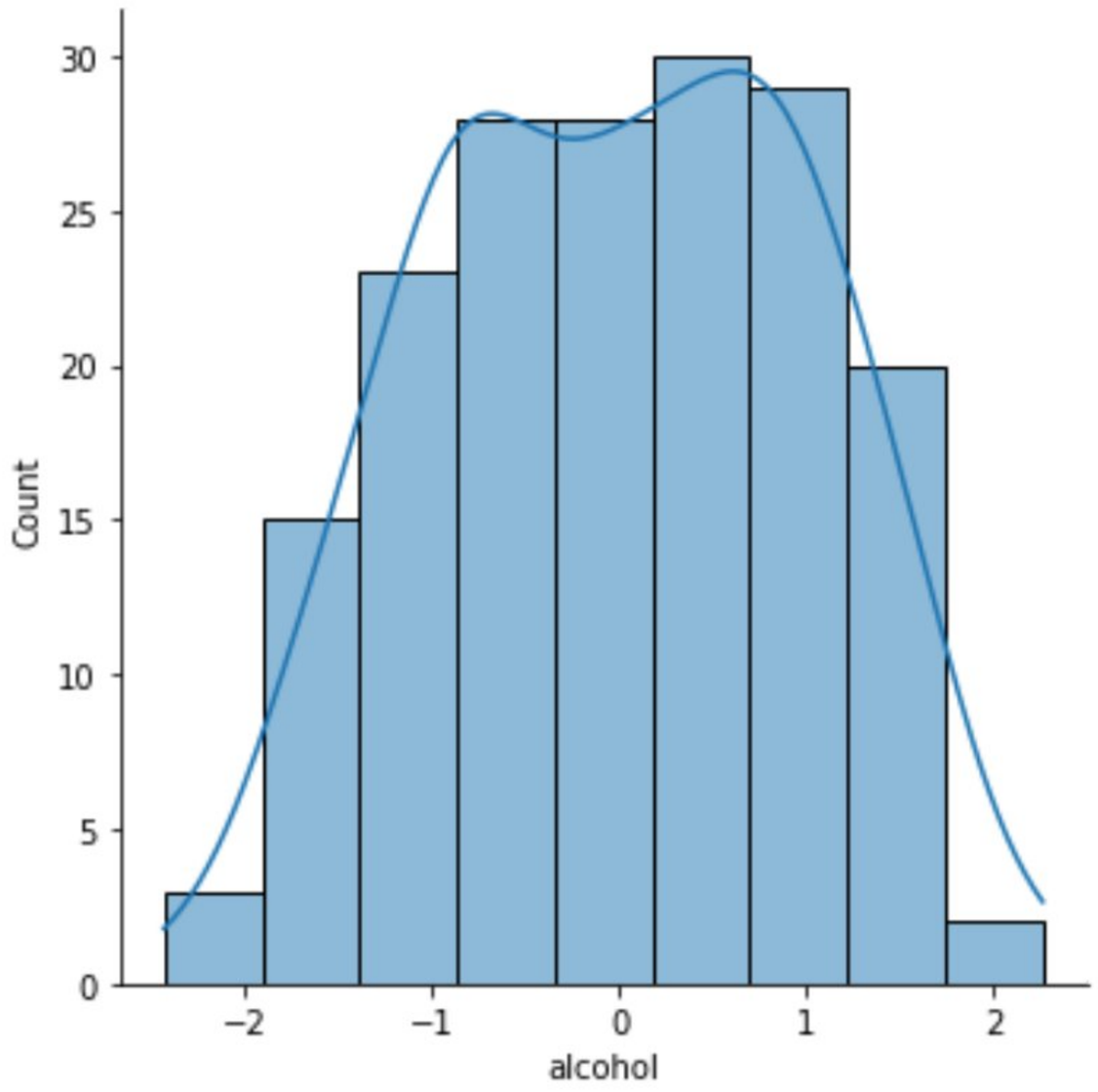
In [73]:
```
new_df.describe()
```

Out[73]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phen
count	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+
mean	-8.619821e-16	-8.357859e-17	-8.657245e-16	-1.160121e-16	-1.995907e-17	-2.972030e-
std	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+
min	-2.434235e+00	-1.432983e+00	-3.679162e+00	-2.671018e+00	-2.088255e+00	-2.107246e+
25%	-7.882448e-01	-6.587486e-01	-5.721225e-01	-6.891372e-01	-8.244151e-01	-8.854682e-
50%	6.099988e-02	-4.231120e-01	-2.382132e-02	1.518295e-03	-1.222817e-01	9.595986e-
75%	8.361286e-01	6.697929e-01	6.981085e-01	6.020883e-01	5.096384e-01	8.089974e-
max	2.259772e+00	3.109192e+00	3.156325e+00	3.154511e+00	4.371372e+00	2.539515e+

In [74]:
```
sns.displot(new_df['alcohol'],kde=True)
```

Out[74]:
```
<seaborn.axisgrid.FacetGrid at 0x1b4af74b5e0>
```



Iris Dataset

In [80]:
```
from sklearn.datasets import load_iris
iris=load_iris()
features=iris.feature_names
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df['Species']=pd.Categorical.from_codes(iris.target,iris.target_names)
df
```

Out[80]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [76]:
```
df.describe()
```

Out[76]:

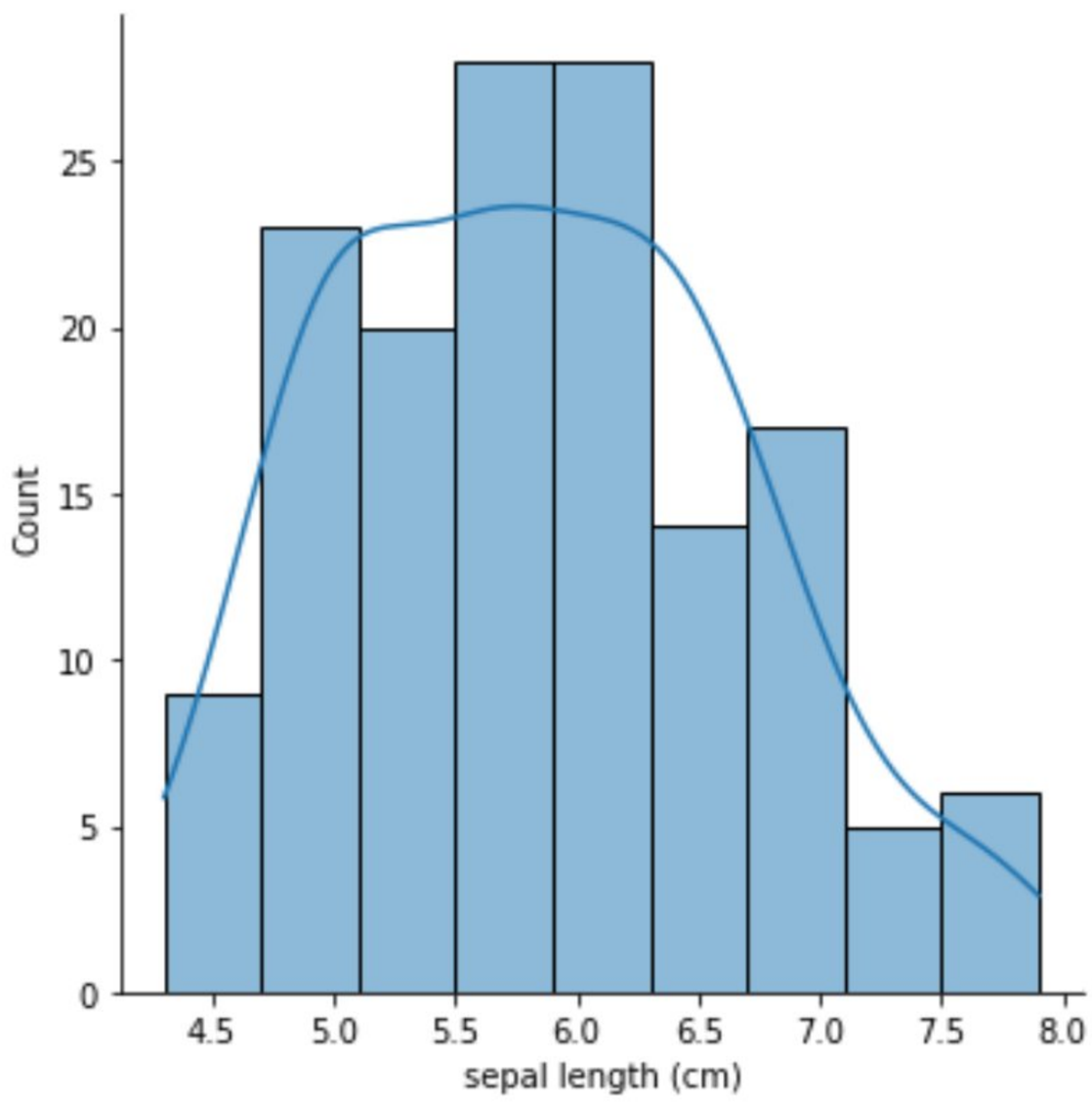
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

All attributes are not Standardized.

In [79]:
```
sns.displot(df['sepal length (cm)'],kde=True)
```

Out[79]:
```
<seaborn.axisgrid.FacetGrid at 0x1b4ad8a2f40>
```



In [81]:
```
iris_copy=df.copy()
```

In [85]:
```
# Standardizing all features
scaler=StandardScaler()
for i in features:
    scale=scaler.fit(iris_copy[[i]])
    iris_copy[i]=scale.transform(iris_copy[[i]])
iris_copy
```

Out[85]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	-0.900681	1.019004	-1.340227	-1.315444	setosa
1	-1.143017	-0.131979	-1.340227	-1.315444	setosa
2	-1.385353	0.328414	-1.397064	-1.315444	setosa
3	-1.506521	0.098217	-1.283389	-1.315444	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
4	-1.021849	1.249201	-1.340227	-1.315444	setosa
...	...	...	...	...	...
145	1.038005	-0.131979	0.819596	1.448832	virginica
146	0.553333	-1.282963	0.705921	0.922303	virginica
147	0.795669	-0.131979	0.819596	1.053935	virginica
148	0.432165	0.788808	0.933271	1.448832	virginica
149	0.068662	-0.131979	0.762758	0.790671	virginica

150 rows × 5 columns

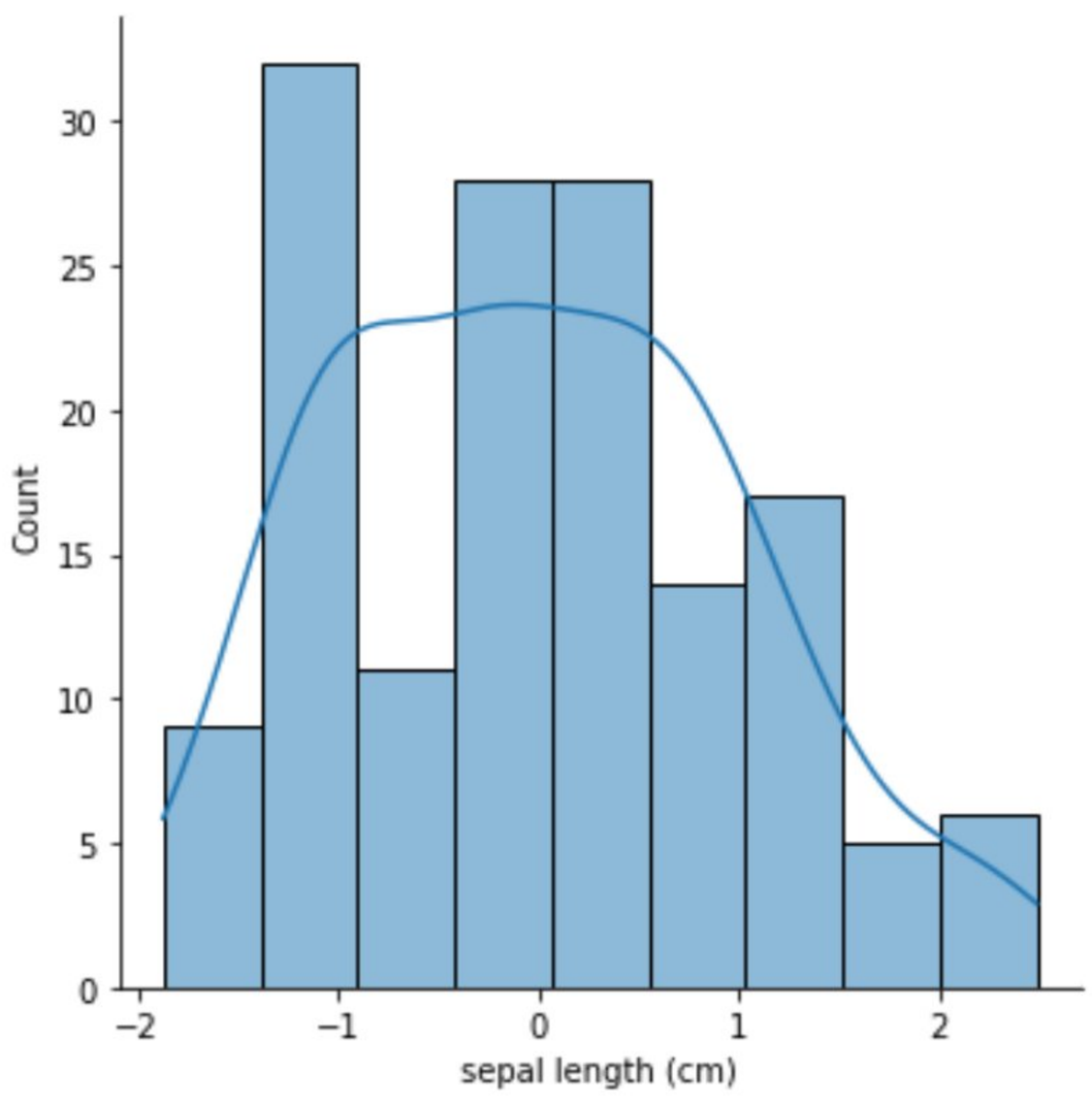
In [86]:
```
iris_copy.describe()
```

Out[86]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	-2.775558e-16	-9.695948e-16	-8.652338e-16	-4.662937e-16
std	1.003350e+00	1.003350e+00	1.003350e+00	1.003350e+00
min	-1.870024e+00	-2.433947e+00	-1.567576e+00	-1.447076e+00
25%	-9.006812e-01	-5.923730e-01	-1.226552e+00	-1.183812e+00
50%	-5.250608e-02	-1.319795e-01	3.364776e-01	1.325097e-01
75%	6.745011e-01	5.586108e-01	7.627583e-01	7.906707e-01
max	2.492019e+00	3.090775e+00	1.785832e+00	1.712096e+00

In [87]:
```
sns.displot(iris_copy['sepal length (cm)'],kde=True)
```

Out[87]:
```
<seaborn.axisgrid.FacetGrid at 0x1b4af8db220>
```



In [ ]:

Market Basket Analysis (Practical 4)

> Name - Ajit Singh
> Roll No. - 19058570034

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
from apyori import apriori
```

Reading data

In [2]:
```
df=pd.read_excel("market.xlsx",header=None)
df
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	ho
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	N
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

7501 rows × 20 columns

Data Preprocessing

In [3]:
```
data=df.iloc[1:,:]
```

In [4]:
```
data.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Removing null values

In [5]:
```
baskets=[]
for item in range(1,len(data)):
    store=data.iloc[item,:]
    lst=[x for x in store if pd.isnull(x)==False]
    if len(lst)==0:
        pass
    else:
        baskets.append(lst)
```

In [6]:
```
print(baskets[:20])
```

```
[['chutney'], ['turkey', 'avocado'], ['mineral water', 'milk', 'energy bar', 'whole wheat
rice', 'green tea'], ['low fat yogurt'], ['whole wheat pasta', 'french fries'], ['soup',
'light cream', 'shallot'], ['frozen vegetables', 'spaghetti', 'green tea'], ['french frie
s'], ['eggs', 'pet food'], ['cookies'], ['turkey', 'burgers', 'mineral water', 'eggs', 'co
oking oil'], ['spaghetti', 'champagne', 'cookies'], ['mineral water', 'salmon'], ['mineral
water'], ['shrimp', 'chocolate', 'chicken', 'honey', 'oil', 'cooking oil', 'low fat yogur
t'], ['turkey', 'eggs'], ['turkey', 'fresh tuna', 'tomatoes', 'spaghetti', 'mineral wate
r', 'black tea', 'salmon', 'eggs', 'chicken', 'extra dark chocolate'], ['meatballs', 'mil
k', 'honey', 'french fries', 'protein bar'], ['red wine', 'shrimp', 'pasta', 'pepper', 'eg
gs', 'chocolate', 'shampoo'], ['rice', 'sparkling water']]
```

Applying Rule association mining

By using mlxtend

In [7]:
```
from mlxtend.preprocessing import TransactionEncoder
```

In [8]:
```
te=TransactionEncoder()
tr_arr=te.fit_transform(baskets)
```

In [9]:
```
df1 = pd.DataFrame(tr_arr, columns=te.columns_)
df1
```

Out[9]:

	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	body spray	...	turkey
0	False	False	False	False	False	False	False	False	False	False	...	False

	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	body spray	...	turkey
1	False	False	False	True	False	False	False	False	False	False	...	True
2	False	False	False	False	False	False	False	False	False	False	...	False
3	False	False	False	False	False	False	False	False	False	False	...	False
4	False	False	False	False	False	False	False	False	False	False	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
7494	False	False	False	False	False	False	False	False	False	False	...	False
7495	False	False	False	False	False	False	False	False	False	False	...	False
7496	False	False	False	False	False	False	False	False	False	False	...	False
7497	False	False	False	False	False	False	False	False	False	False	...	False
7498	False	False	False	False	False	False	False	False	False	False	...	False

7499 rows × 119 columns

In [10]:
```
from mlxtend.frequent_patterns import apriori
```

In [11]:
```
apriori(df1,min_support=0.05,use_colnames=True)
```

Out[11]:

	support	itemsets
0	0.087078	(burgers)
1	0.081077	(cake)
2	0.060008	(chicken)
3	0.163889	(chocolate)
4	0.080411	(cookies)
5	0.051073	(cooking oil)
6	0.179624	(eggs)
7	0.079344	(escalope)
8	0.170956	(french fries)
9	0.063208	(frozen smoothie)
10	0.095346	(frozen vegetables)
11	0.052407	(grated cheese)
12	0.132018	(green tea)
13	0.098280	(ground beef)
14	0.076410	(low fat yogurt)
15	0.129617	(milk)
16	0.238298	(mineral water)
17	0.065742	(olive oil)

	support	itemsets
18	0.095079	(pancakes)
19	0.071343	(shrimp)
20	0.050540	(soup)
21	0.174157	(spaghetti)
22	0.068409	(tomatoes)
23	0.062542	(turkey)
24	0.058541	(whole wheat rice)
25	0.052674	(chocolate, mineral water)
26	0.050940	(mineral water, eggs)
27	0.059741	(mineral water, spaghetti)

In [12]:
```
apriori(df1,min_support=0.075,use_colnames=True)
```

Out[12]:

	support	itemsets
0	0.087078	(burgers)
1	0.081077	(cake)
2	0.163889	(chocolate)
3	0.080411	(cookies)
4	0.179624	(eggs)
5	0.079344	(escalope)
6	0.170956	(french fries)
7	0.095346	(frozen vegetables)
8	0.132018	(green tea)
9	0.098280	(ground beef)
10	0.076410	(low fat yogurt)
11	0.129617	(milk)
12	0.238298	(mineral water)
13	0.095079	(pancakes)
14	0.174157	(spaghetti)

In [13]:
```
apriori(df1,min_support=0.025,use_colnames=True)
```

Out[13]:

	support	itemsets
0	0.033204	(avocado)
1	0.033738	(brownies)
2	0.087078	(burgers)
3	0.030137	(butter)

	support	itemsets
4	0.081077	(cake)
...	...	...
68	0.035471	(spaghetti, milk)
69	0.027470	(olive oil, mineral water)
70	0.033738	(mineral water, pancakes)
71	0.059741	(mineral water, spaghetti)
72	0.025203	(pancakes, spaghetti)

73 rows × 2 columns

In [14]:
```
apriori(df1,min_support=0.015,use_colnames=True)
```

Out[14]:

	support	itemsets
0	0.020269	(almonds)
1	0.033204	(avocado)
2	0.033738	(brownies)
3	0.087078	(burgers)
4	0.030137	(butter)
...	...	...
147	0.020936	(spaghetti, tomatoes)
148	0.016536	(spaghetti, turkey)
149	0.015869	(chocolate, mineral water, spaghetti)
150	0.017069	(mineral water, spaghetti, ground beef)
151	0.015735	(mineral water, spaghetti, milk)

152 rows × 2 columns

From Efficient apriori

In [15]:
```
from efficient_apriori import apriori
```

In [16]:
```
itemset,rules=apriori(baskets,min_support=0.05,min_confidence=0.075)
print(itemset)
print(rules)
```

```
{1: {('turkey',): 469, ('mineral water',): 1787, ('milk',): 972, ('whole wheat rice',): 43
9, ('green tea',): 990, ('low fat yogurt',): 573, ('french fries',): 1282, ('soup',): 379,
('frozen vegetables',): 715, ('spaghetti',): 1306, ('eggs',): 1347, ('cookies',): 603, ('b
urgers',): 653, ('cooking oil',): 383, ('shrimp',): 535, ('chocolate',): 1229, ('chicke
n',): 450, ('tomatoes',): 513, ('pancakes',): 713, ('grated cheese',): 393, ('ground bee
f',): 737, ('frozen smoothie',): 474, ('escalope',): 595, ('cake',): 608, ('olive oil',):
493}, 2: {('chocolate', 'mineral water'): 395, ('eggs', 'mineral water'): 382, ('mineral w
ater', 'spaghetti'): 448}}
[{mineral water} -> {chocolate}, {chocolate} -> {mineral water}, {mineral water} -> {egg
```

```
s}, {eggs} -> {mineral water}, {spaghetti} -> {mineral water}, {mineral water} -> {spaghet
ti}]
```

In [17]:

```
from efficient_apriori import apriori
itemset,rules=apriori(baskets,min_support=0.06,min_confidence=0.60)
print(itemset)
print(rules)
```

```
{1: {('turkey',): 469, ('mineral water',): 1787, ('milk',): 972, ('green tea',): 990, ('lo
w fat yogurt',): 573, ('french fries',): 1282, ('frozen vegetables',): 715, ('spaghett
i',): 1306, ('eggs',): 1347, ('cookies',): 603, ('burgers',): 653, ('shrimp',): 535, ('cho
colate',): 1229, ('chicken',): 450, ('tomatoes',): 513, ('pancakes',): 713, ('ground bee
f',): 737, ('frozen smoothie',): 474, ('escalope',): 595, ('cake',): 608, ('olive oil',):
493}}
[]
```

Practical 5

Name - Ajit Singh
Roll No. - 19058570034

Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:

5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3%

5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) Cross-Validation. Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
df=pd.read_excel("data.xlsx")
df
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactne
0	842302.0	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517.0	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903.0	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301.0	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402.0	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	...	...
564	926424.0	M	21.56	22.39	142.00	1479.0	0.11100	
565	926682.0	M	20.13	28.25	131.20	1261.0	0.09780	
566	926954.0	M	16.60	28.08	108.30	858.1	0.08455	
567	927241.0	M	20.60	29.33	140.10	1265.0	0.11780	
568	92751.0	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 32 columns

Splitting target and features data

In [2]:
```
from sklearn.model_selection import train_test_split
X=df.iloc[:,2:-1] # Features
y=df['diagnosis'].values
y
```

Out[2]:
```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
```

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

In [3]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

In [27]:

```
# Test Size = 25 %
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=100)
# Test Size = 33.3%
x_train2,x_test2,y_train2,y_test2=train_test_split(X,y,test_size=0.33,random_state=100)
print(x_train.shape)
print(x_test.shape)
print(x_train2.shape)
print(x_test2.shape)
```

```
(426, 29)
(143, 29)
(381, 29)
(188, 29)
```

In [28]:

```
# Building models
classifer_nb1=GaussianNB()
classifer_nb2=GaussianNB()
classifer_nb1.fit(x_train,y_train)
```

```
classifer_nb2.fit(x_train2,y_train2)

classifier_kn1=KNeighborsClassifier()
classifier_kn2=KNeighborsClassifier()
classifier_kn1.fit(x_train,y_train)
classifier_kn2.fit(x_train2,y_train2)

classifier_dt1=DecisionTreeClassifier()
classifier_dt2=DecisionTreeClassifier()
classifier_dt1.fit(x_train,y_train)
classifier_dt2.fit(x_train2,y_train2)
```

Out[28]:
```
DecisionTreeClassifier()
```

In [30]:
```
y_nb1=classifer_nb1.predict(x_test)
y_nb2=classifer_nb2.predict(x_test2)
y_nb1
```

Out[30]:
```
array(['M', 'B', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B',
       'M', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'M',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M',
       'M', 'B', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B'],
      dtype='<U1')
```

In [33]:
```
y_kn1=classifier_kn1.predict(x_test)
y_kn2=classifier_kn2.predict(x_test2)
y_kn1
```

Out[33]:
```
array(['M', 'B', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'M', 'M',
       'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B',
       'M', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B',
       'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'M',
       'M', 'B', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B'],
      dtype=object)
```

In [36]:
```
y_dt1=classifier_dt1.predict(x_test)
y_dt2=classifier_dt2.predict(x_test2)
y_dt1
```

Out[36]:
```
array(['M', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M',
       'M', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B',
       'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M',
       'M', 'B', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B',
```

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

Metrics

In [42]:

```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score
print("When Test Size= 25 %")
print("Accuracy for Naive Bayes : ",accuracy_score(y_test,y_nb1))
print("Accuracy for K Neighbours : " ,accuracy_score(y_test,y_kn1))
print("Accuracy for Decision Tree : ",accuracy_score(y_test,y_dt1))
print("\n<---------------------------------------------------->\n")
print("When Test Size =33.3 %")
print("Accuracy for Naive Bayes : ",accuracy_score(y_test2,y_nb2))
print("Accuracy for K Neighbours : " ,accuracy_score(y_test2,y_kn2))
print("Accuracy for Decision Tree : ",accuracy_score(y_test2,y_dt2))
```

```
When Test Size= 25 %
Accuracy for Naive Bayes :  0.9440559440559441
Accuracy for K Neighbours :  0.958041958041958
Accuracy for Decision Tree :  0.9230769230769231

<---------------------------------------------------->

When Test Size =33.3 %
Accuracy for Naive Bayes :  0.9308510638297872
Accuracy for K Neighbours :  0.9468085106382979
Accuracy for Decision Tree :  0.9095744680851063
```

Cross Validation

In [47]:

```
from sklearn.model_selection import cross_val_score
score_nb=cross_val_score(classifer_nb1,x_train,y_train,cv=10).mean()
score_kn=cross_val_score(classifier_kn1,x_train,y_train,cv=10).mean()
score_dt=cross_val_score(classifier_dt1,x_train,y_train,cv=10).mean()
print("When Test Size= 25 %")
print("Accuracy of 10 Fold CV of Naive bayes : ",score_nb)
print("Accuracy of 10 Fold CV of K Neighbours : ",score_kn)
print("Accuracy of 10 Fold CV of Decision Tree : ",score_dt)
print("\n<---------------------------------------------------->\n")
score_nb2=cross_val_score(classifer_nb2,x_train2,y_train2,cv=10).mean()
score_kn2=cross_val_score(classifier_kn2,x_train2,y_train2,cv=10).mean()
score_dt2=cross_val_score(classifier_dt2,x_train2,y_train2,cv=10).mean()
print("When Test Size= 33.3 %")
print("Accuracy of 10 Fold CV of Naive bayes : ",score_nb2)
print("Accuracy of 10 Fold CV of K Neighbours : ",score_kn2)
print("Accuracy of 10 Fold CV of Decision Tree : ",score_dt2)
```

```
When Test Size= 25 %
Accuracy of 10 Fold CV of Naive bayes :  0.9437430786267995
Accuracy of 10 Fold CV of K Neighbours :  0.9249723145071982
Accuracy of 10 Fold CV of Decision Tree :  0.9224806201550388

<---------------------------------------------------->

When Test Size= 33.3 %
Accuracy of 10 Fold CV of Naive bayes :  0.9501349527665315
Accuracy of 10 Fold CV of K Neighbours :  0.9237516869095816
Accuracy of 10 Fold CV of Decision Tree :  0.9211875843454791
```

Random sub sampling

In [12]:

```
from imblearn.under_sampling import RandomUnderSampler
rus=RandomUnderSampler(random_state=40)
```

```
x_rus,y_rus=rus.fit_resample(X,y)
```

In [14]:

```
classifer_nb.fit(x_rus,y_rus)
y_nb=classifer_nb.predict(x_rus)
y_nb
```

Out[14]:

```
array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M',
       'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M',
       'B', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B',
       'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M',
       'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M'], dtype='<U1')
```

In [15]:

```
classifier_kn.fit(x_rus,y_rus)
y_kn=classifier_kn.predict(x_rus)
y_kn
```

Out[15]:

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

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

In [16]:

```
classifier_dt.fit(x_rus,y_rus)
y_dt=classifier_dt.predict(x_rus)
y_dt
```

Out[16]:

```
array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M'], dtype=object)
```

In [18]:

```
scale=StandardScaler()
scale.fit(X)
```

Out[18]:

```
StandardScaler()
```

In [26]:

```
scaled=scale.transform(X)
scaled_df=pd.DataFrame(scaled,columns=X.columns)
scaled_df
```

Out[26]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.65287
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.02384
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.36347
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.91589
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.37101
...	...	...	...	...	...	...	.
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.94728
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.69304
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.04658
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.29694
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.11487

569 rows × 29 columns

Practical 6

> Name - Ajit Singh
> Roll No. - 19058570034

Importing Libraries

In [117…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mp
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans,AgglomerativeClustering,DBSCAN
from scipy.cluster.hierarchy import dendrogram
from sklearn.preprocessing import LabelEncoder, StandardScaler
from datetime import datetime
```

Plotting Theme

In [31]:

```
#set matplotlib parameters
mp.rc_file_defaults()
rc = {'axes.facecolor':'white','grid.color': '.8',}
plt.rcParams.update(rc)

#set seaborn color palette
sns.set_palette(sns.color_palette("RdPu", 6))
palette = ["#fcd7d3","#faabb8","#f667a0","#cd238e","#cd238e"]
```

1. Loading Data

In [32]:

```
data = pd.read_csv("marketing_campaign.csv", sep="\t")
print("Data loaded with", len(data), "rows")
data.head()
```

```
Data loaded with 2240 rows
```

Out[32]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	.
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	.
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	.
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	.
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	.
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	.

5 rows × 29 columns

2. Data Cleaning

In [33]:
```
#overview of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

Data cleaning steps:

- **Drop un-needed columns**
- **Drop rows with empty income values (These form a small percentage of data)**
- **Change the Dt_Customer to a DateTime format**

In [34]:
```
#drop un-needed columns
data= data.drop(["ID","Z_CostContact", "Z_Revenue"], axis=1)
#drop rows with missing values
data= data.dropna()
#change date format
data["Dt_Customer"]= pd.to_datetime(data["Dt_Customer"])
```

3. Preprocessing

3.1. Data Transformation

In [35]:
```
#give each feature a smaller set of values
```

```
edu= {"Basic": "Undergraduate", "2n Cycle": "Undergraduate", "Graduation": "Graduate", "Ma
data["Education"]= data["Education"].replace(edu)

status= {"YOLO": "Single", "Absurd": "Single", "Alone": "Single", "Widow": "Single", "Divo
data["Marital_Status"]= data["Marital_Status"].replace(status)

#new values
print("Education Values: ", data["Education"].unique())
print("Marital_Status Values:", data["Marital_Status"].unique())
```

```
Education Values:  ['Graduate' 'Postgraduate' 'Undergraduate']
Marital_Status Values: ['Single' 'Taken']
```

We can then transform some of the features as follows:

In [36]:

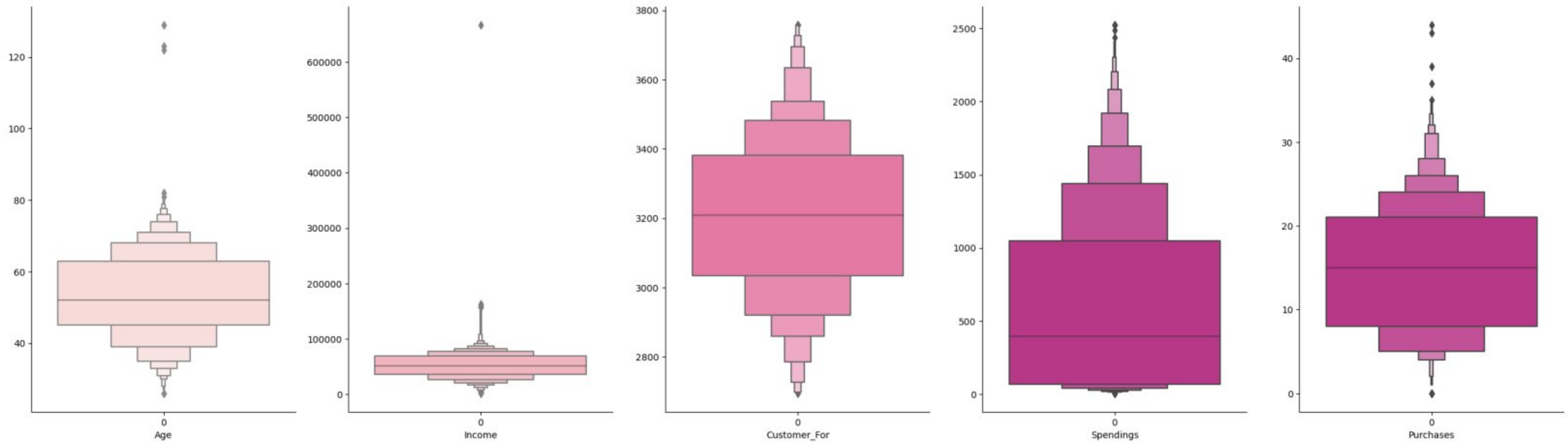
```
#finding customer age
data["Age"]= datetime.now().year-data["Year_Birth"]

#finding family size and number of children
data["Children_Count"]= data["Kidhome"]+data["Teenhome"]
data["Family_Size"]= 1+data["Children_Count"]+data["Marital_Status"].replace({"Taken": 1,

#finding number of days since person became a customer
data["Customer_For"]= (datetime.now()-data["Dt_Customer"]).dt.days

#finding total spendings of customer
data["Spendings"]= data["MntWines"]+data["MntFruits"]+data["MntFishProducts"]+data["MntMea

#finding total number of purchases of customer
data["Purchases"]= data["NumDealsPurchases"]+data["NumWebPurchases"]+data["NumCatalogPurch

#finding total number of accepted campaigns
data["Accepted_Campaigns"]= data["AcceptedCmp1"]+data["AcceptedCmp2"]+data["AcceptedCmp3"]

#dropping un-needed columns
data= data.drop(["Year_Birth","Dt_Customer"],axis=1)
```

3.2. Outlier Removal

Now we need to check the distribution of values in the dataset to remove any outliers

In [37]:

```
#examine the boxplots of different features
features= ["Age", "Income", "Customer_For", "Spendings", "Purchases"]

#create plots
fig, axs = plt.subplots(ncols=len(features),figsize=(6*len(features),8))
for i in range(len(features)):
  sns.boxenplot(data=data[features[i]],
                showfliers=True,
                ax=axs[i],
                palette=[palette[i]]
                ).set(xlabel=features[i])
  sns.despine()
```



In [38]:
```
#checking the number of outliers in age and income
print("Number of customers above the age of 100= ", len(data[data["Age"]>100]))
print("Number of customers with income above 200,000= ", len(data[data["Income"]>200000]))
```

```
Number of customers above the age of 100=  3
Number of customers with income above 200,000=  1
```

We can see some outliers in the age, income and purchases. Those in the age and income will be removed as they are very few data points. The ones in the purchases will be kept as the range of values is considerable.

In [39]:
```
data= data.drop(data[(data["Age"]>100) | (data["Income"]>200000)].index)
print("Current data points count= ",len(data))
```

```
Current data points count=  2212
```

In [40]:
```
dataCopy= data.copy()
data= data.drop(["Kidhome","Teenhome"],axis=1)
```

3.3. Encoding Categorical Features

The 2 categorial features we have can be encoded as follows:

- **Education: using a label encoder since it's considered ordinal**
- **Marital_Status: using one hot encoding since it's considered nominal**

In [41]:
```
#label encode education
encoder= LabelEncoder()
data["Education"]= data[["Education"]].apply(encoder.fit_transform)

#hot encode marital_status
data = pd.get_dummies(data)

#check data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2212 entries, 0 to 2239
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Education             2212 non-null   int32
 1   Income                2212 non-null   float64
 2   Recency               2212 non-null   int64
 3   MntWines              2212 non-null   int64
 4   MntFruits             2212 non-null   int64
```

```
 5   MntMeatProducts      2212 non-null   int64
 6   MntFishProducts      2212 non-null   int64
 7   MntSweetProducts     2212 non-null   int64
 8   MntGoldProds         2212 non-null   int64
 9   NumDealsPurchases    2212 non-null   int64
 10  NumWebPurchases      2212 non-null   int64
 11  NumCatalogPurchases  2212 non-null   int64
 12  NumStorePurchases    2212 non-null   int64
 13  NumWebVisitsMonth    2212 non-null   int64
 14  AcceptedCmp3         2212 non-null   int64
 15  AcceptedCmp4         2212 non-null   int64
 16  AcceptedCmp5         2212 non-null   int64
 17  AcceptedCmp1         2212 non-null   int64
 18  AcceptedCmp2         2212 non-null   int64
 19  Complain             2212 non-null   int64
 20  Response             2212 non-null   int64
 21  Age                  2212 non-null   int64
 22  Children_Count       2212 non-null   int64
 23  Family_Size          2212 non-null   int64
 24  Customer_For         2212 non-null   int64
 25  Spendings            2212 non-null   int64
 26  Purchases            2212 non-null   int64
 27  Accepted_Campaigns   2212 non-null   int64
 28  Marital_Status_Single  2212 non-null   uint8
 29  Marital_Status_Taken   2212 non-null   uint8
dtypes: float64(1), int32(1), int64(26), uint8(2)
memory usage: 496.8 KB
```

3.4. Feature Scaling

In [42]:

```
#scale features
scaler= StandardScaler()
data = pd.DataFrame(scaler.fit_transform(data),columns = data.columns)

#check
data.head()
```

Out[42]:

	Education	Income	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts
0	-0.893586	0.287105	0.310353	0.977660	1.552041	1.690293	2.453472	1.483713
1	-0.893586	-0.260882	-0.380813	-0.872618	-0.637461	-0.718230	-0.651004	-0.634019
2	-0.893586	0.913196	-0.795514	0.357935	0.570540	-0.178542	1.339513	-0.147184
3	-0.893586	-1.176114	-0.795514	-0.872618	-0.561961	-0.655787	-0.504911	-0.585335
4	0.571657	0.294307	1.554453	-0.392257	0.419540	-0.218684	0.152508	-0.001133

5 rows × 30 columns

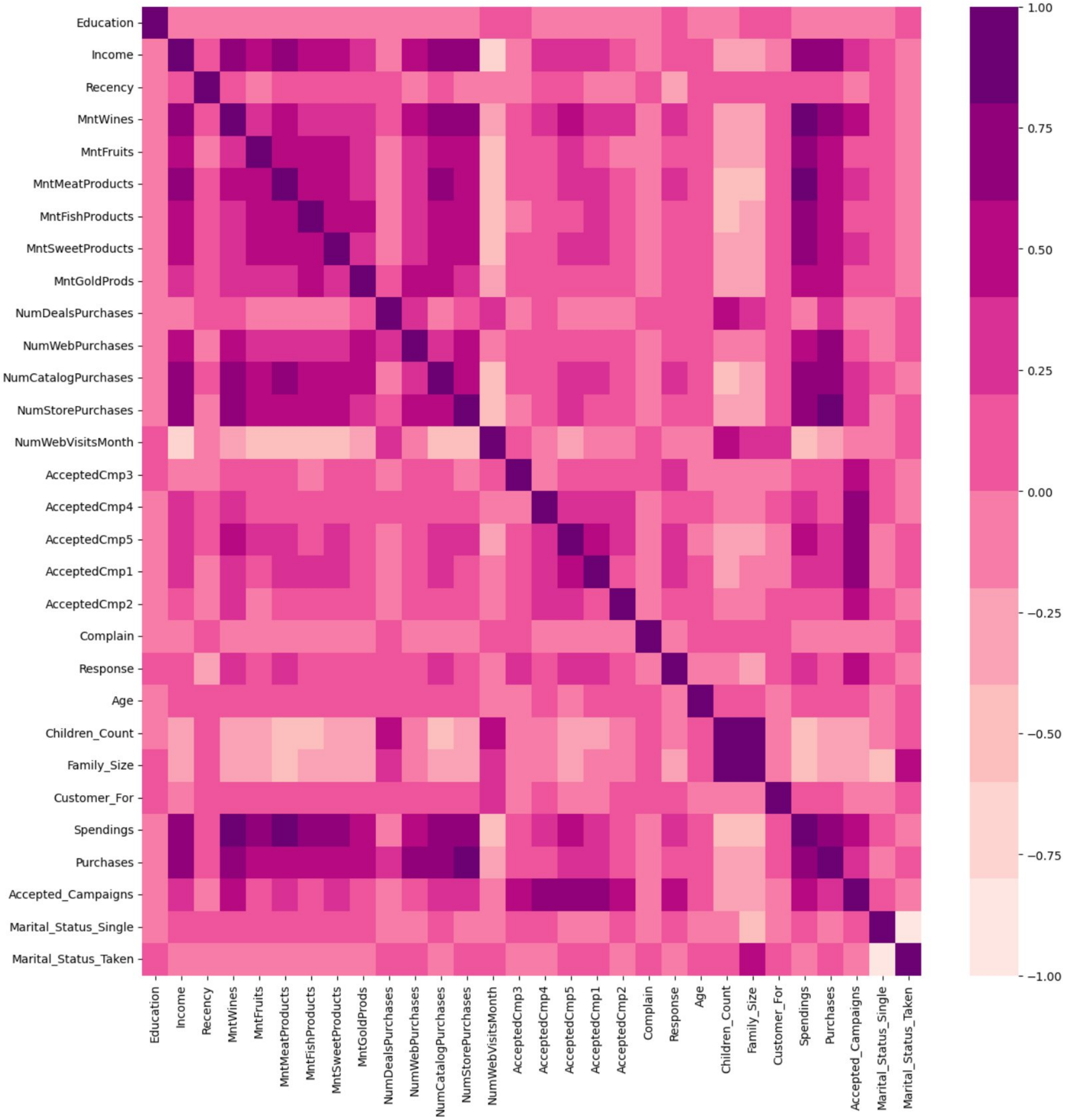
4. Dimensionality Reduction

4.1. Feature Extraction

In [43]:

```
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(data.corr(),
            # cmap="YlGnBu",
            cmap= sns.color_palette("RdPu", 10),
            ax=ax)
```

Out[43]: <AxesSubplot:>



A lot of features have high correlation values. PCA will be used to reduce the dimensions while keeping 95% of the variations

In [44]:
```
#95% variations
pca = PCA(n_components = 0.95)
pca.fit(data)
reduced_data = pd.DataFrame(pca.transform(data))

print("Current number of features= ",len(reduced_data.columns))
```

```
Current number of features=  20
```

By keeping 95% of the variations, the number of features dropped from 32 to 20, but we want to reduce that even more to reduce complexity, so we will compromise more variations.

In [45]:
```
#75% variations
pca = PCA(n_components = 0.75)
pca.fit(data)
reduced_data = pd.DataFrame(pca.transform(data))

print("Current number of features= ",len(reduced_data.columns))
```

```
Current number of features=  10
```

In [88]:
```
reduced_data
```

Out[88]:

	0	1	2	3	4	5	6	7	8	
0	4.605378	-0.755068	-1.917570	2.577038	1.937583	0.384600	0.434224	0.569928	-0.574984	0.2411
1	-2.863008	-1.413863	-0.423587	0.376513	-2.066487	-2.106596	0.043017	0.289241	-0.495433	-0.1321
2	1.781123	0.887343	-1.229014	-0.806737	-0.085843	0.225743	-0.612138	0.499073	-0.820123	1.4476
3	-2.869203	-0.248172	0.138326	-1.621028	-0.020566	-0.484270	-0.110295	0.215018	-1.308722	-0.0144
4	-0.405215	1.478620	-0.469430	-0.465599	-0.170005	0.450082	0.709911	-0.958755	0.527134	-1.0449
...	...	...	...	...	...	...	...	...	...	...
2207	2.250210	2.069555	-0.892137	0.014253	0.995412	-0.447299	0.947423	-0.323680	-0.907672	1.0946
2208	-0.849203	3.396810	2.938072	0.818744	-2.147752	-2.665592	-0.383630	-0.589209	0.577828	-2.2641
2209	2.405278	-2.102802	0.369897	0.971885	-2.355757	2.084351	1.211738	-0.288315	-0.501117	0.3074
2210	1.556511	1.774588	-0.816800	-0.725608	-0.760810	-0.876337	-1.827557	0.252262	0.209832	0.6649
2211	-2.173372	1.354253	1.271357	0.495825	0.721042	-0.455286	-1.116965	0.659663	0.180362	-0.3360

2212 rows × 10 columns

4. Data Clustering

4.1 K-Means Clustering

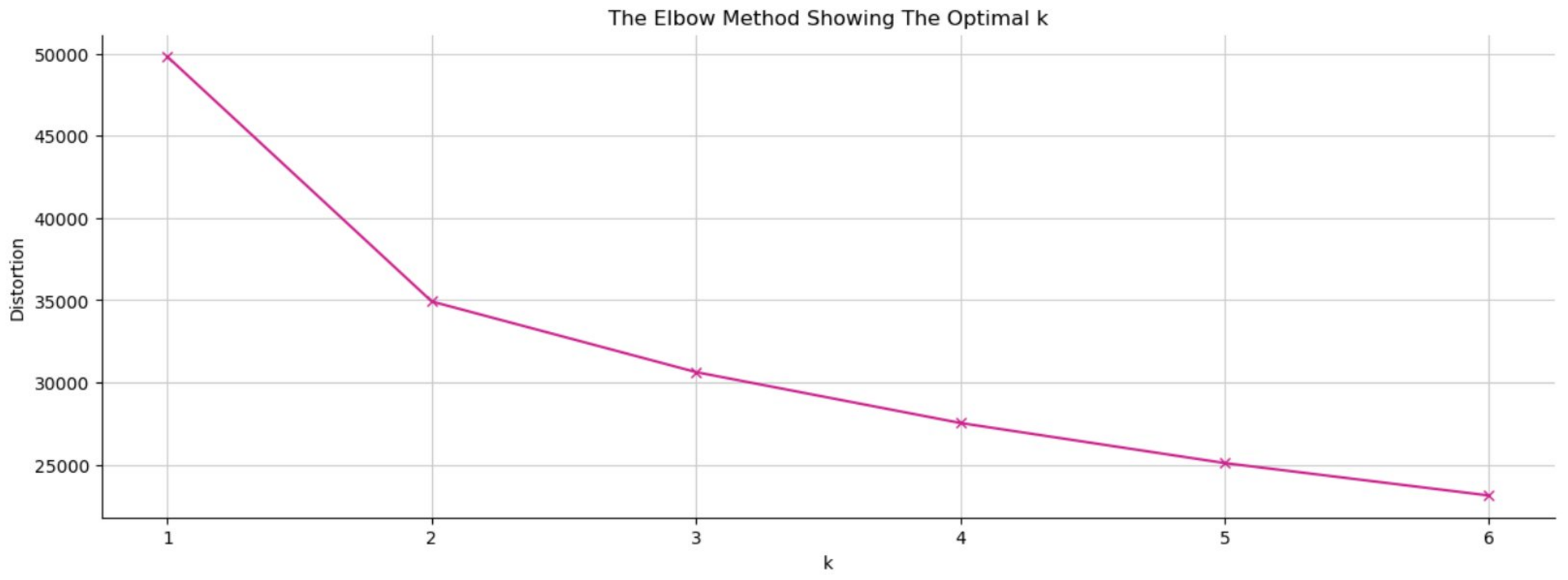
First, we need to have a good sense of how many clusters are in our dataset. To determine this, we will use the elbow method.

In [46]:
```
#calculate distortions for different values of k (number of clusters)
distortions = []
K = range(1,7)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(reduced_data)
    distortions.append(kmeanModel.inertia_)
```

In [47]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [48]:
```
#plot elbow graph
plt.figure(figsize=(15,5))
plt.rcParams.update({'axes.grid': True})
plt.plot(K, distortions, 'bx-', color=palette[-1])
plt.xlabel('k')
```

```
plt.ylabel('Distortion')
plt.title('The Elbow Method Showing The Optimal k')
sns.despine()
plt.show()
```



Using the above plot, we will choose the value of k to be 4.

In [55]:
```
#clustering data and adding the output to the data dataframe
clusters_km = KMeans(n_clusters=4, random_state=202).fit(reduced_data)
data["Personality"]= clusters_km.labels_
data.head()
```

Out[55]:

	Education	Income	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts
0	-0.893586	0.287105	0.310353	0.977660	1.552041	1.690293	2.453472	1.483713
1	-0.893586	-0.260882	-0.380813	-0.872618	-0.637461	-0.718230	-0.651004	-0.634019
2	-0.893586	0.913196	-0.795514	0.357935	0.570540	-0.178542	1.339513	-0.147184
3	-0.893586	-1.176114	-0.795514	-0.872618	-0.561961	-0.655787	-0.504911	-0.585335
4	0.571657	0.294307	1.554453	-0.392257	0.419540	-0.218684	0.152508	-0.001133

5 rows × 31 columns

- **Cluster Distribution**

In [56]:
```
print(data["Personality"].value_counts())
```

```
0    1025
2     548
1     492
3     147
Name: Personality, dtype: int64
```

4.2 Agglomerative (Hierarchical) Clustering

In [131…
```
clusters_hi=AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward').fi
data2=data.copy()
data2['Personality']=clusters_hi.labels_
data2.head()
```

Out[131…

	Education	Income	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts
0	-0.893586	0.287105	0.310353	0.977660	1.552041	1.690293	2.453472	1.483713
1	-0.893586	-0.260882	-0.380813	-0.872618	-0.637461	-0.718230	-0.651004	-0.634019
2	-0.893586	0.913196	-0.795514	0.357935	0.570540	-0.178542	1.339513	-0.147184
3	-0.893586	-1.176114	-0.795514	-0.872618	-0.561961	-0.655787	-0.504911	-0.585335
4	0.571657	0.294307	1.554453	-0.392257	0.419540	-0.218684	0.152508	-0.001133

5 rows × 31 columns

In [132…

```
print(data2["Personality"].value_counts())
```

```
0    780
1    739
3    462
2    231
Name: Personality, dtype: int64
```

4.3 DBSCAN

In [115…

```
clusters_db=DBSCAN(min_samples=5,eps=2.5).fit(reduced_data)
data3=data.copy()
data3['Personality']=clusters_db.labels_
data3.head()
```

Out[115…

	Education	Income	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts
0	-0.893586	0.287105	0.310353	0.977660	1.552041	1.690293	2.453472	1.483713
1	-0.893586	-0.260882	-0.380813	-0.872618	-0.637461	-0.718230	-0.651004	-0.634019
2	-0.893586	0.913196	-0.795514	0.357935	0.570540	-0.178542	1.339513	-0.147184
3	-0.893586	-1.176114	-0.795514	-0.872618	-0.561961	-0.655787	-0.504911	-0.585335
4	0.571657	0.294307	1.554453	-0.392257	0.419540	-0.218684	0.152508	-0.001133

5 rows × 31 columns

In [116…

```
print(data3["Personality"].value_counts())
```

```
 1    1239
 0     641
-1     210
 4      58
 3      23
 2      18
 5      15
 7       5
 6       3
Name: Personality, dtype: int64
```

According to given parameters 210 data points are noise point

Titanic Survival Prediction

> Name - Ajit SIngh
> Roll No. - 19058570034

Practical 7

Importing Libraries

In [40]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Dataset

In [41]:
```
df_train = pd.read_csv ('train.csv')
df_test = pd.read_csv('test.csv')
```

In [42]:
```
df_train.head()
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [43]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
```

```
 3   Name          891 non-null    object
 4   Sex           891 non-null    object
 5   Age           714 non-null    float64
 6   SibSp         891 non-null    int64
 7   Parch         891 non-null    int64
 8   Ticket        891 non-null    object
 9   Fare          891 non-null    float64
 10  Cabin         204 non-null    object
 11  Embarked      889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

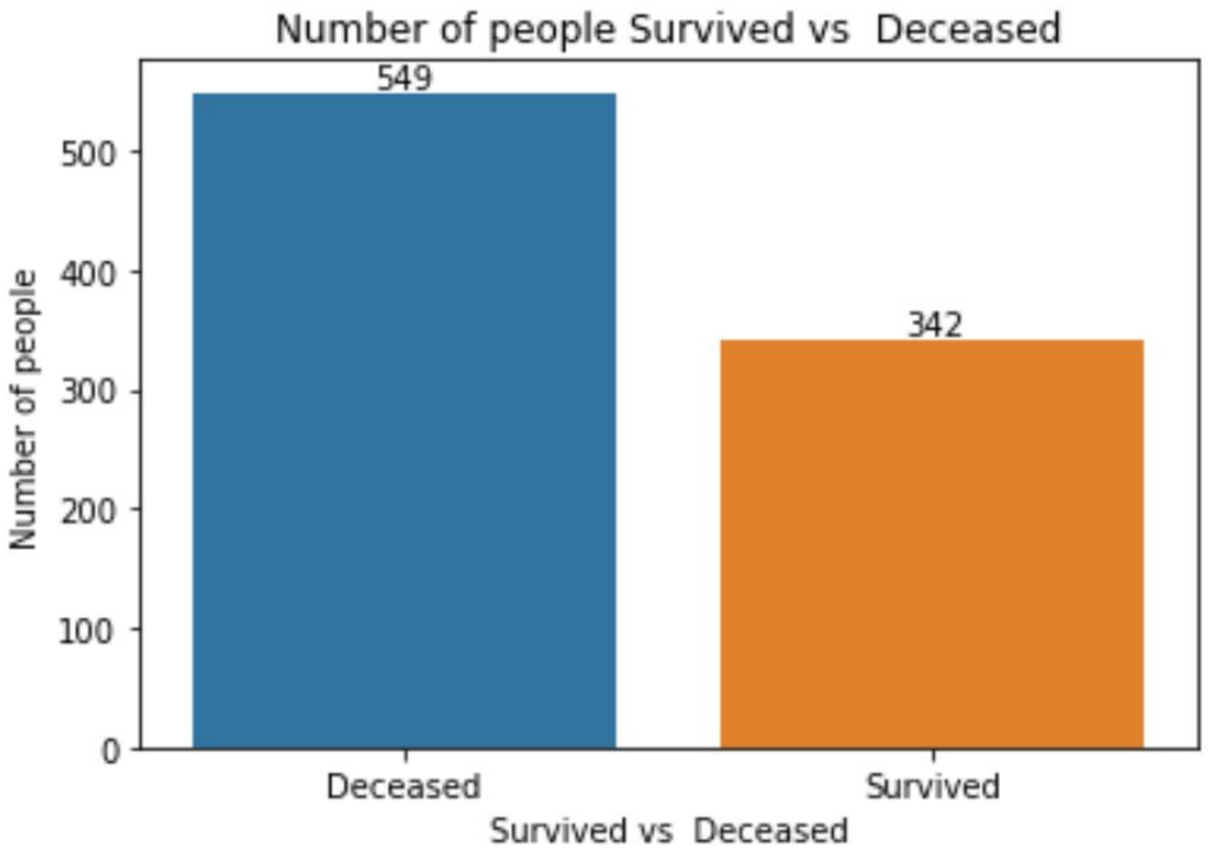
In [44]:
```
## Checking for  the number of rows and columns  in the dataset
print(f"Number of rows :{df_train.shape[0]} \nNumber of columns:{df_train.shape[1]}")
```

```
Number of rows :891
Number of columns:12
```

Data Exploration

Survived

In [45]:
```
ax = sns.countplot(data=df_train,x ='Survived');
ax.bar_label(ax.containers[0])
plt.title("Number of people Survived vs  Deceased")
plt.xlabel("Survived vs  Deceased")
plt.ylabel("Number of people")
plt.xticks(ticks=[0,1],labels=['Deceased','Survived'])
plt.show();
```



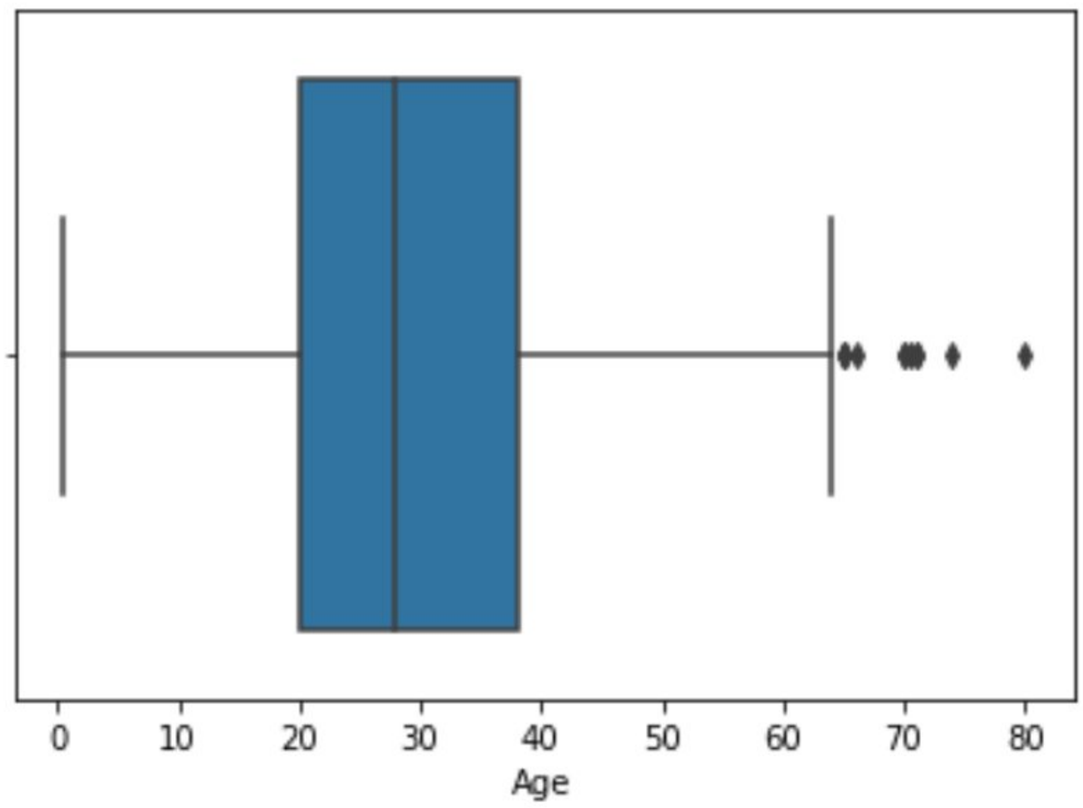
Age

In [46]:
```
df_train.Age.isna().sum()
```

Out[46]:
```
177
```

In [47]:
```
sns.boxplot(x=df_train['Age'])
```

Out[47]:
```
<AxesSubplot:xlabel='Age'>
```



There are outliers but those are valid age

In [48]:
```
### Filling NAN Values With Mean
df_train["Age"]= df_train["Age"].fillna(df_train["Age"].mean())
### Plotting People On Different Age Groups
df_train.Age = df_train.Age.astype(int)
```

In [49]:
```
df_train['Age'].isna().sum()
```

Out[49]:
```
0
```

In [50]:
```
#### Creating A Copy of The Dataset
temp = df_train.copy()
```

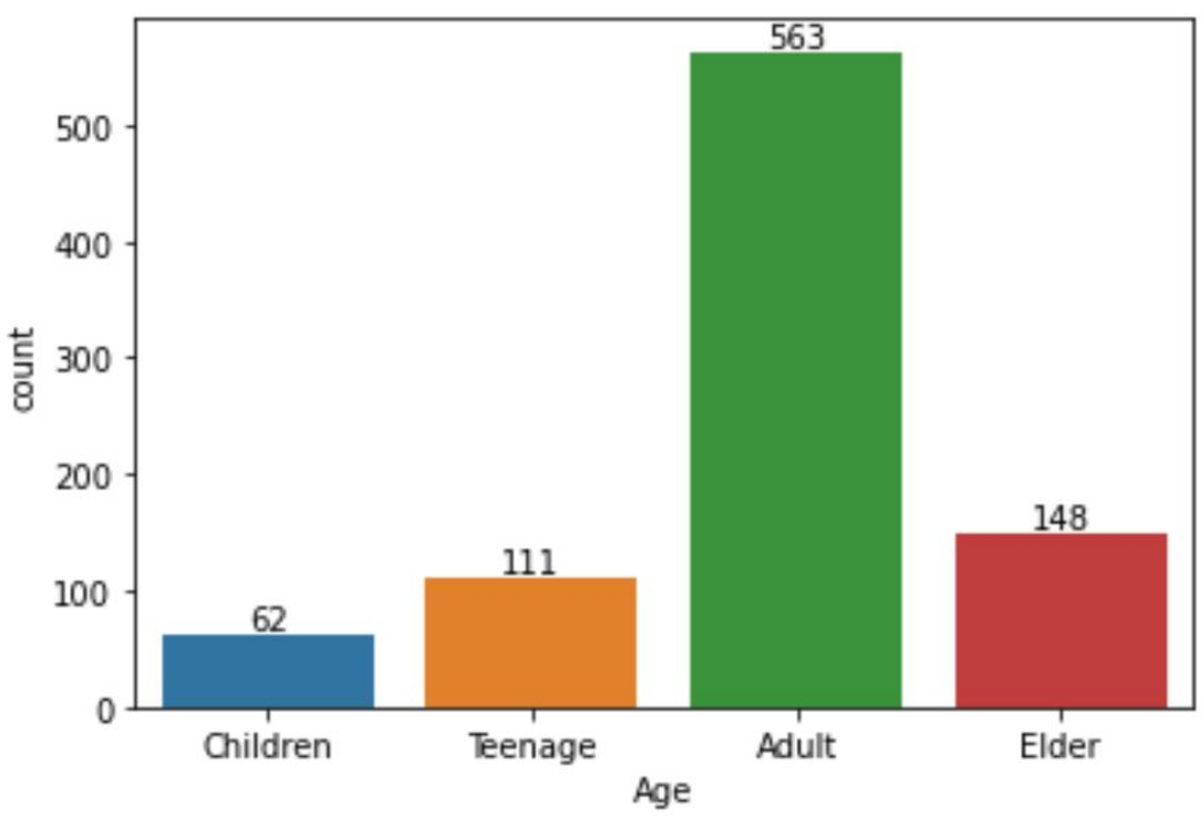
In [51]:
```
temp['Age'] = pd.cut(temp['Age'], bins=[0,12,20,40,120], labels=['Children','Teenage','Adu
```

In [53]:
```
temp['Age'].value_counts()
```

Out[53]:
```
Adult       563
Elder       148
Teenage     111
Children     62
Name: Age, dtype: int64
```

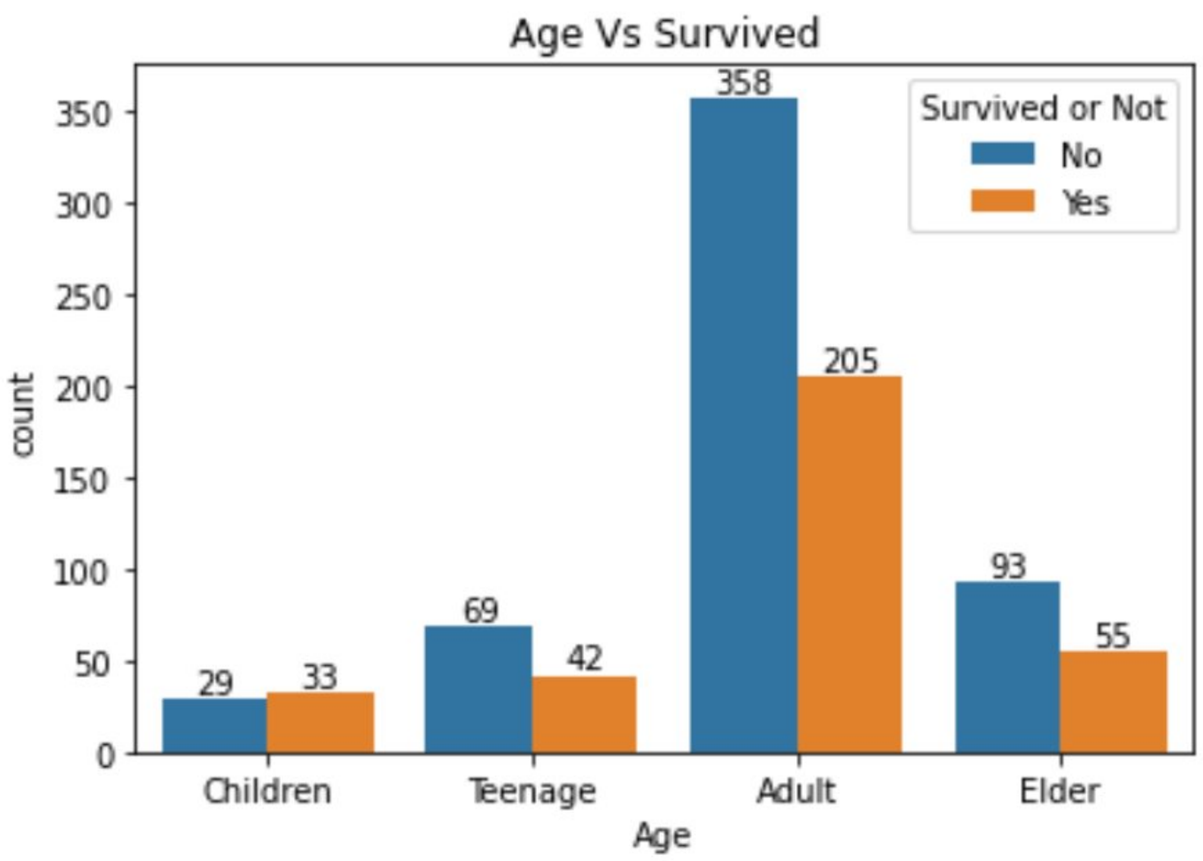
In [54]:
```
ax = sns.countplot(data=temp,x = 'Age')
ax.bar_label(ax.containers[0])
```

Out[54]:
```
[Text(0, 0, '62'), Text(0, 0, '111'), Text(0, 0, '563'), Text(0, 0, '148')]
```



In [55]:

```
ax = sns.countplot(data=temp,x = 'Age',hue='Survived')
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Age Vs Survived')
plt.show();
```



Fare

In [14]:

```
temp['Fare'].mean()
```

Out[14]:

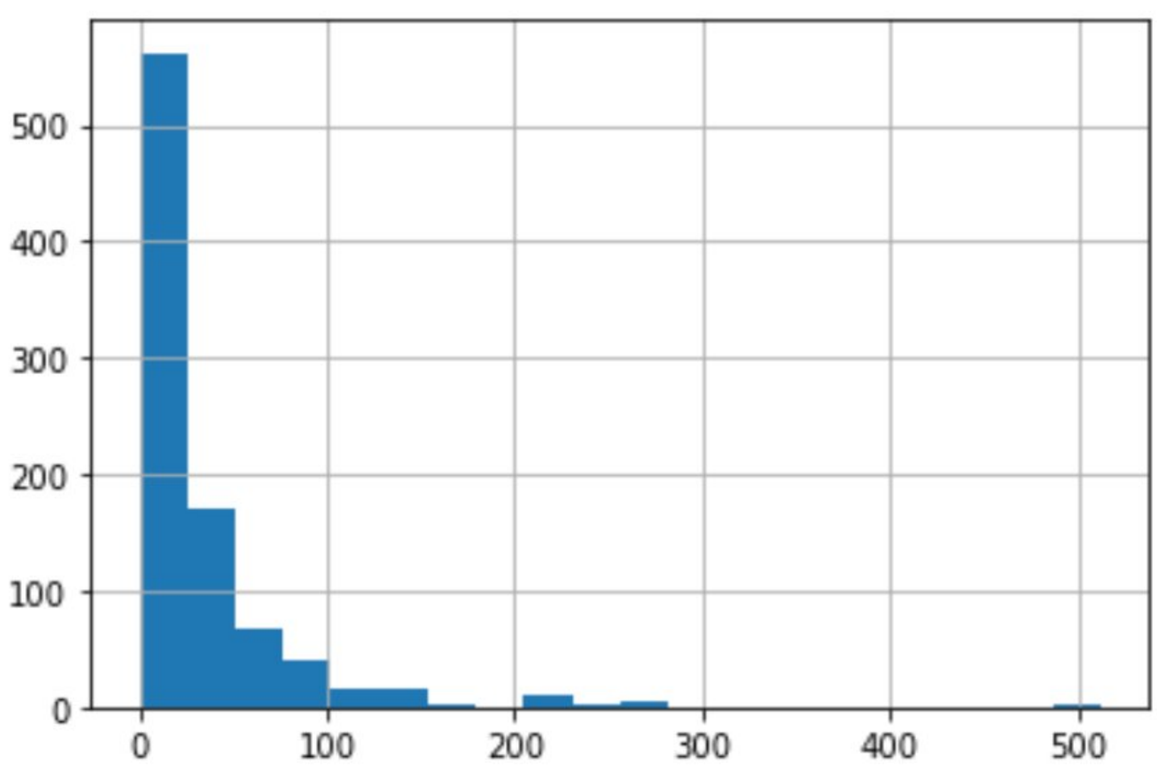
```
32.2042079685746
```

In [56]:

```
temp['Fare'].hist(bins=20)
```

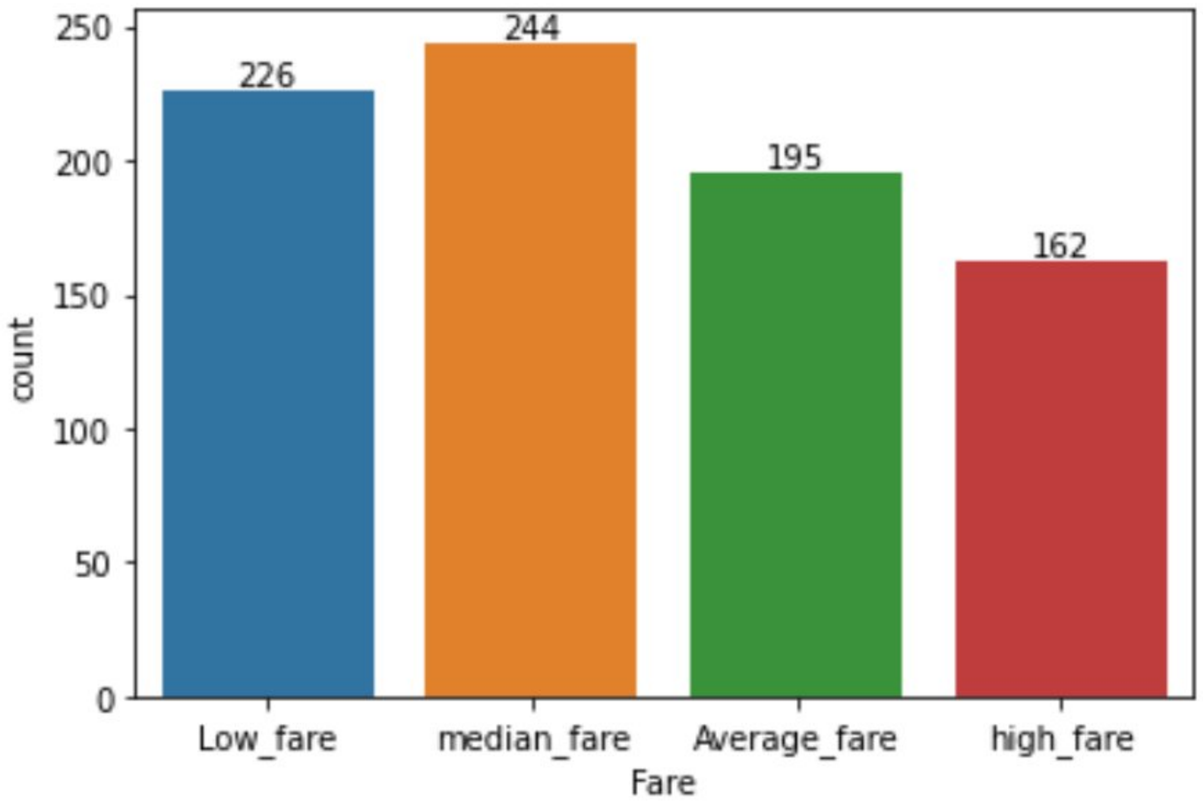
Out[56]:

```
<AxesSubplot:>
```



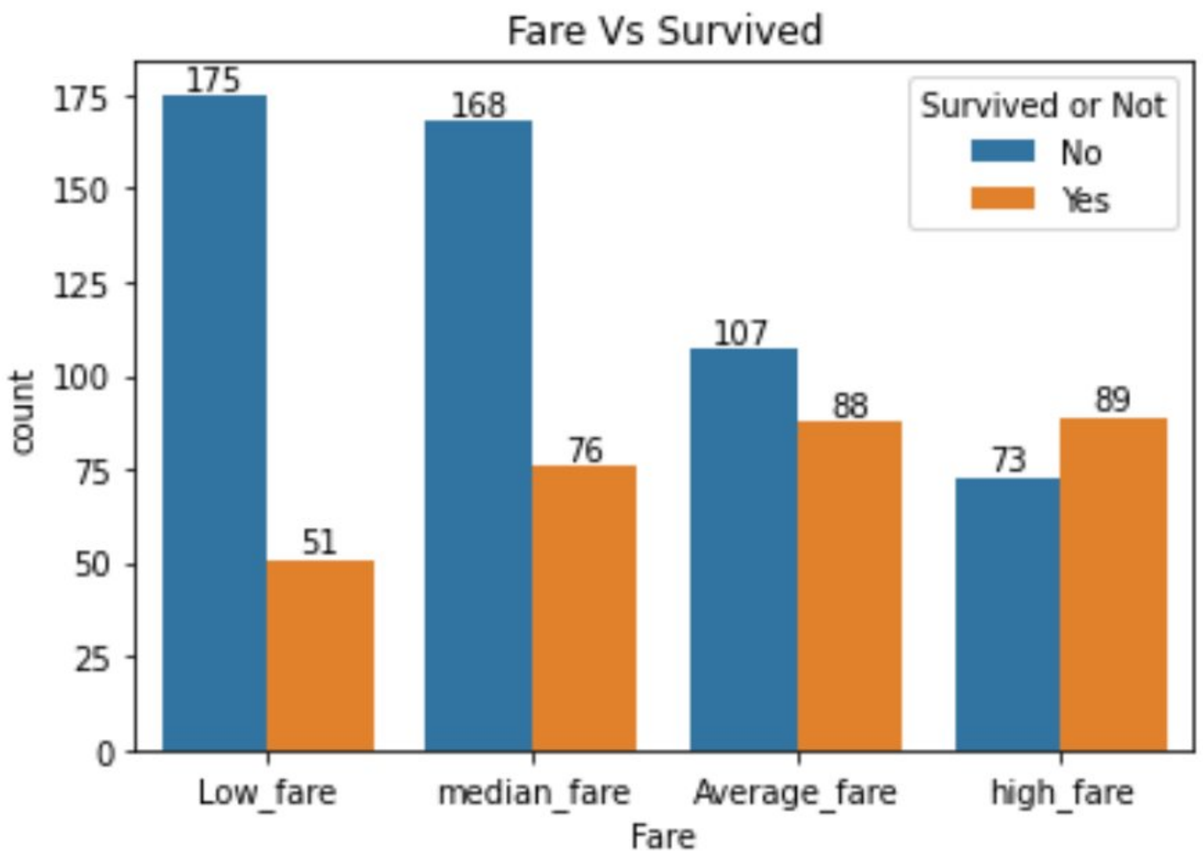
In [57]:

```
temp['Fare'] = pd.cut(temp['Fare'], bins=[0,8,16,32,110], labels=['Low_fare','median_fare'
ax = sns.countplot(data=temp, x='Fare')
ax.bar_label(ax.containers[0]);
```



In [58]:

```
ax = sns.countplot(data=temp, x='Fare',hue='Survived')
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Fare Vs Survived')
plt.show();
```



Cabin

In [59]:
```
nullCabin = temp.Cabin.isna().sum()
nullCabin
```

Out[59]:
```
687
```

In [60]:
```
str(round(nullCabin/(len(temp))*100,2))+"% Null Values"
```

Out[60]:
```
'77.1% Null Values'
```

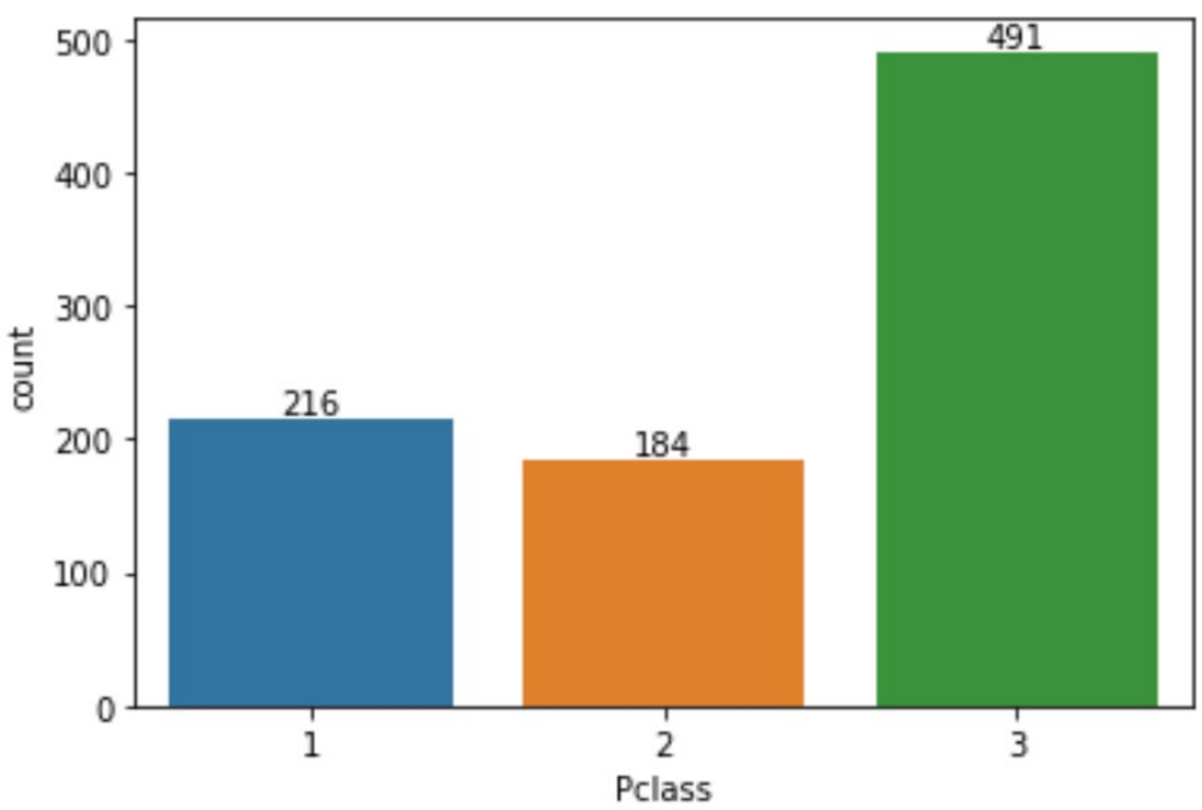
There Are So Many Null Values So Its Better To Drop It.

Pclass : Passenger Class

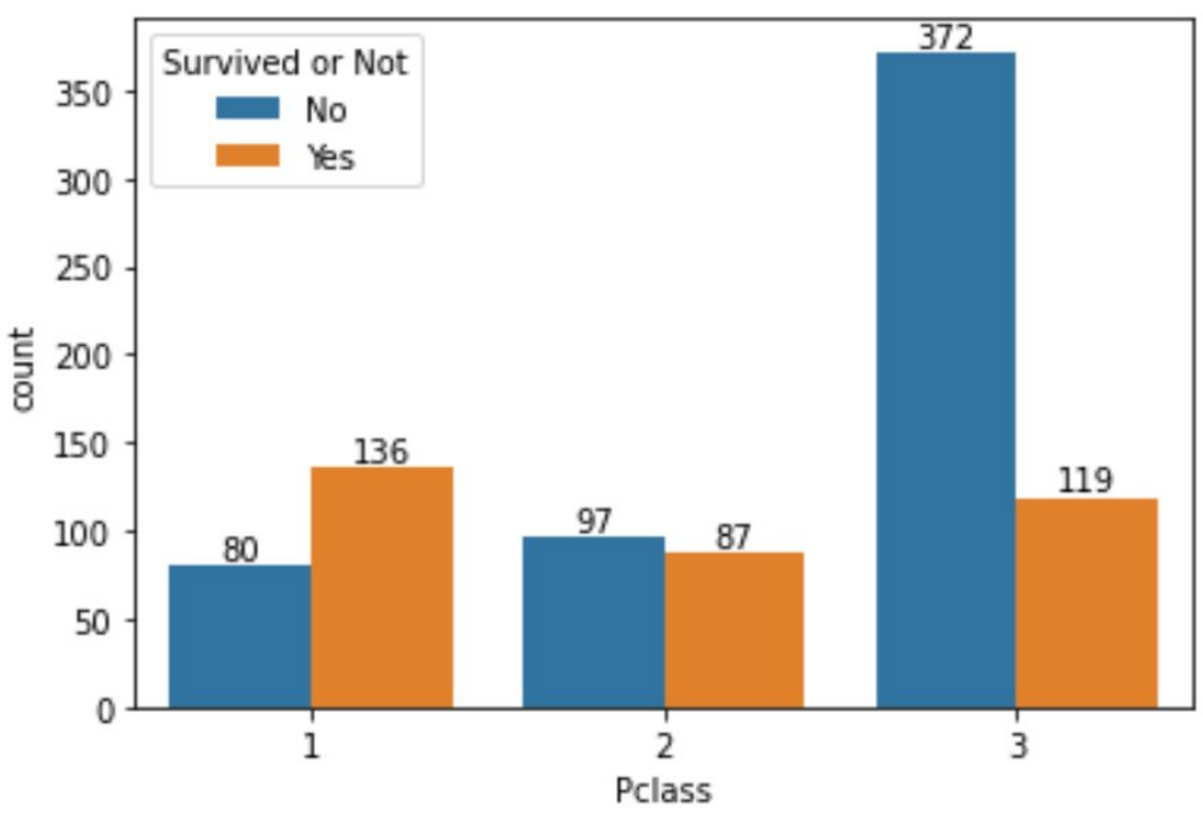
In [61]:
```
temp.Pclass.unique()
```

Out[61]:
```
array([3, 1, 2], dtype=int64)
```

In [62]:
```
ax = sns.countplot(data=temp,x='Pclass');
ax.bar_label(ax.containers[0]);
```



In [63]:
```
ax = sns.countplot(data=temp,x='Pclass',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper left', labels=['No', 'Yes']);
```

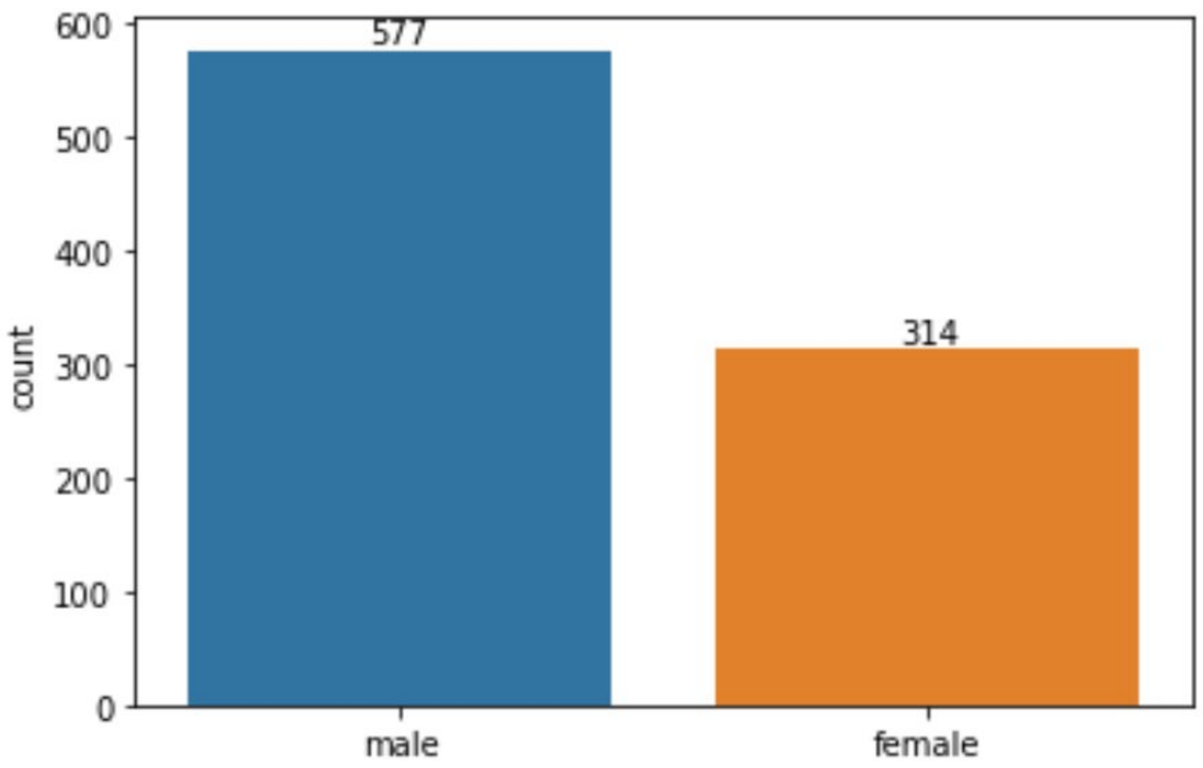


Sex

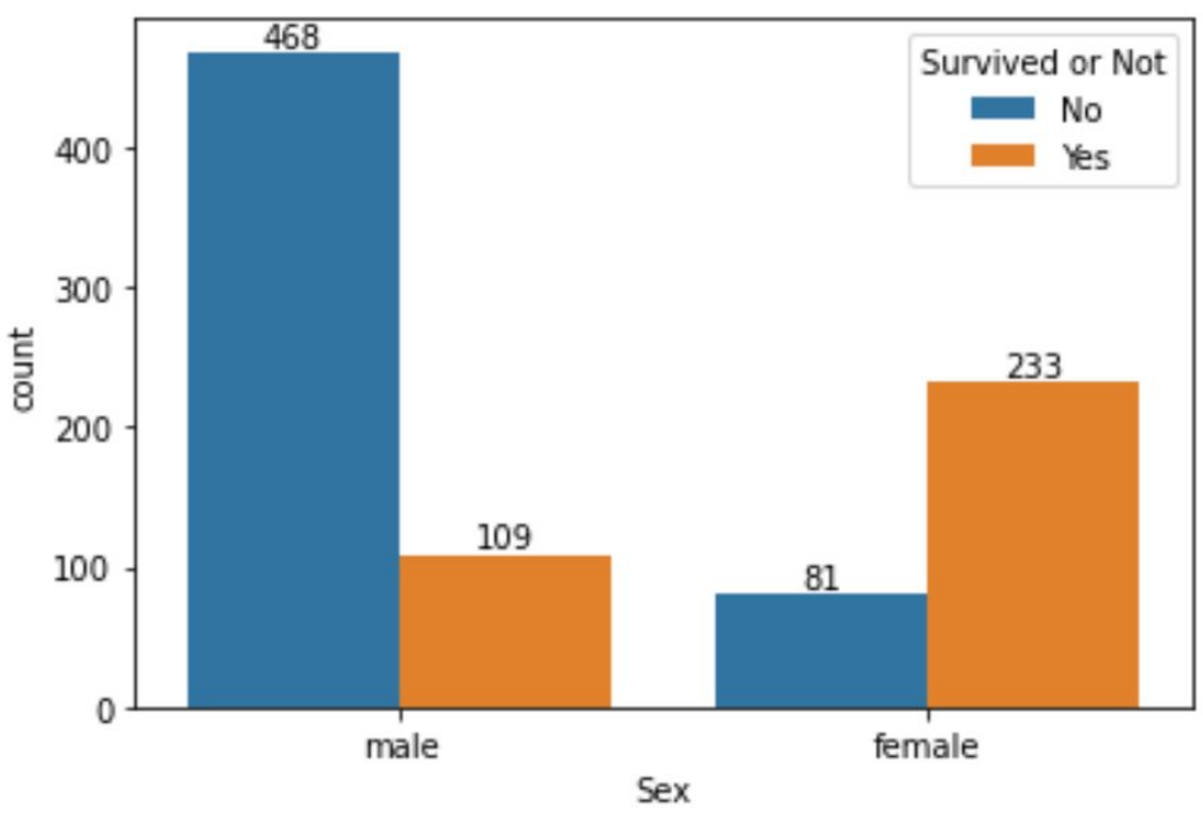
In [64]:
```
temp.Sex.unique()
```

Out[64]:
```
array(['male', 'female'], dtype=object)
```

In [65]:
```
ax = sns.countplot(data=temp,x='Sex');
ax.bar_label(ax.containers[0]);
```



In [66]:
```
### Sex vs Survived
ax = sns.countplot(data=temp,x='Sex',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
```

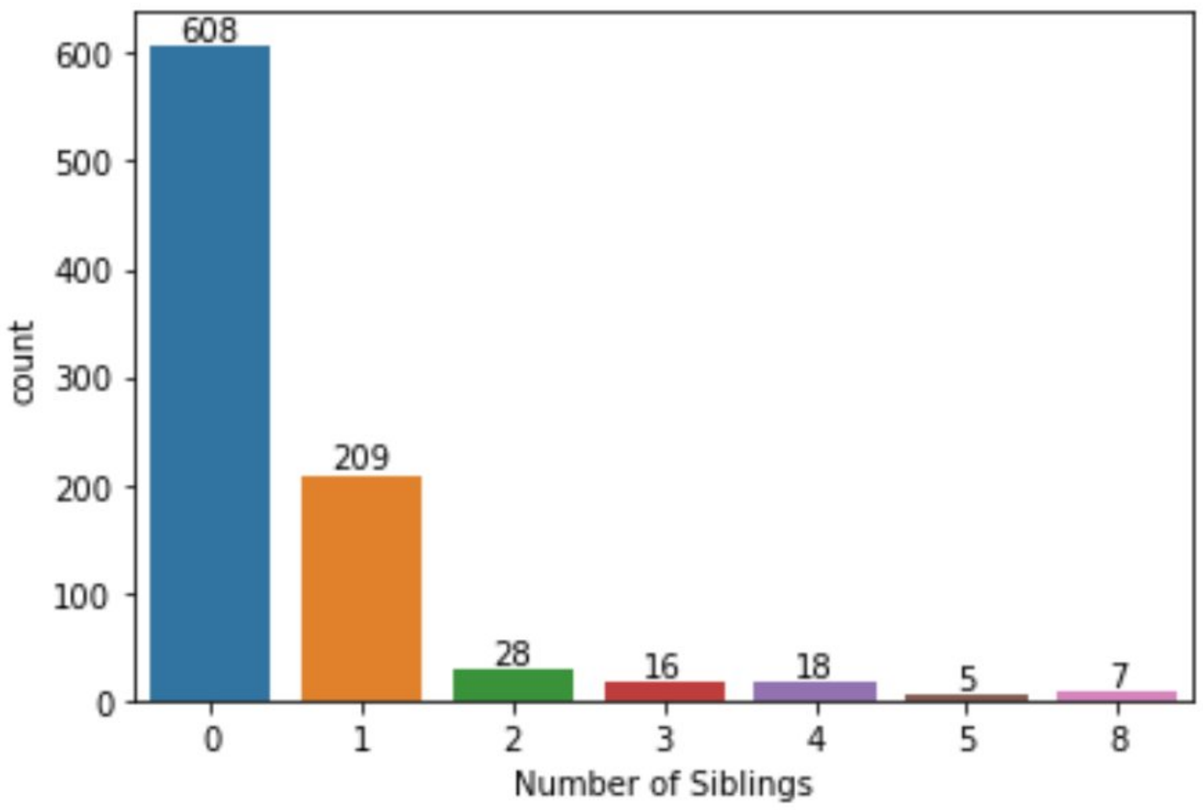


SibSp : # of siblings / spouses aboard the Titanic

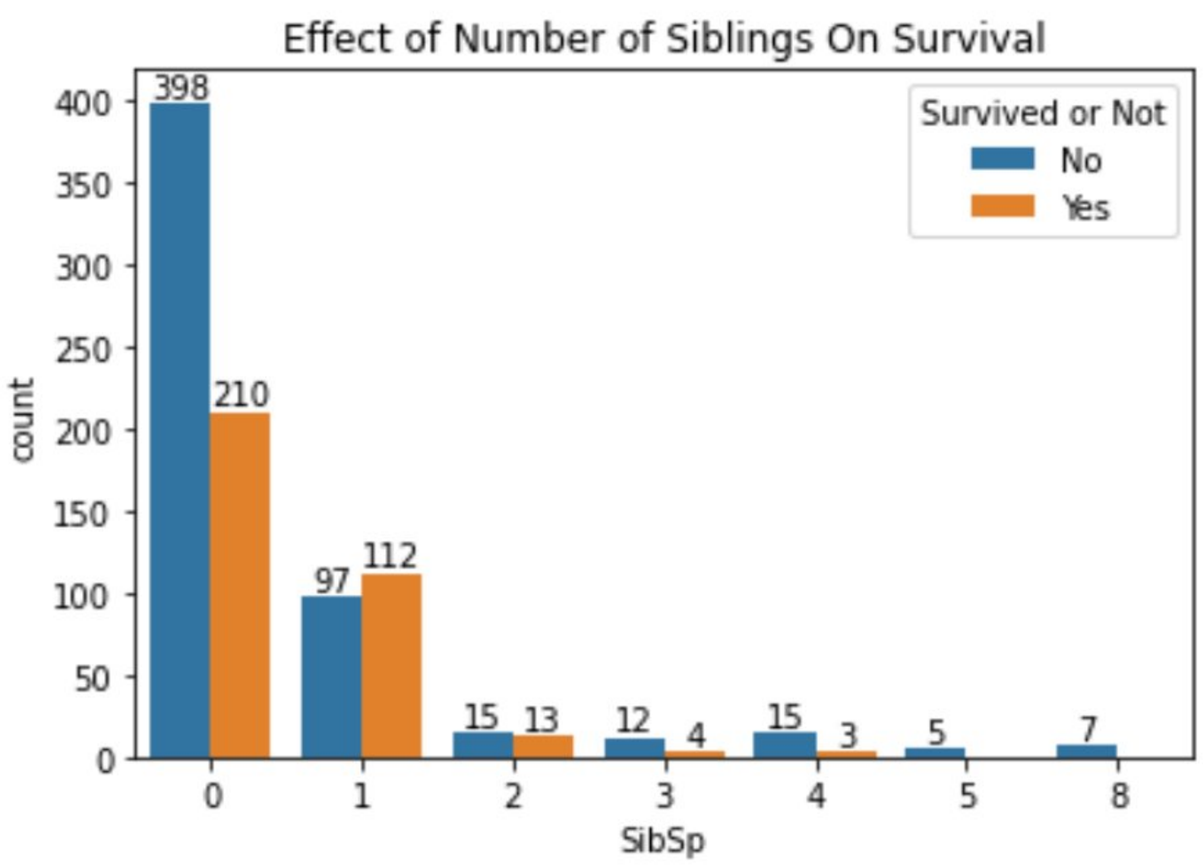
In [67]:
```
temp.SibSp.unique()
```

Out[67]:
```
array([1, 0, 3, 4, 2, 5, 8], dtype=int64)
```

In [68]:
```
ax = sns.countplot(data=temp,x='SibSp');
ax.bar_label(ax.containers[0]);
ax.set_xlabel('Number of Siblings');
```



In [69]:
```
### Number of Siblings vs Survived
ax = sns.countplot(data=temp,x='SibSp',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Effect of Number of Siblings On Survival')
plt.show();
```

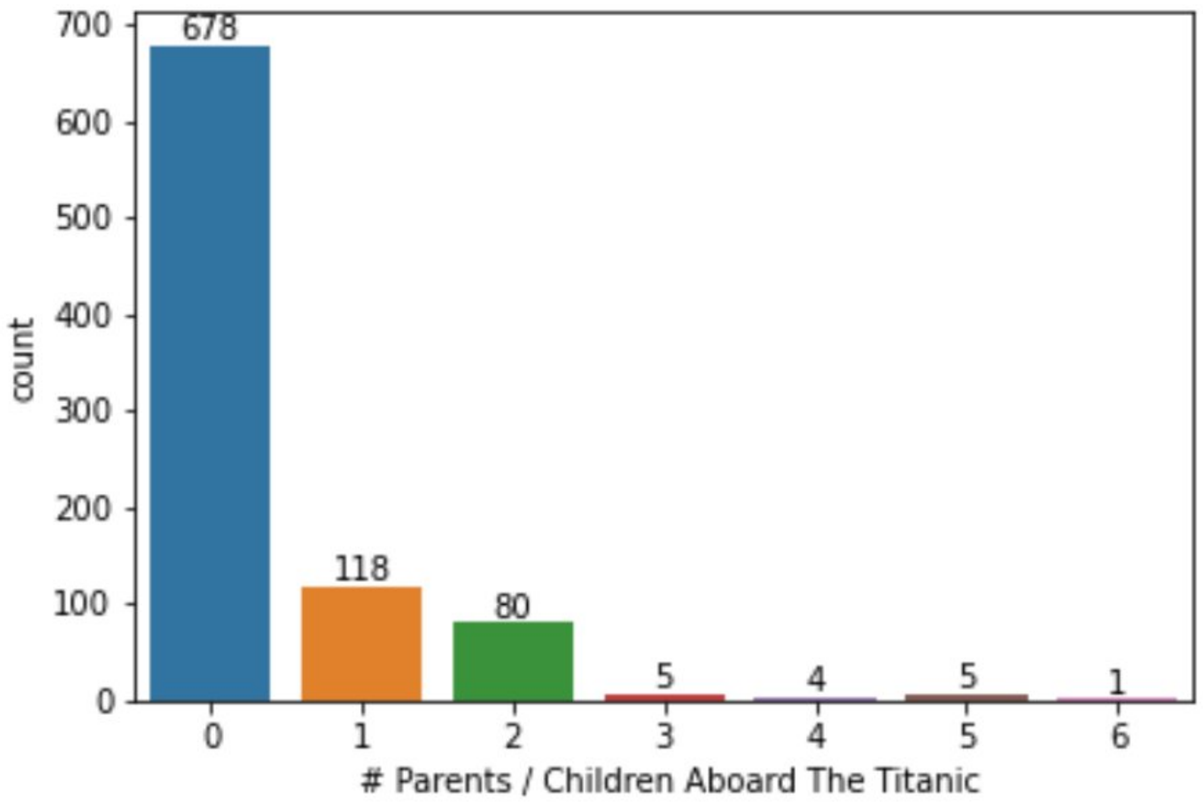


Parch: # of parents / children aboard the titanic

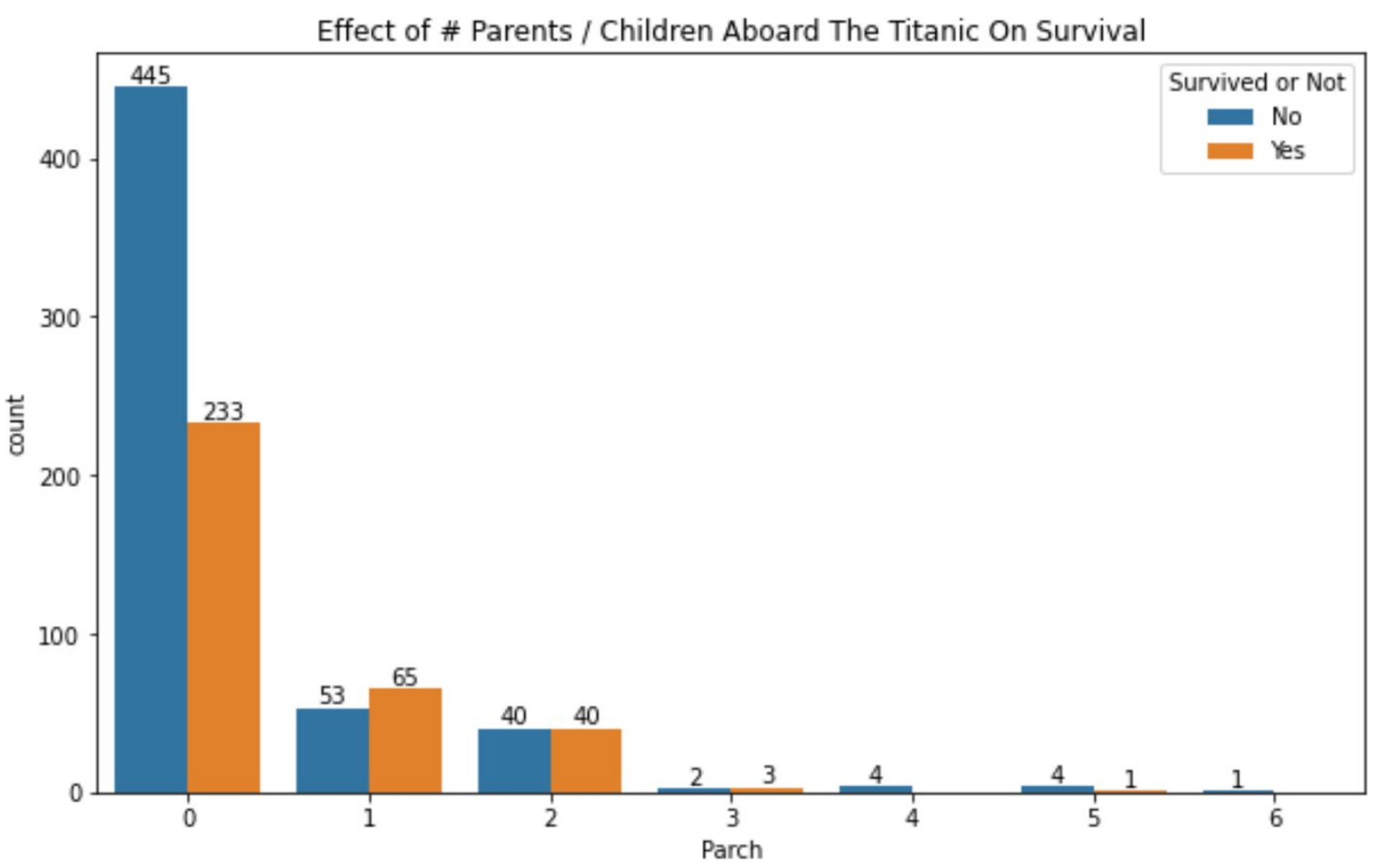
In [70]:
```
temp.Parch.unique()
```

Out[70]:
```
array([0, 1, 2, 5, 3, 4, 6], dtype=int64)
```

In [71]:
```
ax = sns.countplot(data=temp,x='Parch');
ax.bar_label(ax.containers[0]);
ax.set_xlabel(' # Parents / Children Aboard The Titanic');
```

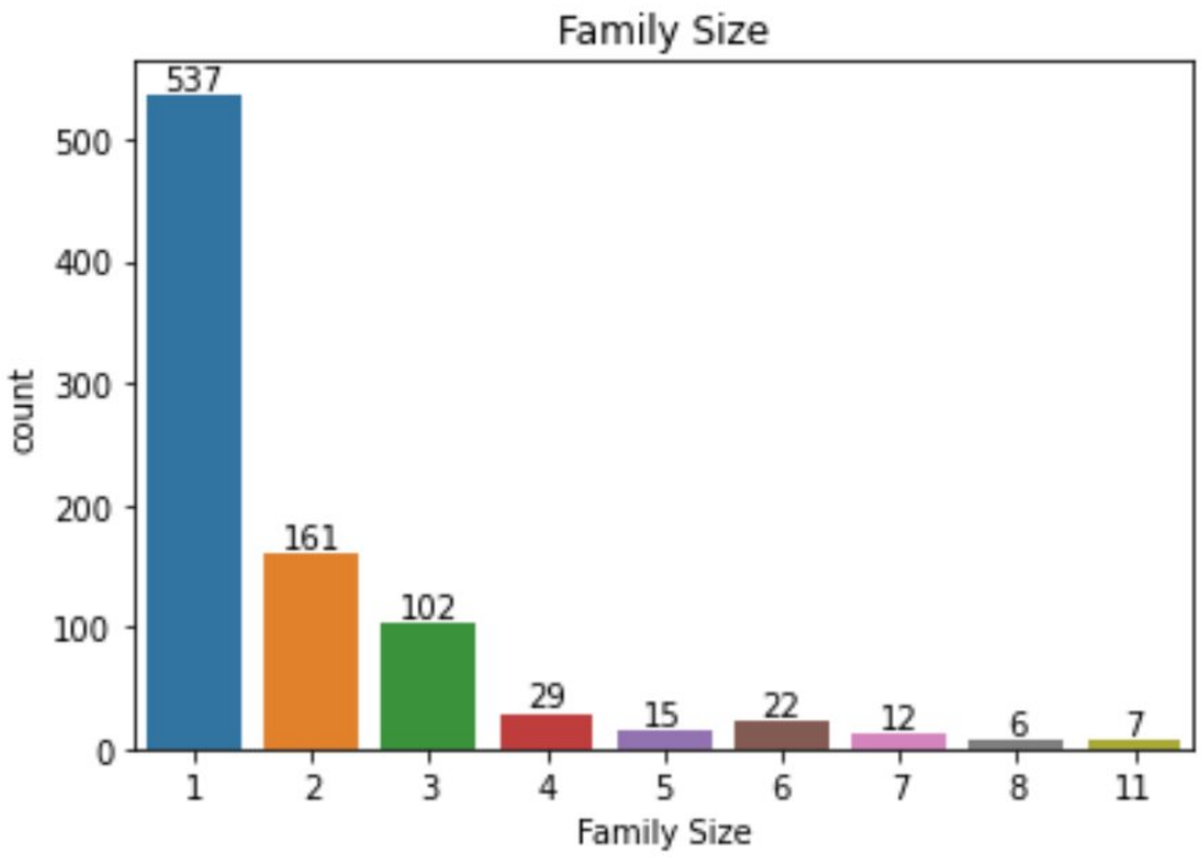


In [72]:
```
### Parch vs Survived
plt.figure(figsize=(10,6))
ax = sns.countplot(data=temp,x='Parch',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Effect of # Parents / Children Aboard The Titanic On Survival')
plt.show();
```

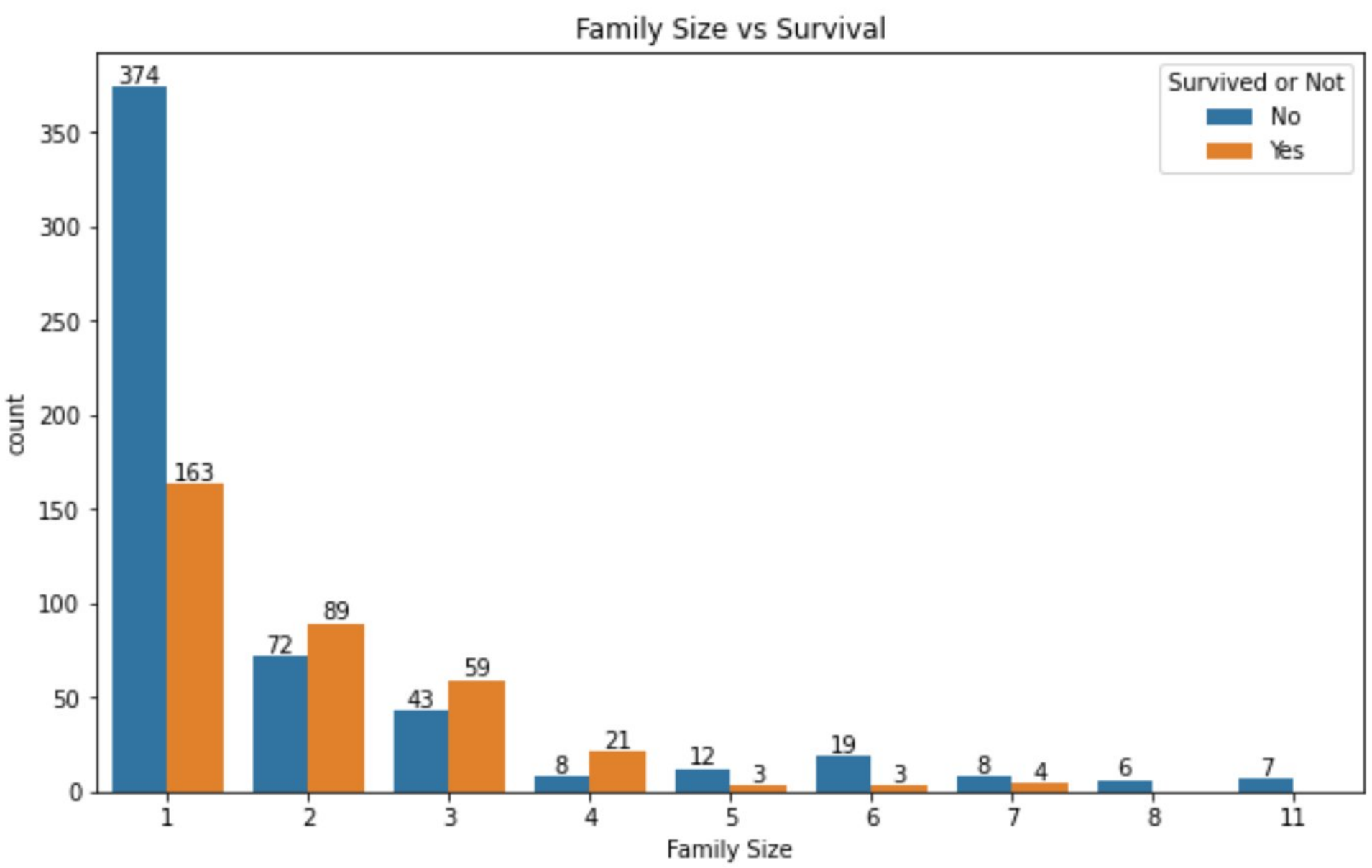


In [73]:
```
temp['Family Size'] = temp['SibSp']+temp['Parch'] + 1
```

In [74]:
```
ax = sns.countplot(data=temp, x='Family Size')
ax.bar_label(ax.containers[0])
ax.set_title('Family Size');
```



In [75]:
```
### Parch vs Survived
plt.figure(figsize=(10,6))
ax = sns.countplot(data=temp,x='Family Size',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Family Size vs Survival')
plt.show();
```

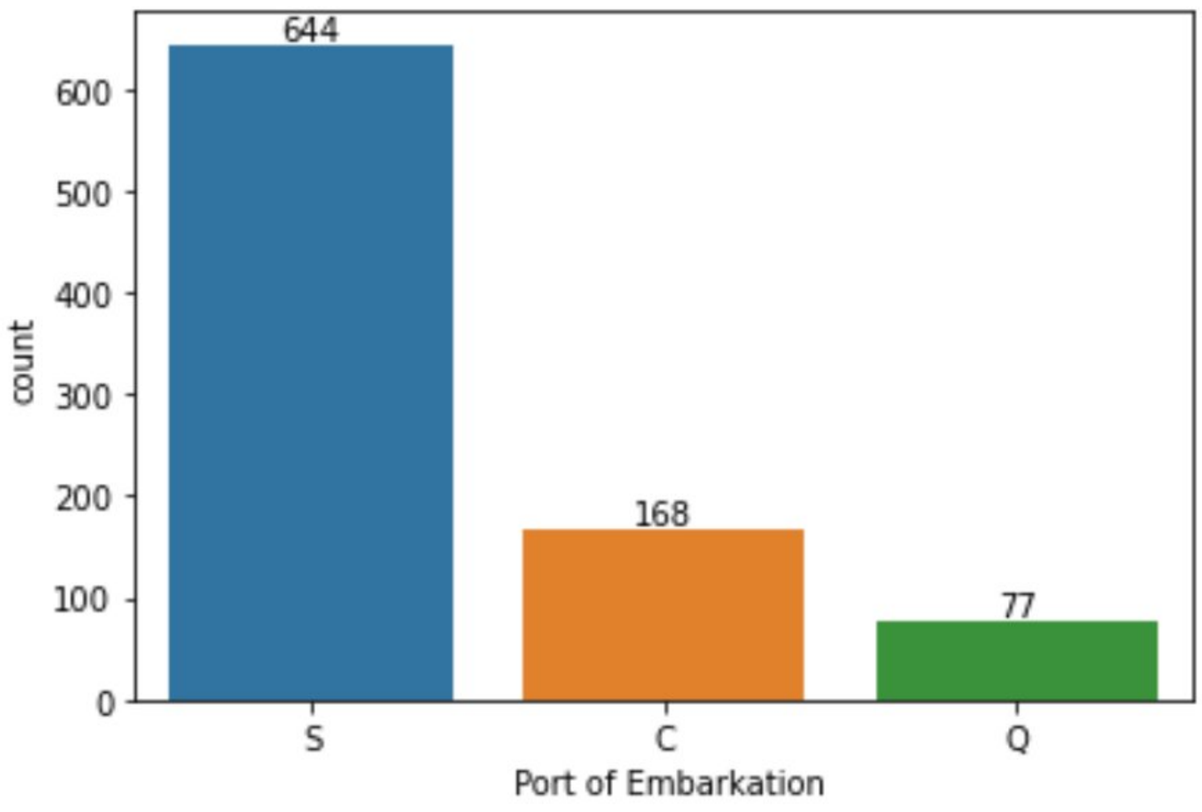


Embarked

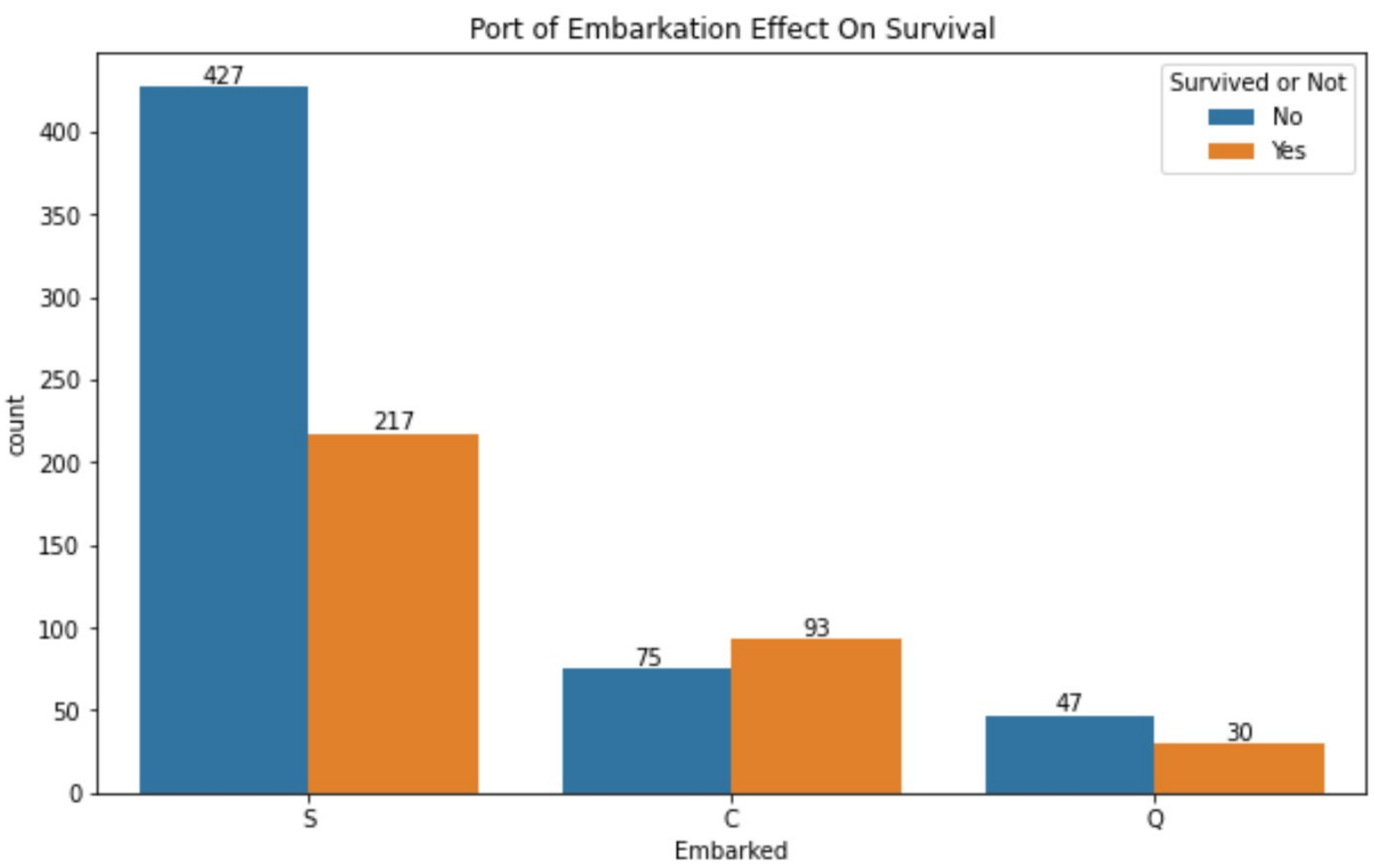
In [76]:
```
temp.Embarked.unique()
```

Out[76]:
```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [77]:
```
ax = sns.countplot(data=temp,x='Embarked');
ax.bar_label(ax.containers[0]);
ax.set_xlabel(' Port of Embarkation');
```



In [78]:
```
### Port of Embarkation vs Survived
plt.figure(figsize=(10,6))
ax = sns.countplot(data=temp,x='Embarked',hue='Survived');
ax.bar_label(ax.containers[0]);
ax.bar_label(ax.containers[1]);
plt.legend(title='Survived or Not', loc='upper right', labels=['No', 'Yes']);
plt.title('Port of Embarkation Effect On Survival')
plt.show();
```



Analysis Results

- Based On The Dataset Only 40% People Were Able To Survived The Disaster.
- Mostly Childrens Are Being Rescued.
- People Above Age 20 had a chance of 35% of Being Survived From The Disaster.
- People Paid High Fare and Class Means VIPs Are Given Priority For Rescued.
- Out of Male and Females, Almost 75% Femals Survived The Disaster.
- People Traveling Alone or With A Smaller Family Size Upto 2 children had high chances of survival.
- Family Size Under 5 Had Higher Chance of Survival On Titanic Disaster.
- People Traveling Alone had approx 43% Chances of Survival.
- Family with Size 5+ Had Lesser Chance of Complete Survival On Titanic Disaster.

Feature Engineering

In [79]:
```
def clean_data(data):
    #### Let's Start By Dropping `PassengerID` `Name` `Ticket` `Cabin`
    data = data.drop(['PassengerId','Name','Ticket','Cabin'],axis=1)

    #### Converting Age Into Four Different Classes 'Children', 'Teenage','Adult', 'old'
    data['Age'] = pd.cut(data['Age'], bins=[0,12,20,40,120], labels=['Children','Teenage',

    #### Converting Fare Into Four Distinct Categories 'Low_fare','median_fare','Average_
    data['Fare'] = pd.cut(data['Fare'], bins=[0,7.91,14.45,31,120], labels=['Low_fare','me

    #### Getting OneHotEncoding For Categorical Columns ['Age','Fare','Sex','Embarked']
    data = pd.get_dummies(data, columns = ["Sex","Age","Embarked","Fare"])

    return data
```

In [80]:
```
df_train = clean_data(df_train)
```

Declaring Dependent and Independent features

In [82]:
```
X = df_train.drop('Survived',axis=1)
y = df_train['Survived']
```

In [83]:
```
X.head()
```

Out[83]:

	Pclass	SibSp	Parch	Sex_female	Sex_male	Age_Children	Age_Teenage	Age_Adult	Age_Elder	Embarked_C	Em
0	3	1	0	0	1	0	0	1	0	0	
1	1	1	0	1	0	0	0	1	0	1	
2	3	0	0	1	0	0	0	1	0	0	
3	1	1	0	1	0	0	0	1	0	0	
4	3	0	0	0	1	0	0	1	0	0	

In [41]:
```
y.head()
```

Out[41]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

Model Training

Models

1. Naive Bayes
2. SVM
3. KNN
4. Logistic Regression
5. Random Forest

In [86]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=25)
```

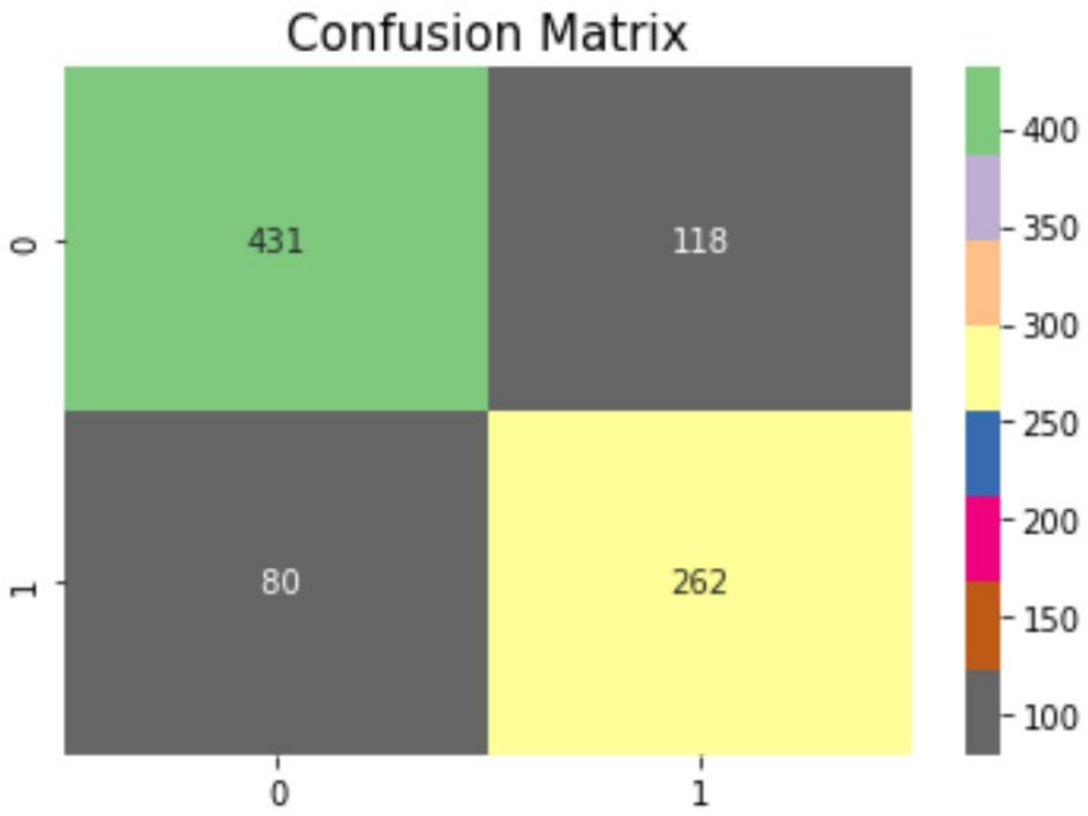
In [87]:
```
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
from sklearn.model_selection import KFold, cross_val_score, cross_val_predict ## Cross Va
```

Naive Bayes

In [97]:
```
################## Naive Bayes ##########################
from sklearn.naive_bayes import GaussianNB
model= GaussianNB()
model.fit(X_train,y_train)
prediction_nb=model.predict(X_test)

print('--------GaussianNB Naive Bayes -------')
print('The accuracy Gaussian Naive Bayes Classifier is',round(accuracy_score(y_test,predic

kfold = KFold(n_splits=8,shuffle=True,random_state=25) # split the data into 10 equal part
```

```
result_nb=cross_val_score(model,X,y,cv=10,scoring='accuracy')

print('The cross validated score for Gaussian Naive Bayes classifier is:',round(result_nb.

y_pred = cross_val_predict(model,X,y,cv=10)
sns.heatmap(confusion_matrix(y,y_pred),annot=True,fmt='3.0f',cmap="Accent_r")
plt.title('Confusion Matrix', y=1, size=15);
```

```
--------GaussianNB Naive Bayes -------
The accuracy Gaussian Naive Bayes Classifier is 78.73
The cross validated score for Gaussian Naive Bayes classifier is: 77.78
```

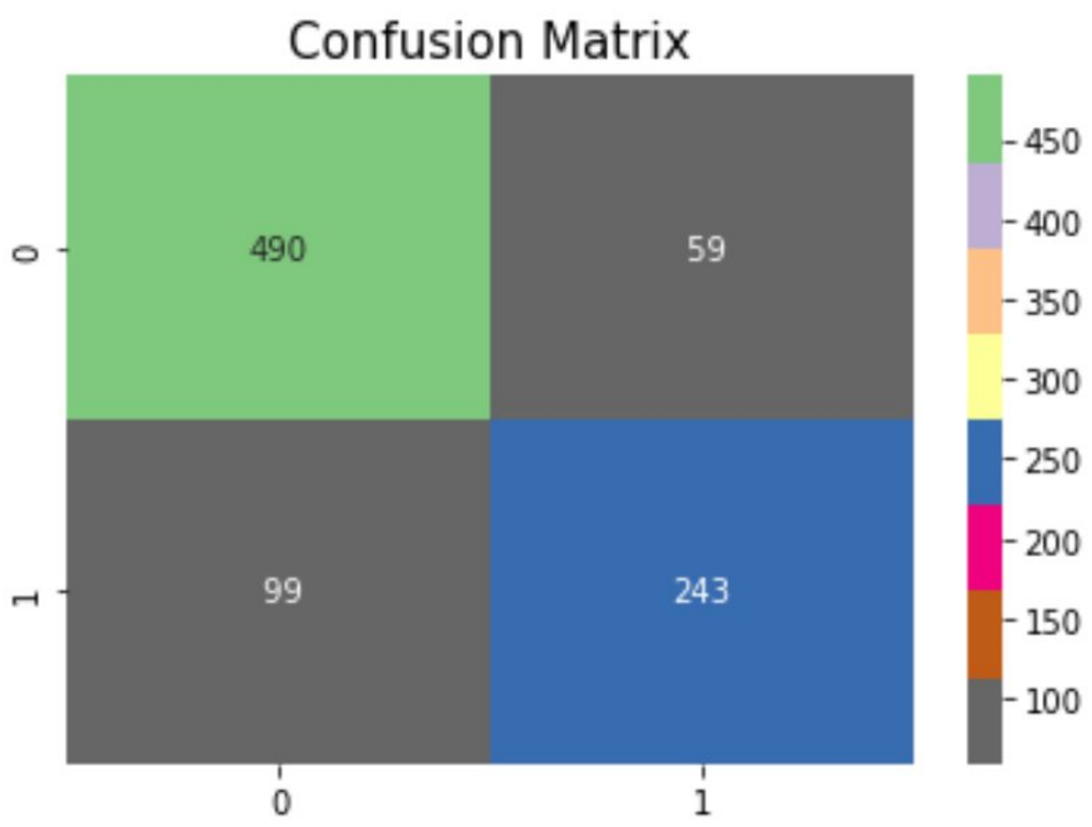


SVM

In [98]:

```
################################### SVM #########################
from sklearn.svm import SVC, LinearSVC
model = SVC()
model.fit(X_train,y_train)
prediction_svm=model.predict(X_test)

print('--------SVM -------')
print('The accuracy SVM is',round(accuracy_score(prediction_svm,y_test)*100,2))

kfold = KFold(n_splits=8,shuffle=True,random_state=25) # split the data into 10 equal part

result_svm=cross_val_score(model,X,y,cv=10,scoring='accuracy')

print('The cross validated score for SVM is:',round(result_svm.mean()*100,2))

y_pred = cross_val_predict(model,X,y,cv=10)
sns.heatmap(confusion_matrix(y,y_pred),annot=True,fmt='3.0f',cmap="Accent_r")
plt.title('Confusion Matrix', y=1, size=15);
```

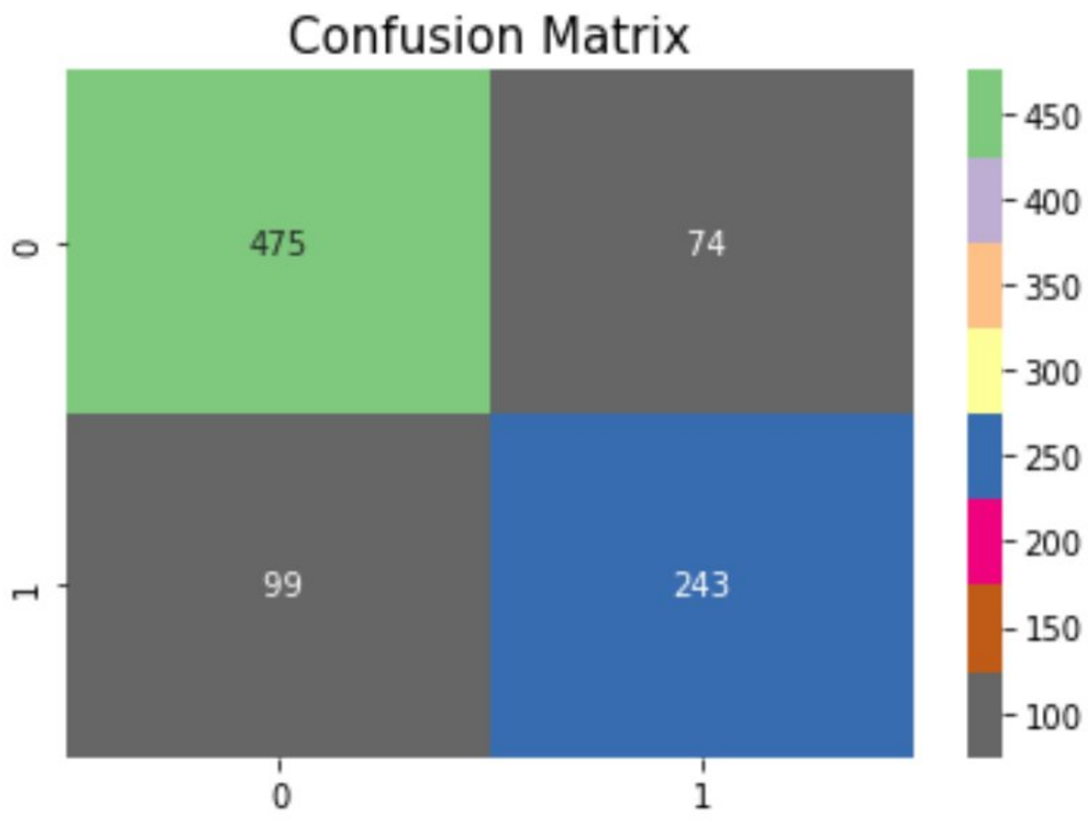
```
--------SVM -------
The accuracy SVM is 80.6
The cross validated score for SVM is: 82.27
```



KNN

In [99]:
```
##################################### KNN ##########################
from sklearn.neighbors import KNeighborsClassifier
model =  KNeighborsClassifier()
model.fit(X_train,y_train)
prediction_knn=model.predict(X_test)

print('--------KNN -------')
print('The accuracy KNN Classifier is',round(accuracy_score(prediction_knn,y_test)*100,2))

kfold = KFold(n_splits=8,shuffle=True, random_state=25) # split the data into 10 equal pai

result_knn=cross_val_score(model,X,y,cv=10,scoring='accuracy')

print('The cross validated score for KNN classifier is:',round(result_knn.mean()*100,2))

y_pred = cross_val_predict(model,X,y,cv=10)
sns.heatmap(confusion_matrix(y,y_pred),annot=True,fmt='3.0f',cmap="Accent_r")
plt.title('Confusion Matrix', y=1, size=15);
```

```
--------KNN -------
The accuracy KNN Classifier is 81.34
The cross validated score for KNN classifier is: 80.59
```

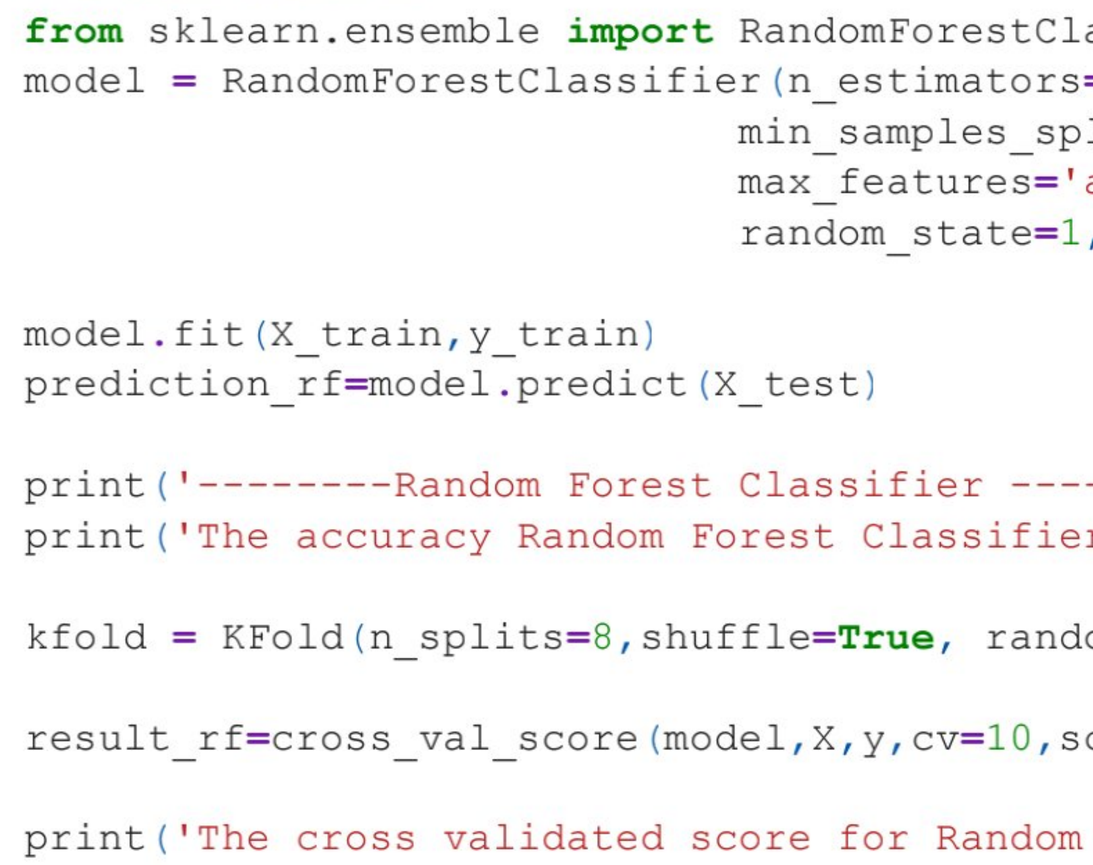


Logistic Regression

In [100…
```
##################################### Logistic Regression ##########################
from sklearn.linear_model import LogisticRegression
```

```
model =  LogisticRegression()
model.fit(X_train,y_train)
prediction_lr=model.predict(X_test)

print('--------Logistic Regression -------')
print('The accuracy Logistic Regression is',round(accuracy_score(prediction_lr,y_test)*100

kfold = KFold(n_splits=8,shuffle=True, random_state=42) # split the data into 10 equal pa

result_lr=cross_val_score(model,X,y,cv=10,scoring='accuracy')

print('The cross validated score for Logistic Regression is:',round(result_lr.mean()*100,2

y_pred = cross_val_predict(model,X,y,cv=10)
sns.heatmap(confusion_matrix(y,y_pred),annot=True,fmt='3.0f',cmap="Accent_r")
plt.title('Confusion Matrix', y=1, size=15);
```

```
--------Logistic Regression -------
The accuracy Logistic Regression is 79.85
The cross validated score for Logistic Regression is: 79.69
```



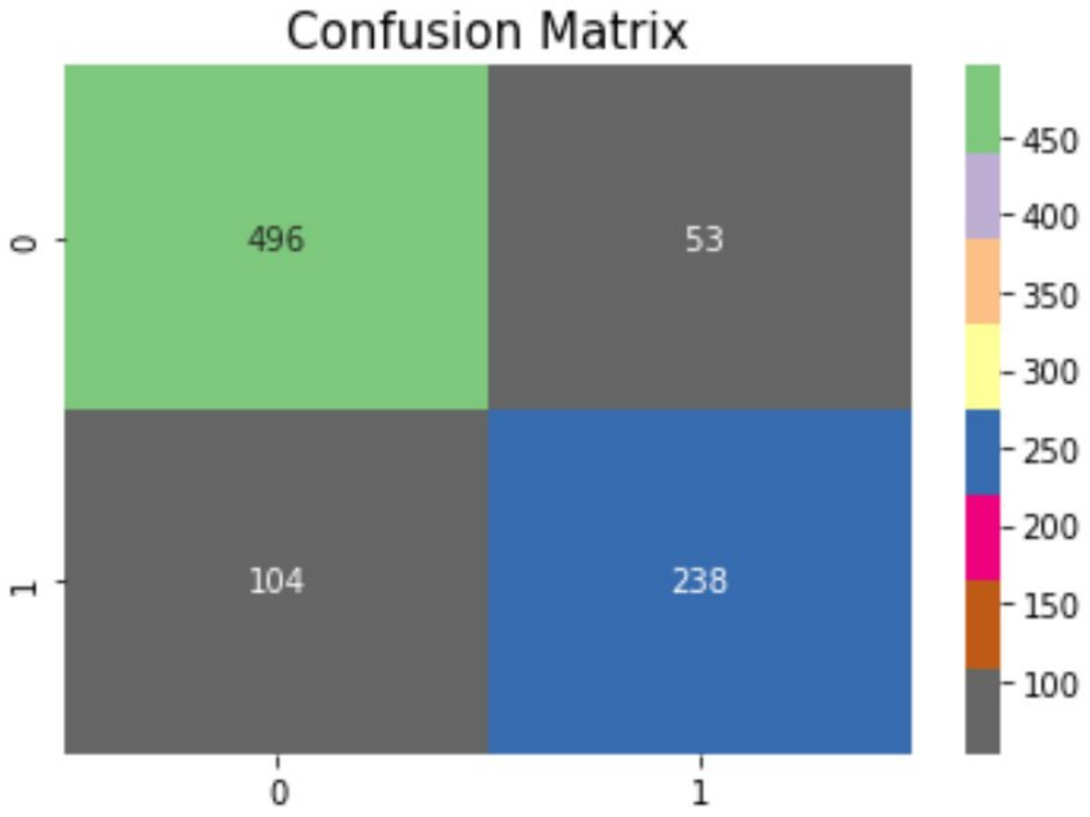
Random Forest

In [101…

```
# Random Forests
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=800,
                               min_samples_split=12,
                               max_features='auto',oob_score=True,
                               random_state=1,n_jobs=-1)

model.fit(X_train,y_train)
prediction_rf=model.predict(X_test)

print('--------Random Forest Classifier -------')
print('The accuracy Random Forest Classifier  is',round(accuracy_score(prediction_rf,y_tes

kfold = KFold(n_splits=8,shuffle=True, random_state=25) # split the data into 10 equal pa

result_rf=cross_val_score(model,X,y,cv=10,scoring='accuracy')

print('The cross validated score for Random Forest Classifier is:',round(result_rf.mean()*

y_pred = cross_val_predict(model,X,y,cv=10)
sns.heatmap(confusion_matrix(y,y_pred),annot=True,fmt='3.0f',cmap="Accent_r")
plt.title('Confusion Matrix', y=1, size=15);
```

```
--------Random Forest Classifier -------
```

```
The accuracy Random Forest Classifier  is 81.34
The cross validated score for Random Forest Classifier is: 82.39
```



Model Evaluation

In [103…
```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'K-Nearest Neighbour', 'Logistic Regression',
              'Random Forest', 'Naive Bayes'],
    'Score': [result_svm.mean(), result_knn.mean(),result_lr.mean(),
              result_rf.mean(), result_nb.mean()]})
models.sort_values(by='Score',ascending=False)
```

Out[103…

	Model	Score
3	Random Forest	0.823870
0	Support Vector Machines	0.822684
1	K-Nearest Neighbour	0.805893
2	Logistic Regression	0.796891
4	Naive Bayes	0.777828

Test Data

In [104…
```
test = clean_data(df_test)
```

Using Random Forest to predict test data

In [107…
```
type(model)
```

Out[107… `sklearn.ensemble._forest.RandomForestClassifier`

In [108…
```
y_pred = model.predict(test)
```

In [110…
```
print(y_pred)
```

```
[0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0
 1 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0
```

```
1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1
0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 1 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0
1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 1
0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0
1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0
0 1 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0
0 1 1 1 1 1 0 1 0 0 1]
```

In [111…

```
submission = pd.DataFrame({
        "PassengerId": df_test["PassengerId"],
        "Survived": y_pred})
```

In [112…

```
submission
```

Out[112…

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	1

418 rows × 2 columns