Online_foods_Project

Import libraries

In [8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import     roc_curve,auc,roc_auc_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix ,ConfusionMatrixDisplay ,accur
```

Load Dataset

In [3]:
```
food_df = pd.read_csv('onlinefoods.csv')
food_df
```

Out[3]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitu
0	20	Female	Single	Student	No Income	Post Graduate	4	12.9
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	12.9
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	12.9
3	22	Female	Single	Student	No Income	Graduate	6	12.9
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	12.9
...	...	...	...	...	...	...	...	
383	23	Female	Single	Student	No Income	Post Graduate	2	12.9
384	23	Female	Single	Student	No Income	Post Graduate	4	12.9
385	22	Female	Single	Student	No Income	Post Graduate	5	12.9
386	23	Male	Single	Student	Below Rs.10000	Post Graduate	2	12.9
387	23	Male	Single	Student	No Income	Post Graduate	5	12.8

388 rows × 13 columns

In [3]:
```
food_df.head(5)
```

Out[3]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitude
0	20	Female	Single	Student	No Income	Post Graduate	4	12.976
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	12.977
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	12.955
3	22	Female	Single	Student	No Income	Graduate	6	12.947
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	12.985

In [4]:
```
food_df.tail(5)
```

Out[4]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitu
383	23	Female	Single	Student	No Income	Post Graduate	2	12.9
384	23	Female	Single	Student	No Income	Post Graduate	4	12.9
385	22	Female	Single	Student	No Income	Post Graduate	5	12.9
386	23	Male	Single	Student	Below Rs.10000	Post Graduate	2	12.9
387	23	Male	Single	Student	No Income	Post Graduate	5	12.8

In [5]:

```
#checking shape of the data
food_df.shape
```

Out[5]:

```
(388, 13)
```

In [6]:

```
food_df.columns
```

Out[6]:

```
Index(['Age', 'Gender', 'Marital Status', 'Occupation', 'Monthly Income',
       'Educational Qualifications', 'Family size', 'latitude', 'longitud
e',
       'Pin code', 'Output', 'Feedback', 'Unnamed: 12'],
      dtype='object')
```

In [7]:

```
food_df.dtypes
```

Out[7]:

```
Age                             int64
Gender                         object
Marital Status                 object
Occupation                     object
Monthly Income                 object
Educational Qualifications     object
Family size                     int64
latitude                      float64
longitude                     float64
Pin code                        int64
Output                         object
Feedback                       object
Unnamed: 12                    object
dtype: object
```

Handeling duplicate data

In [8]:

```
food_df.isnull().sum()
```

Out[8]:
```
Age                            0
Gender                         0
Marital Status                 0
Occupation                     0
Monthly Income                 0
Educational Qualifications     0
Family size                    0
latitude                       0
longitude                      0
Pin code                       0
Output                         0
Feedback                       0
Unnamed: 12                    0
dtype: int64
```

Drop duplicates value

In [9]:
```
food_df.duplicated().sum()
```

Out[9]:
```
103
```

Step-5: Do some data preprocessing

if any column corrupted

ex. Numerical values in categorical columns

ex. Categorical values in numerical columns

In [10]:
```
food_df.drop_duplicates(inplace=True)
```

Step-6: Drop the id type columns

Which means a data has more unique labels

Drop the single value columns

In [11]:
```
food_df.shape
```

Out[11]:
```
(285, 13)
```

In [12]:
```
food_df.head()
```

Out[12]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitude
0	20	Female	Single	Student	No Income	Post Graduate	4	12.976
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	12.977
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	12.955
3	22	Female	Single	Student	No Income	Graduate	6	12.947
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	12.985

tep-7: Categorical column Analysis

In [59]:
```
# Analysing categorical columns
categorical=food_df.select_dtypes(include='object').columns
categorical
```

Out[59]:
```
Index(['Gender', 'Marital Status', 'Occupation', 'Monthly Income',
       'Educational Qualifications', 'Output', 'Feedback'],
      dtype='object')
```

In [14]:
```
unique=food_df['Gender'].unique()
unique
```

Out[14]:
```
array(['Female', 'Male'], dtype=object)
```

In [15]:
```
food_df[['Gender']].value_counts()
```

Out[15]:
```
Gender
Male      164
Female    121
Name: count, dtype: int64
```

In [16]:
```
# count=[]
# for i in unique:
#  con=food_df['Gender']==i
#  count.append(len(food_df[con]))

# count
```

In [17]:
```
# df=pd.DataFrame(zip(unique, count), columns=['labels','count'])
# df
```

In [18]:
```
# count=[]
# for i in unique:
#  con=food_df['Gender']==i
#  count.append(len(food_df[con]))
```

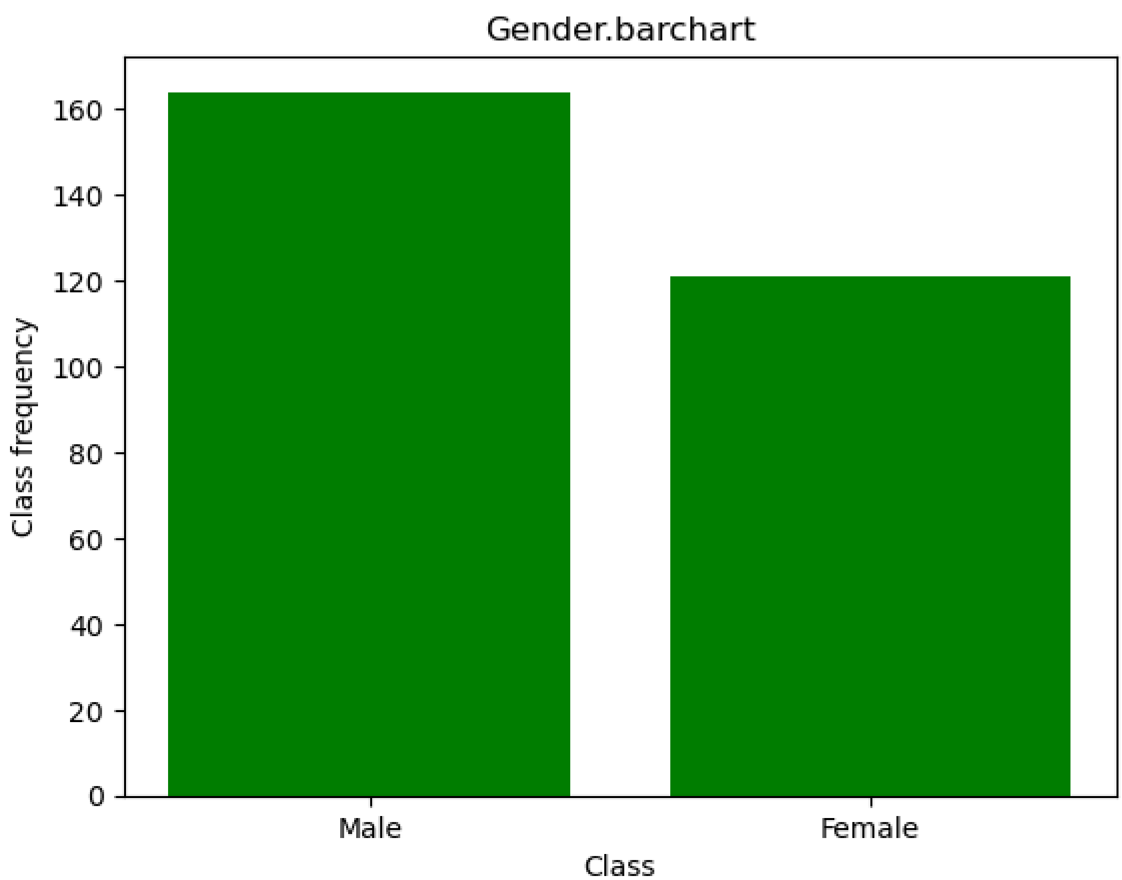
```
# count
```

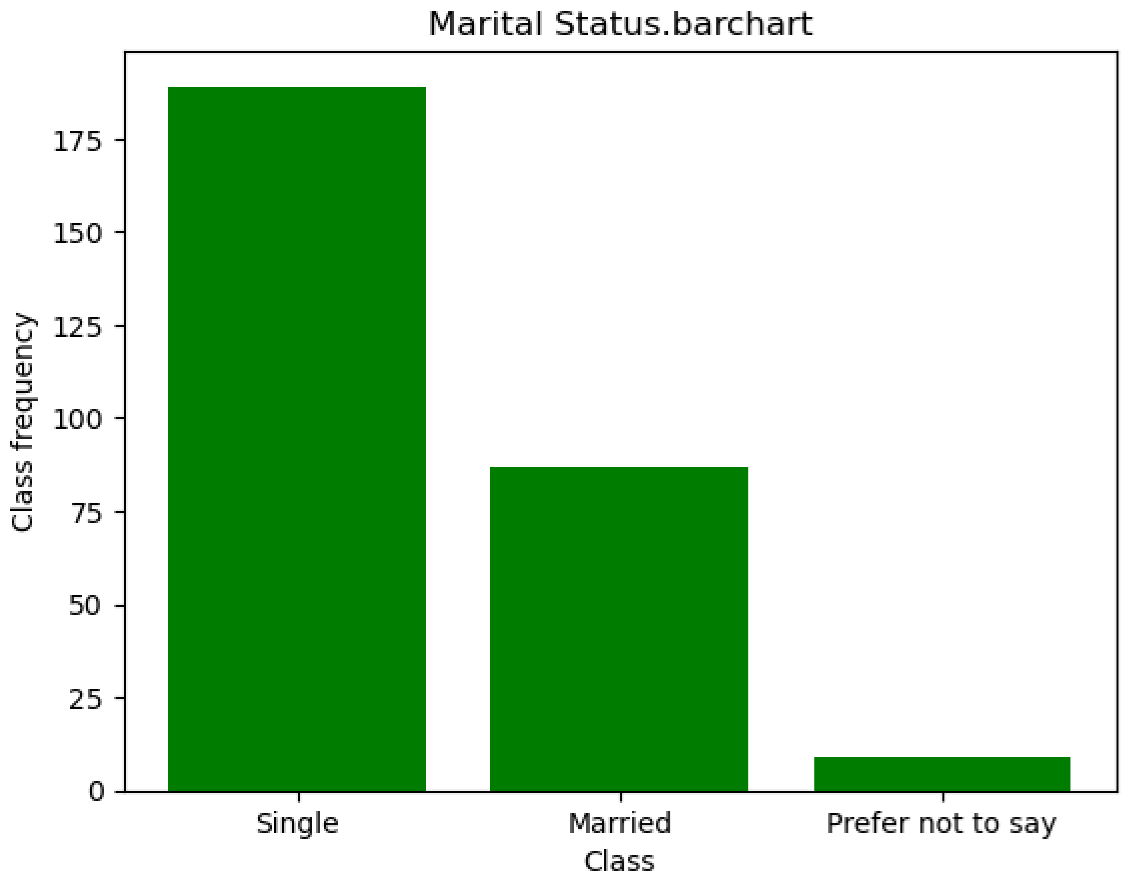
Barchart

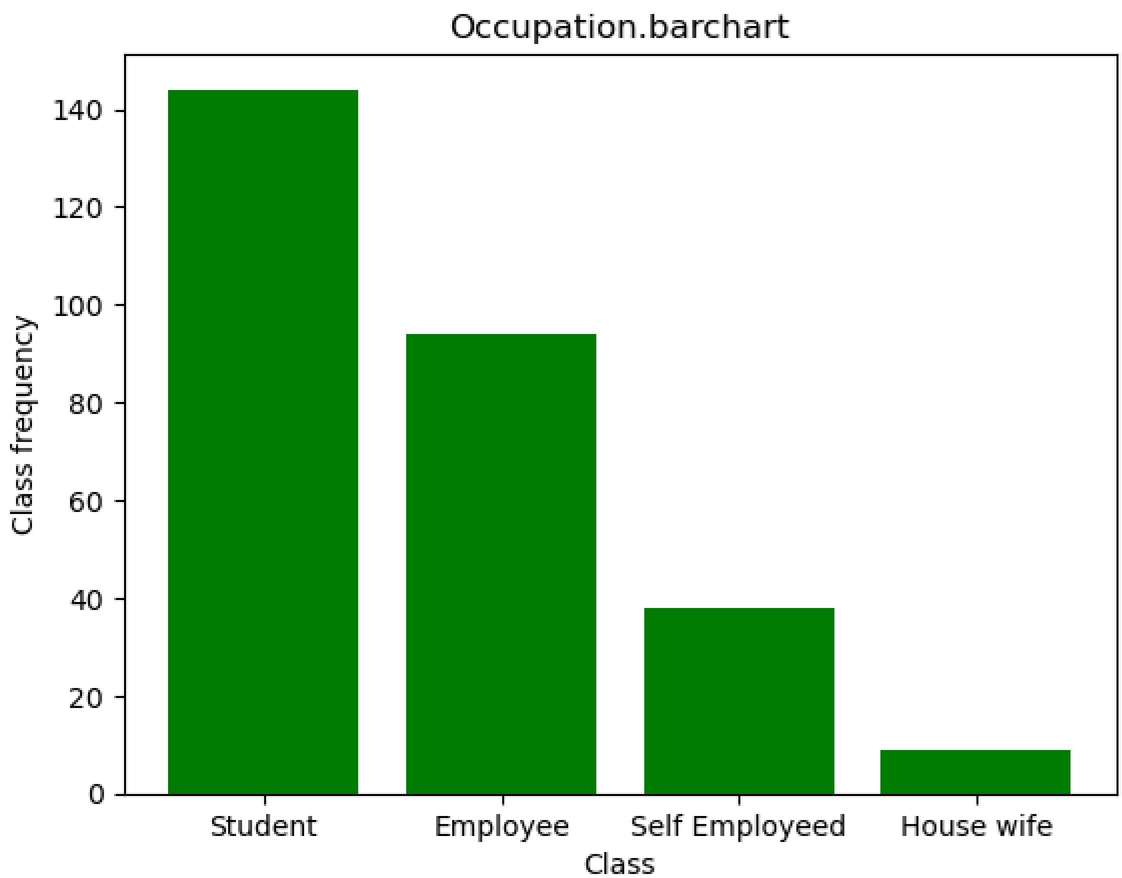
In [19]:
```
categorical
```

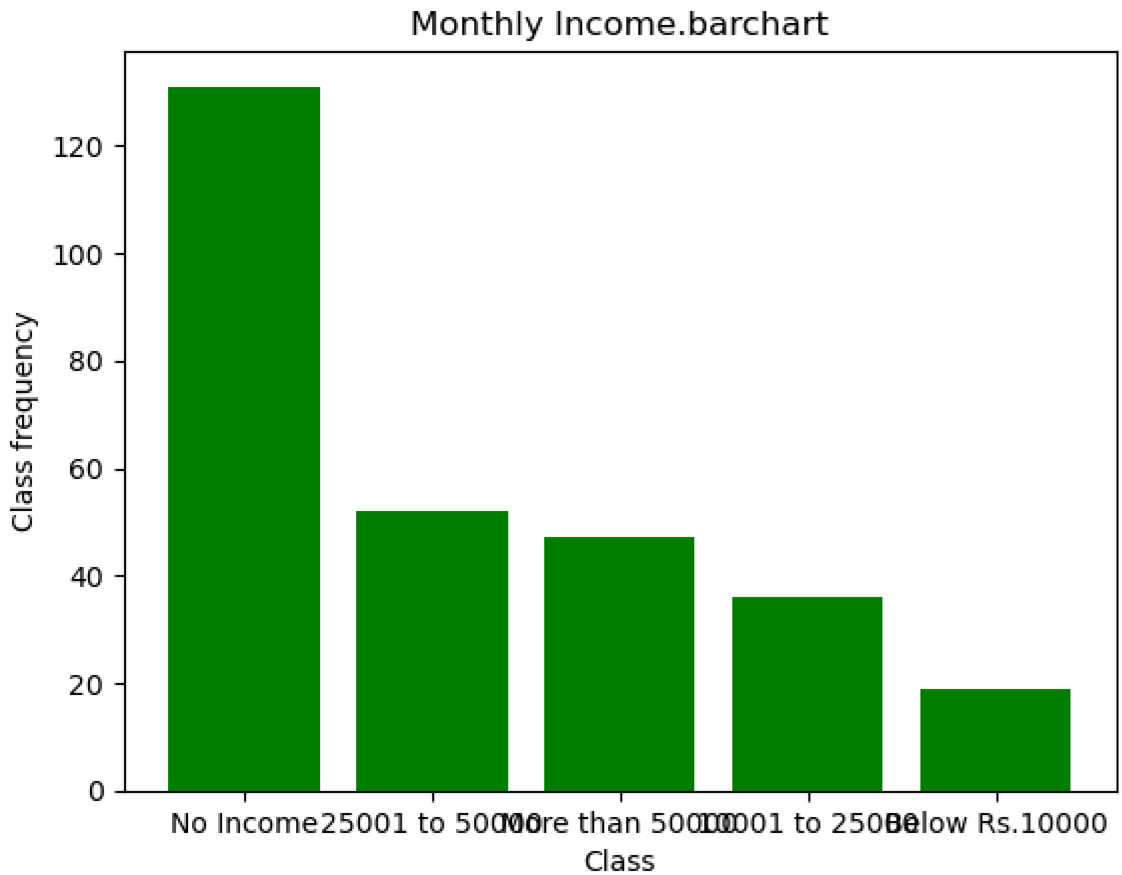
Out[19]:
```
Index(['Gender', 'Marital Status', 'Occupation', 'Monthly Income',
       'Educational Qualifications', 'Output', 'Feedback', 'Unnamed: 12'],
      dtype='object')
```

In [20]:
```
for i in categorical:
 keys=food_df[i].value_counts().keys()
 values=food_df[i].value_counts().values
 plt.bar(keys, values)
 plt.title(f'{i}.barchart')
 plt.xlabel('Class')
 plt.ylabel( 'Class frequency')
 plt.savefig('i.png')
 plt.bar(keys, values, color='green')
 plt.show()
```

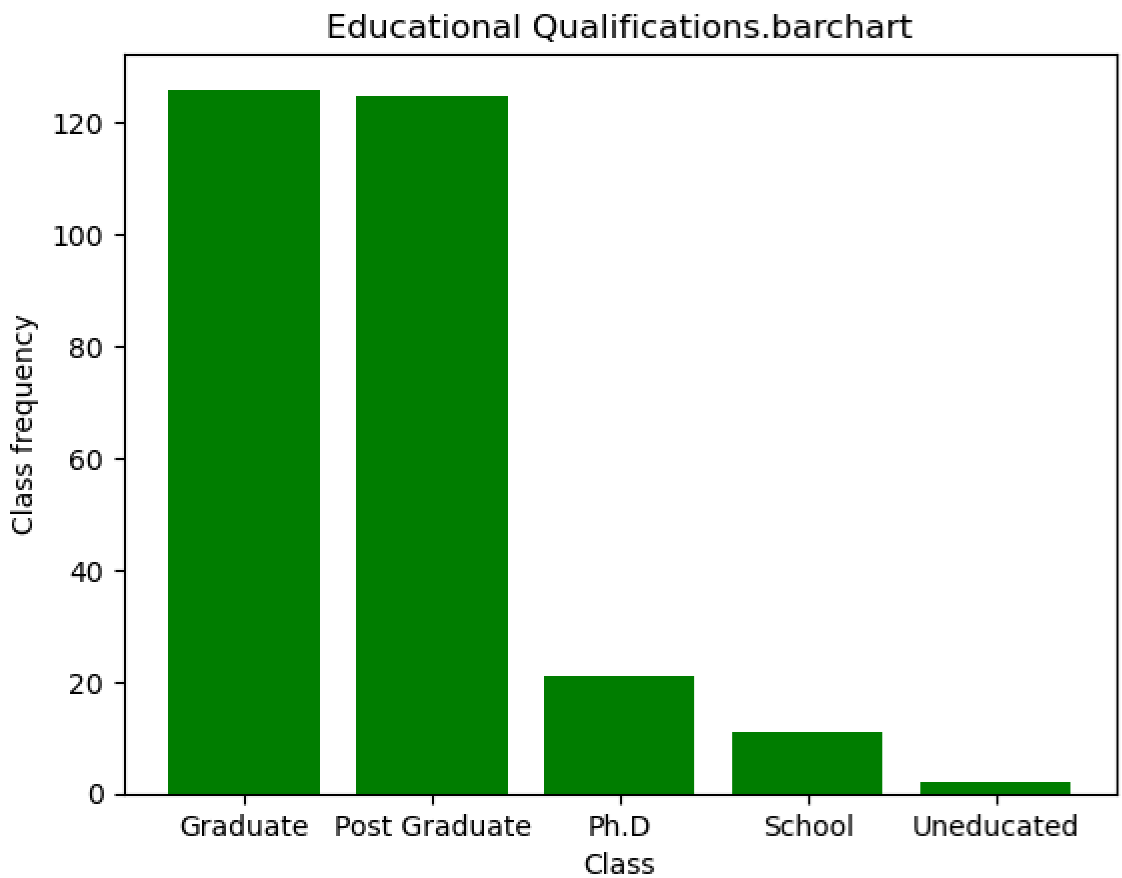



Marital Status.barchart
Class frequency
175
150
125
100
75
50
25
0
Single
Married
Prefer not to say
Class

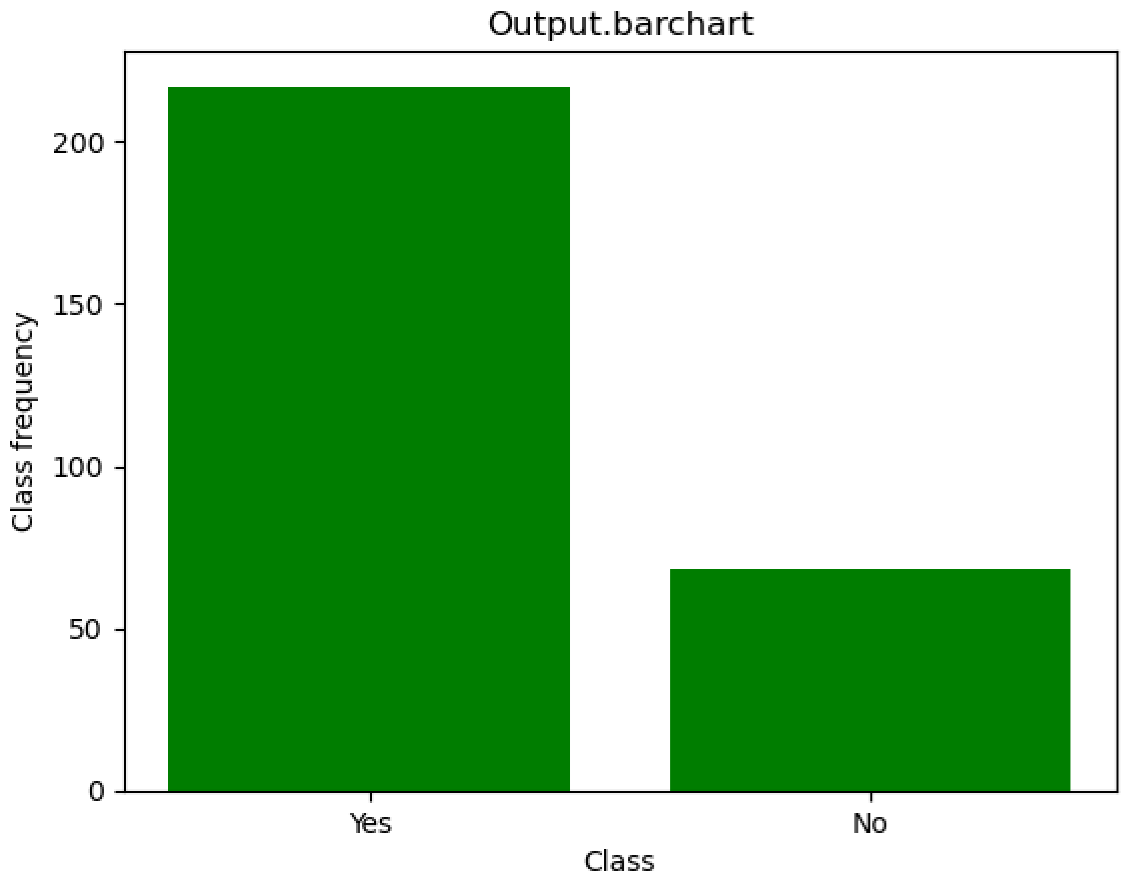

Occupation.barchart
Class frequency
140
120
100
80
60
40
20
0
Student
Employee
Self Employeed
House wife
Class

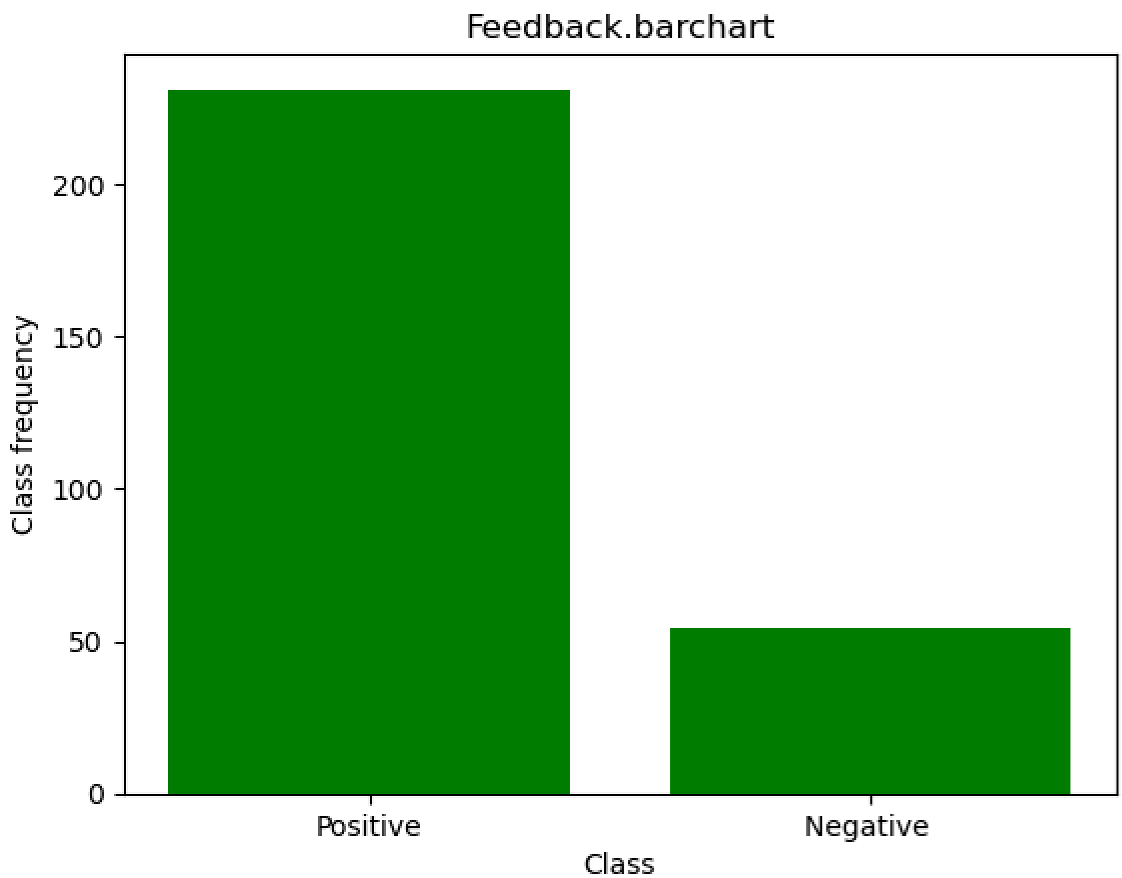


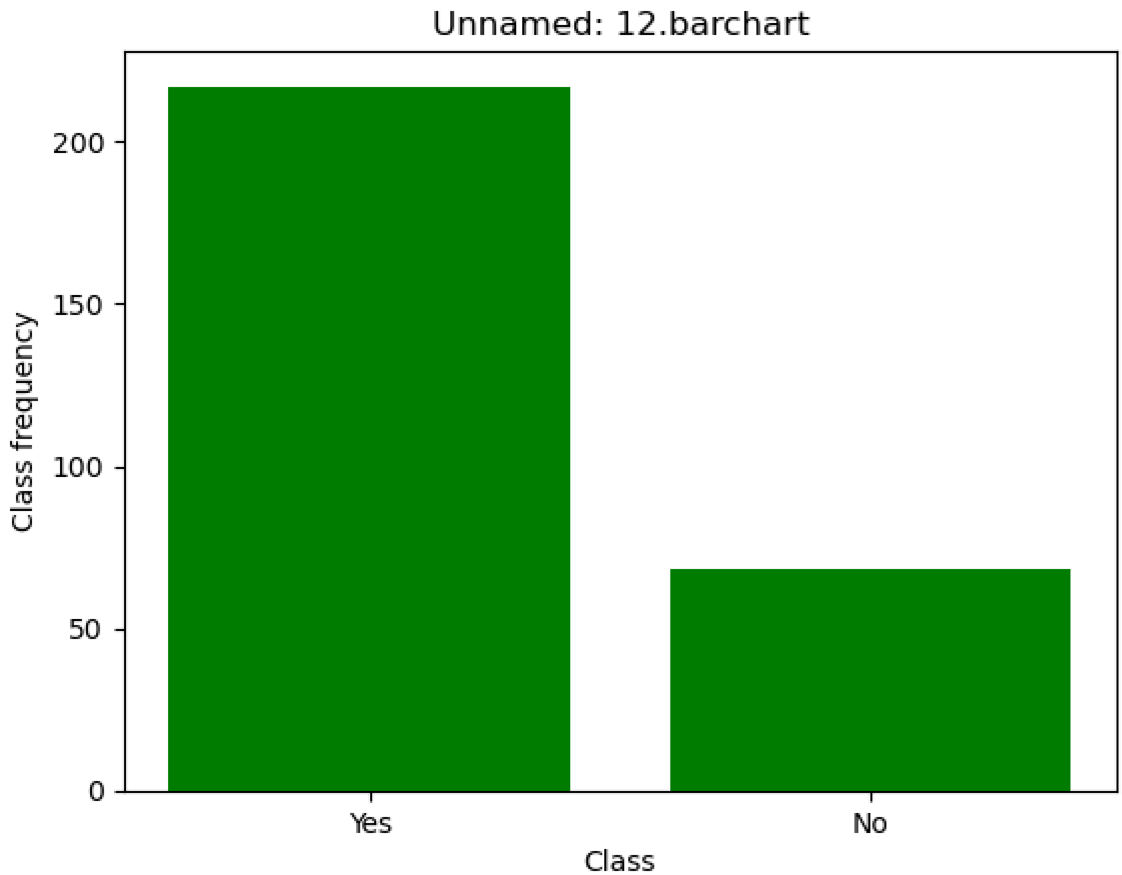
Monthly Income.barchart
Class frequency
0
20
40
60
80
100
120
No Income
25001 to 50000
More than 50000
10001 to 25000
Below Rs.10000
Class



Educational Qualifications.barchart
Class frequency
0
20
40
60
80
100
120
Graduate
Post Graduate
Ph.D
School
Uneducated
Class

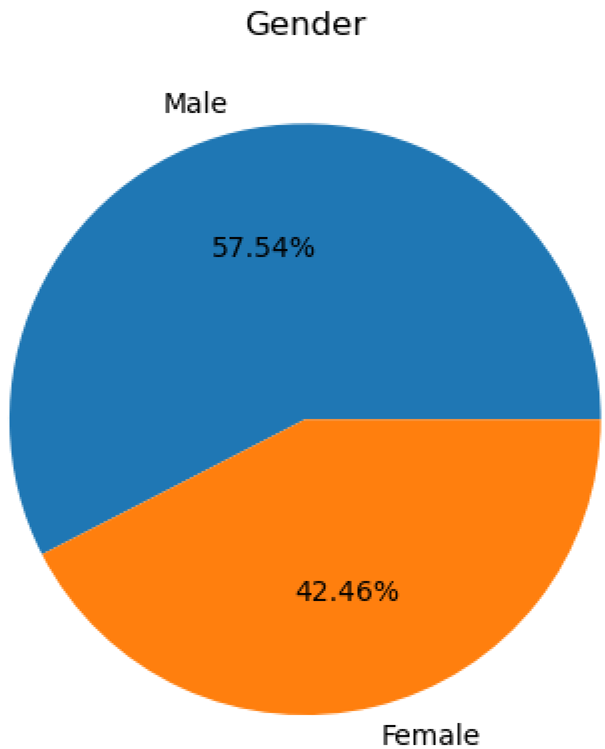

Output.barchart
Class frequency
0
50
100
150
200
Yes
No
Class

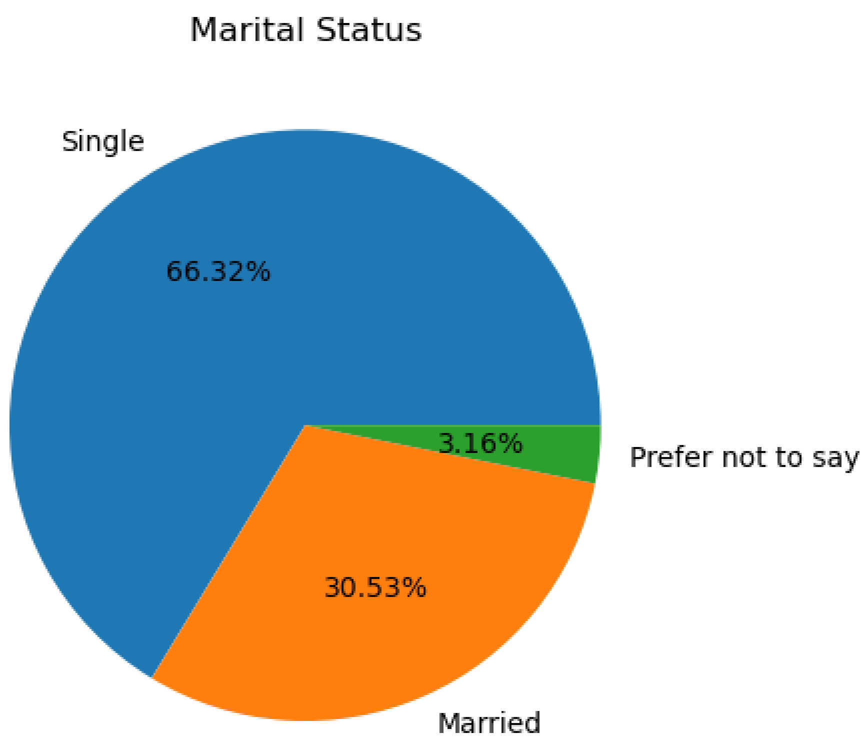

Feedback.barchart
Class frequency
0
50
100
150
200
Positive
Negative
Class

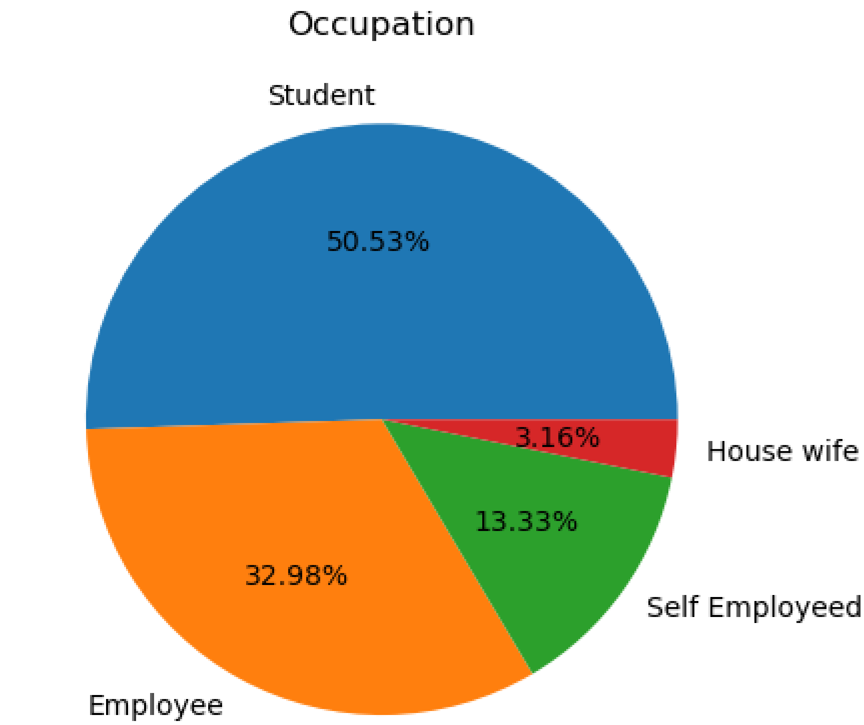


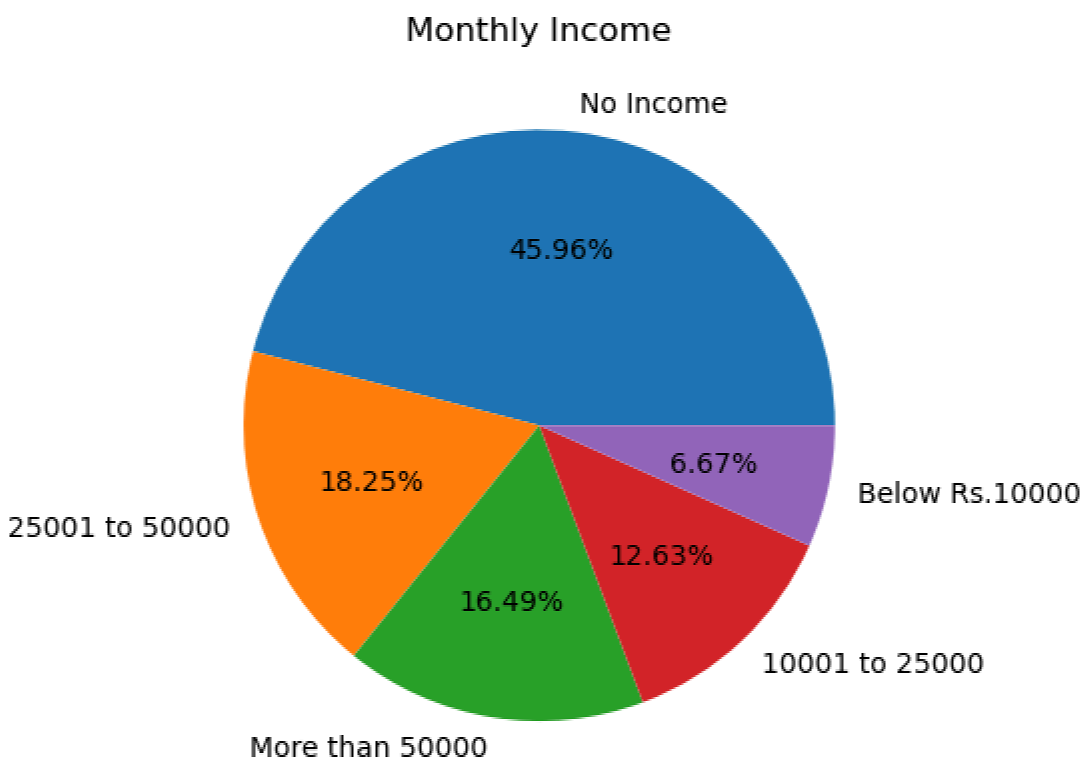
Piechart

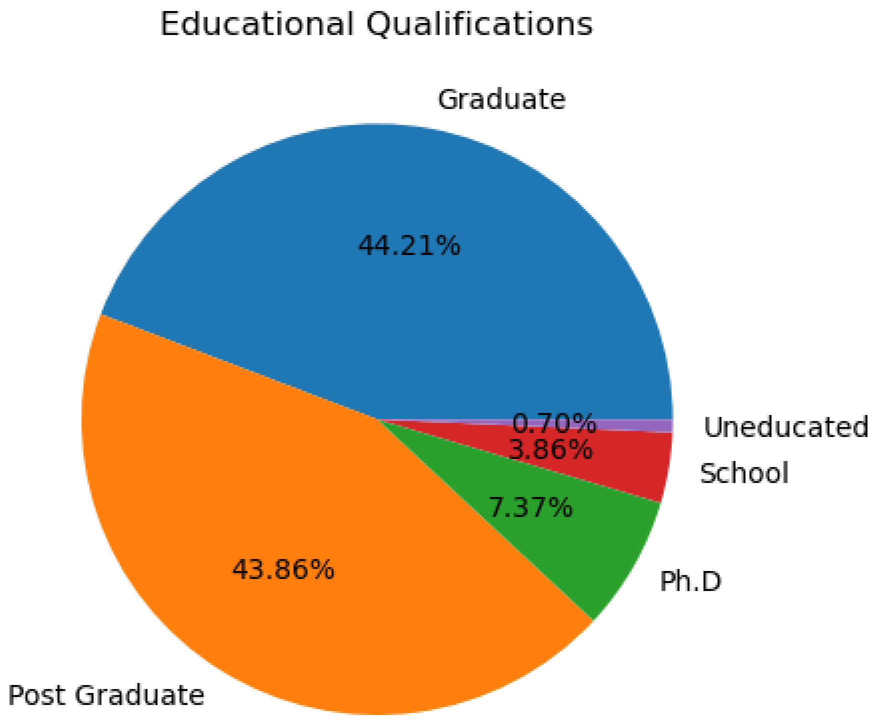
In [21]:
```
for i in categorical:
 keys=food_df[i].value_counts().keys()
 values=food_df[i].value_counts().values
 plt.pie(x=values, labels=keys, autopct='%0.2f%%')
 plt.savefig('i.png')
 plt.title(i)
 plt.show()
```

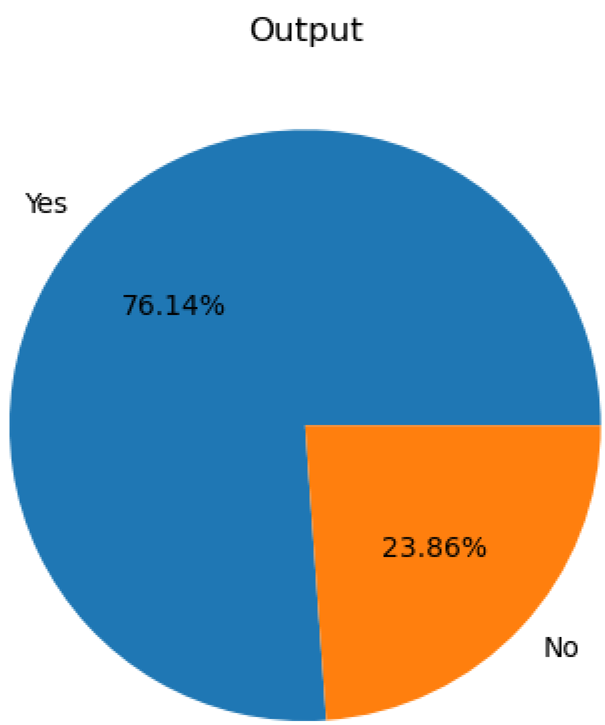

Gender
Male
57.54%
42.46%
Female

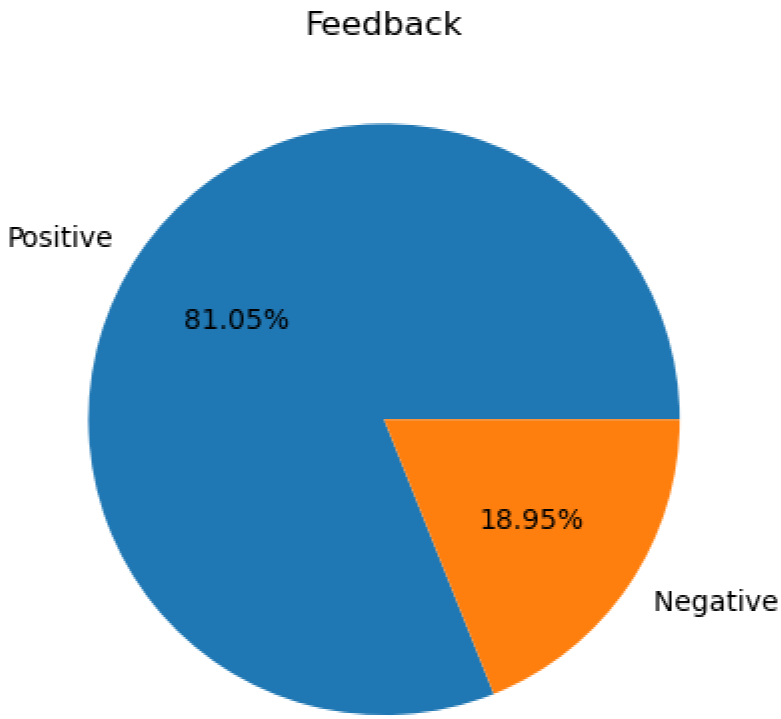

Marital Status
Single
66.32%
3.16%
Prefer not to say
30.53%
Married

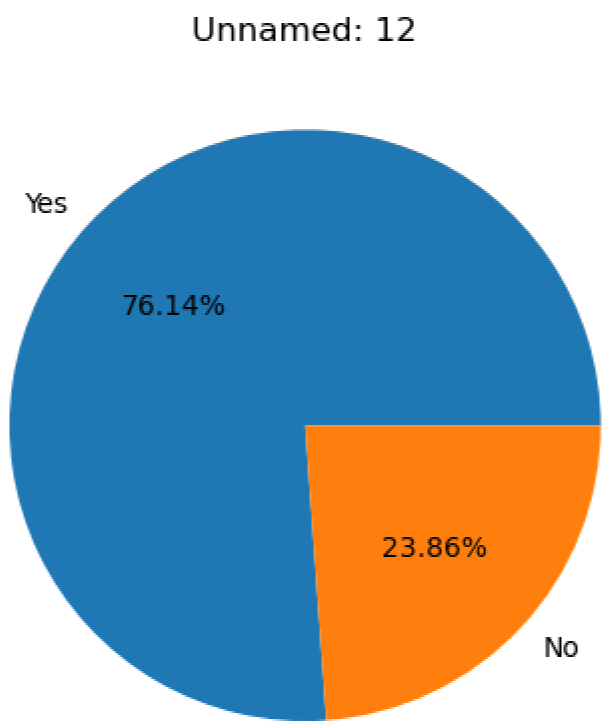

Occupation
Student
50.53%
3.16%
House wife
13.33%
32.98%
Self Employeed
Employee


Monthly Income
No Income
45.96%
6.67%
18.25%
Below Rs.10000
25001 to 50000
12.63%
16.49%
10001 to 25000
More than 50000


Educational Qualifications
Graduate
44.21%
0.70%
Uneducated
3.86%
School
7.37%
Ph.D
43.86%
Post Graduate

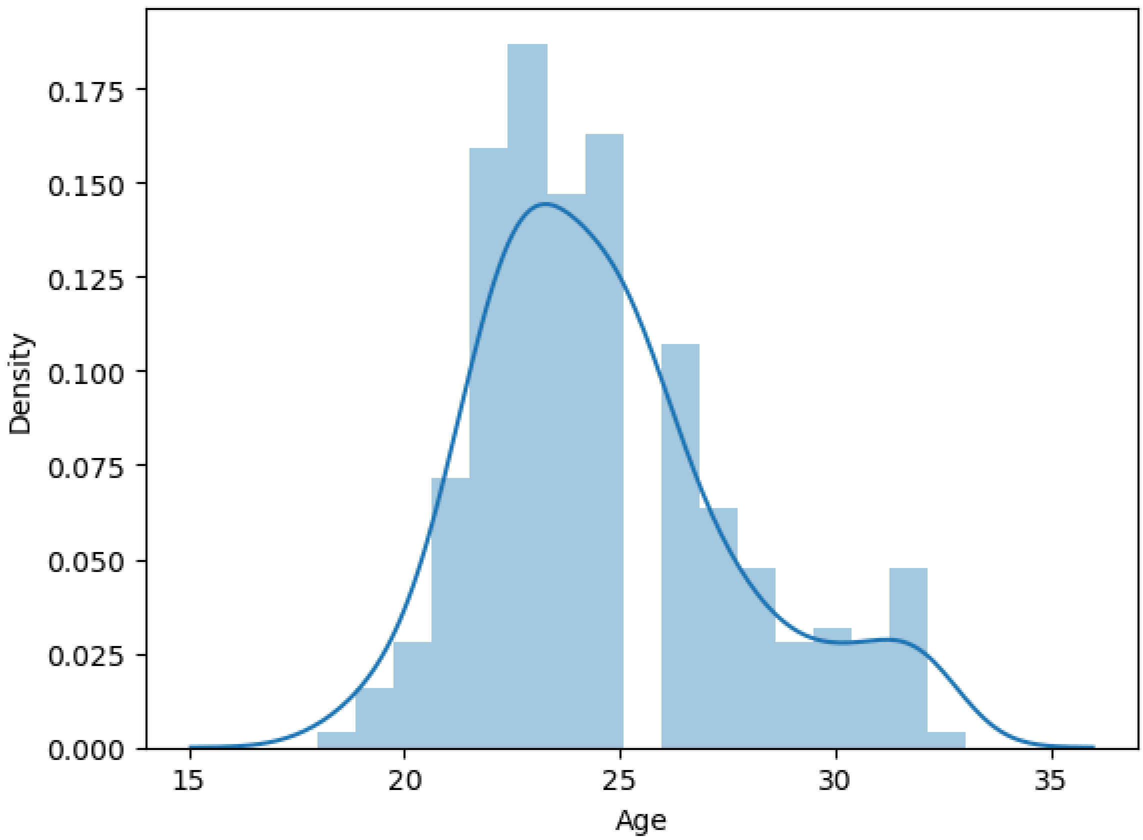

Output
Yes
76.14%
23.86%
No





In [22]:
```
import warnings
warnings.filterwarnings('ignore')

# def distplots(col):
sns.distplot(food_df['Age'])
plt.show()
```

```python
# for i in list(food_df.columns):
# distplots(i)
```



In [23]:
```python
numerical=food_df.select_dtypes(exclude='object').columns
numerical
```

Out[23]:
```
Index(['Age', 'Family size', 'latitude', 'longitude', 'Pin code'], dtype='o
bject')
```

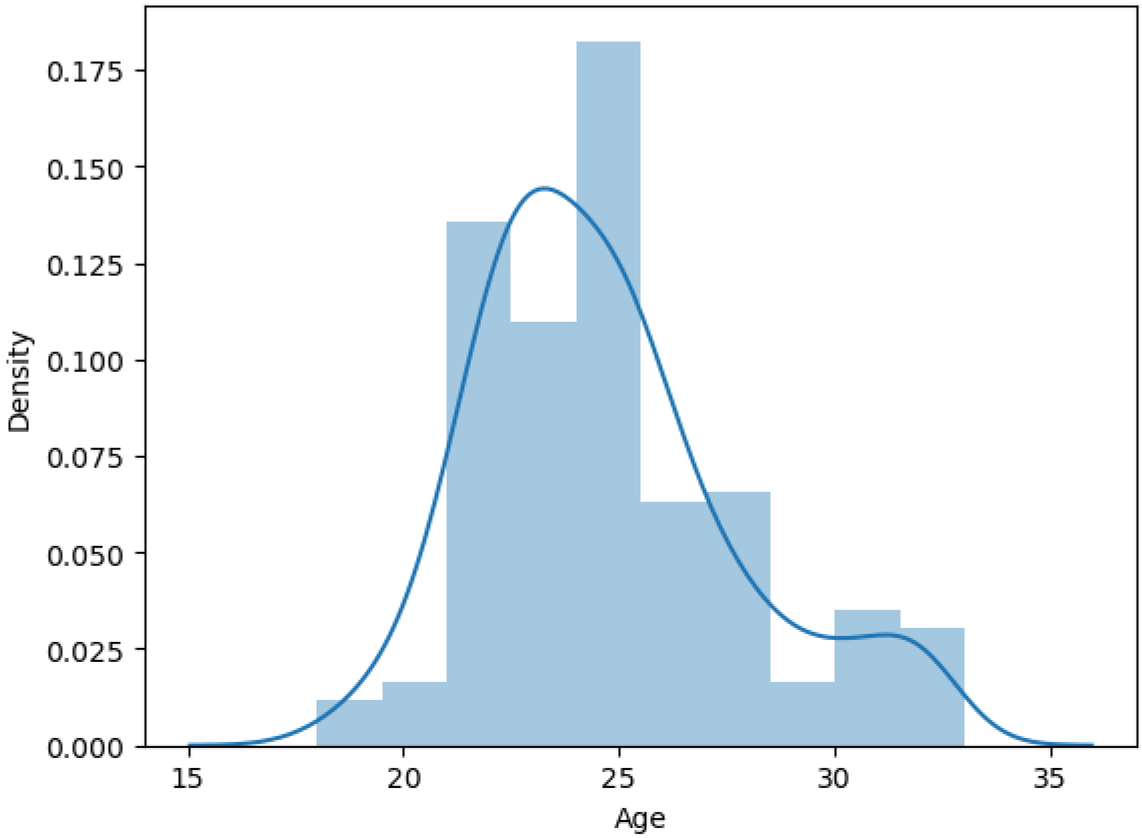
In [24]:
```python
food_df.describe()
```

Out[24]:

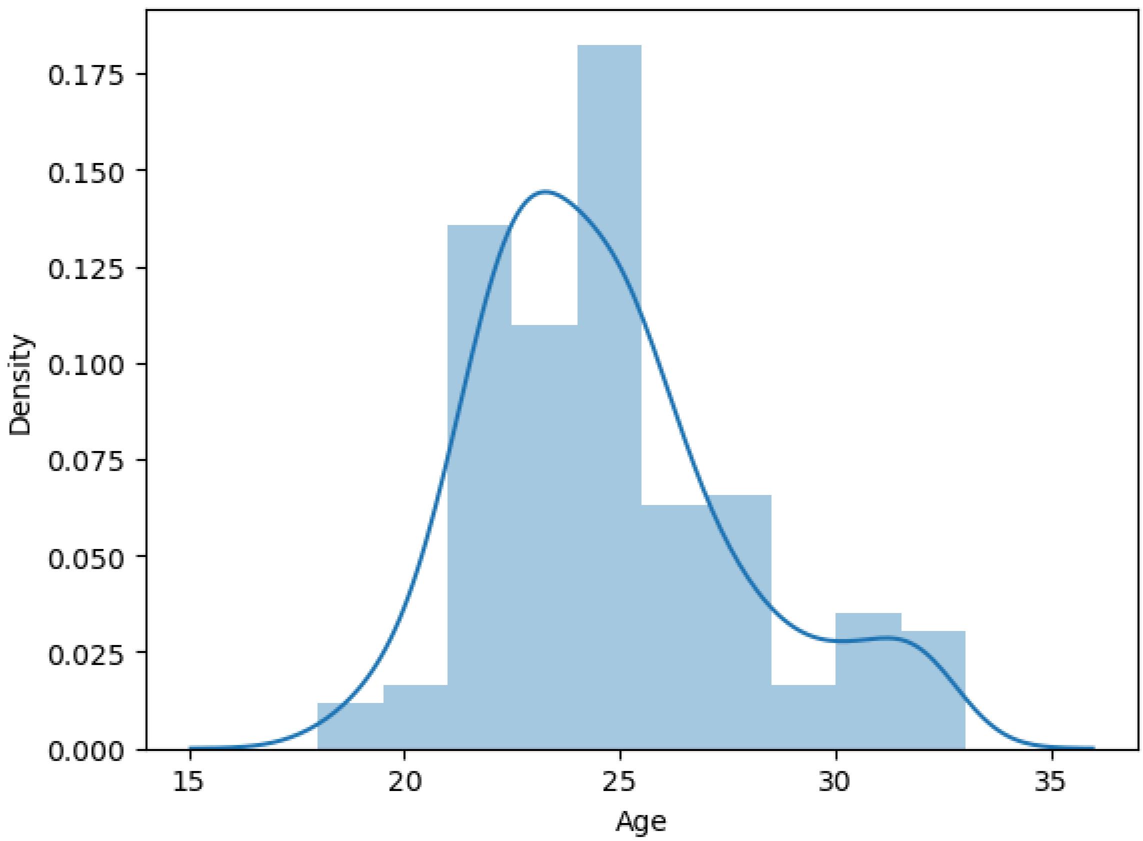
	Age	Family size	latitude	longitude	Pin code
count	285.000000	285.000000	285.000000	285.000000	285.000000
mean	24.677193	3.270175	12.973429	77.597593	560037.280702
std	3.040977	1.361178	0.043964	0.053557	30.738306
min	18.000000	1.000000	12.865200	77.484200	560001.000000
25%	23.000000	2.000000	12.943800	77.563500	560010.000000
50%	24.000000	3.000000	12.977000	77.587700	560028.000000
75%	26.000000	4.000000	12.998000	77.622700	560066.000000
max	33.000000	6.000000	13.102000	77.758200	560109.000000

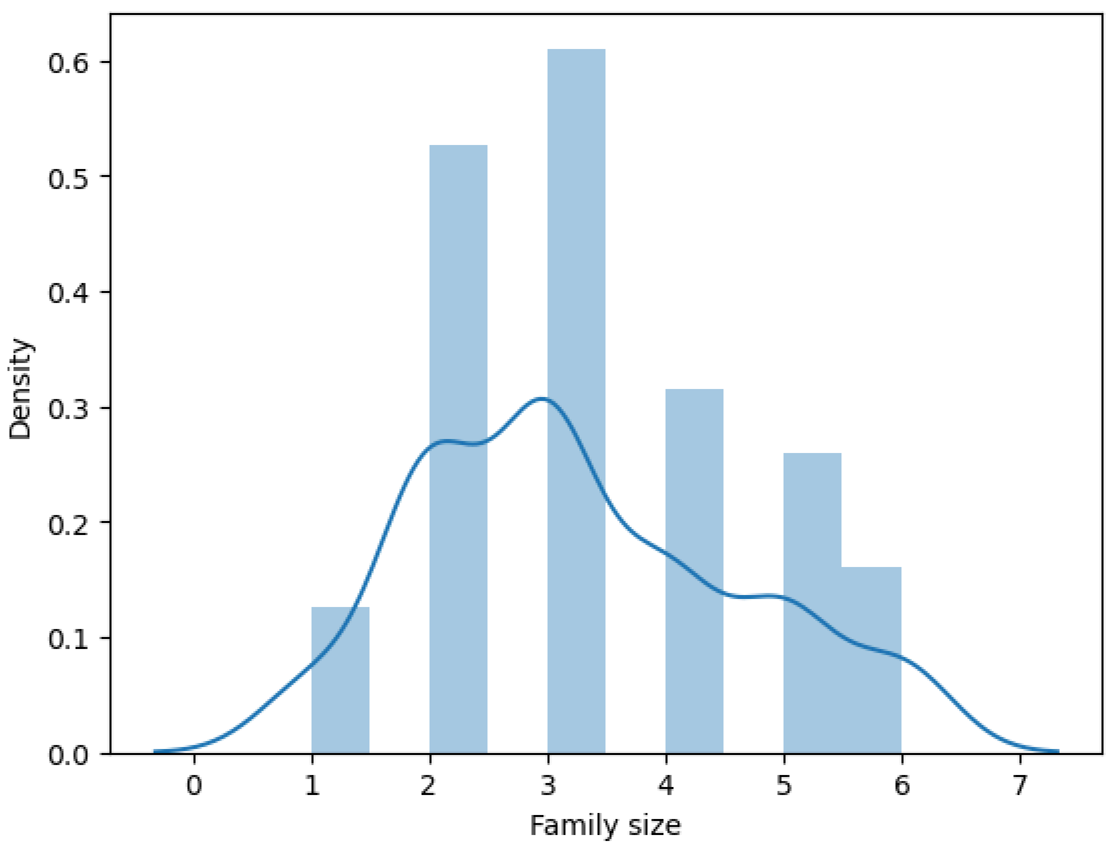
Distplot

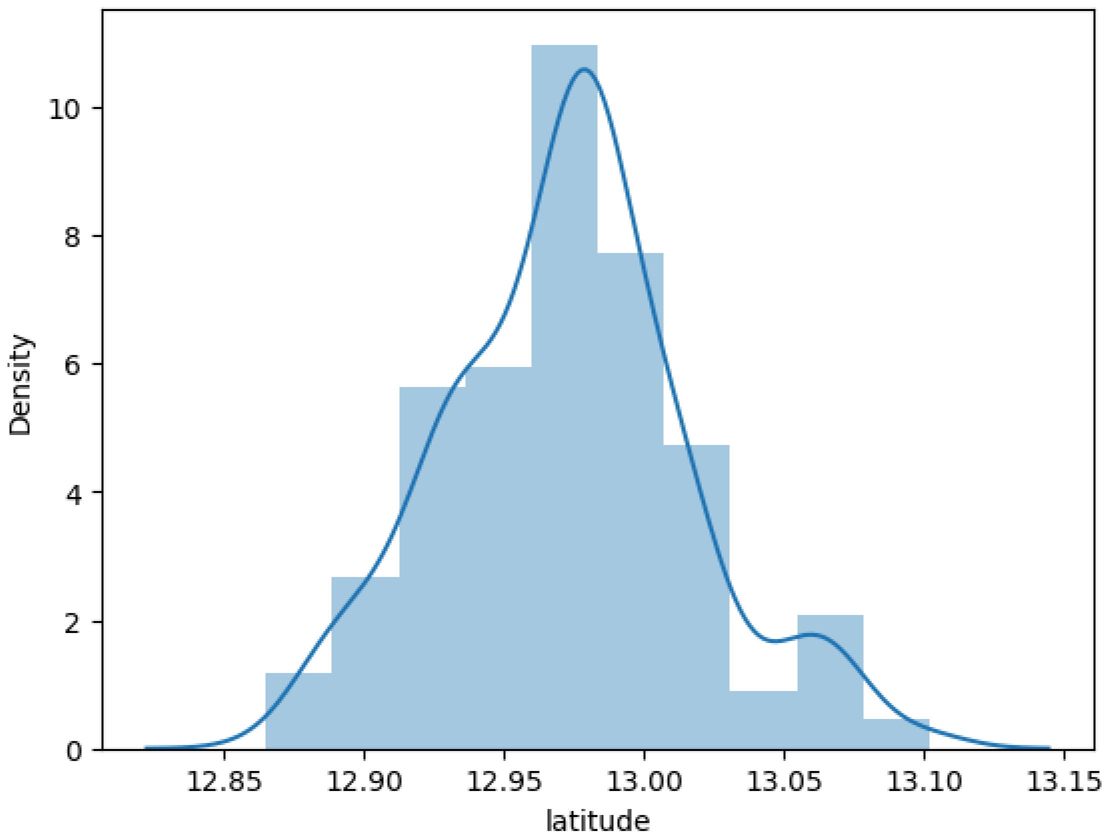
In [25]:
```
import warnings
warnings.filterwarnings('ignore')
sns.distplot(food_df['Age'], bins=10)
plt.show()
```



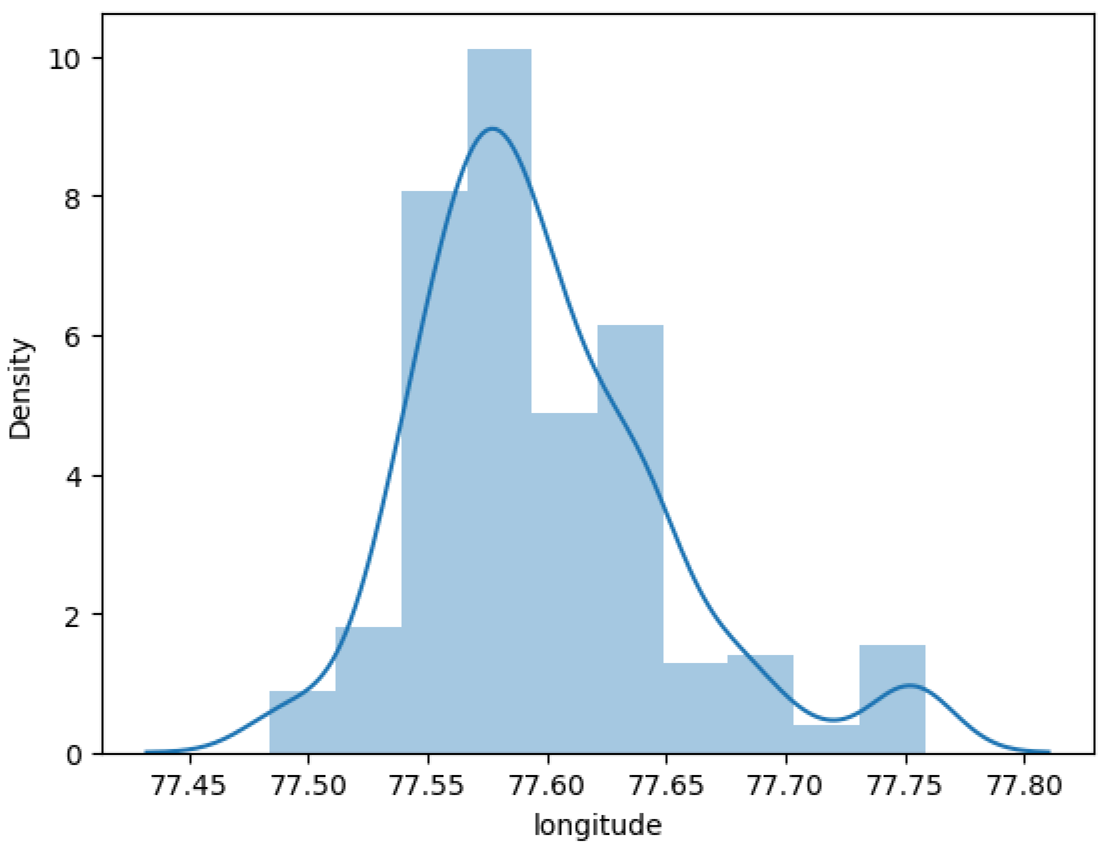
In [26]:
```
for i in numerical:
 sns.distplot(food_df[i], bins=10)
 plt.show()
```


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
15
20
25
30
35
Age

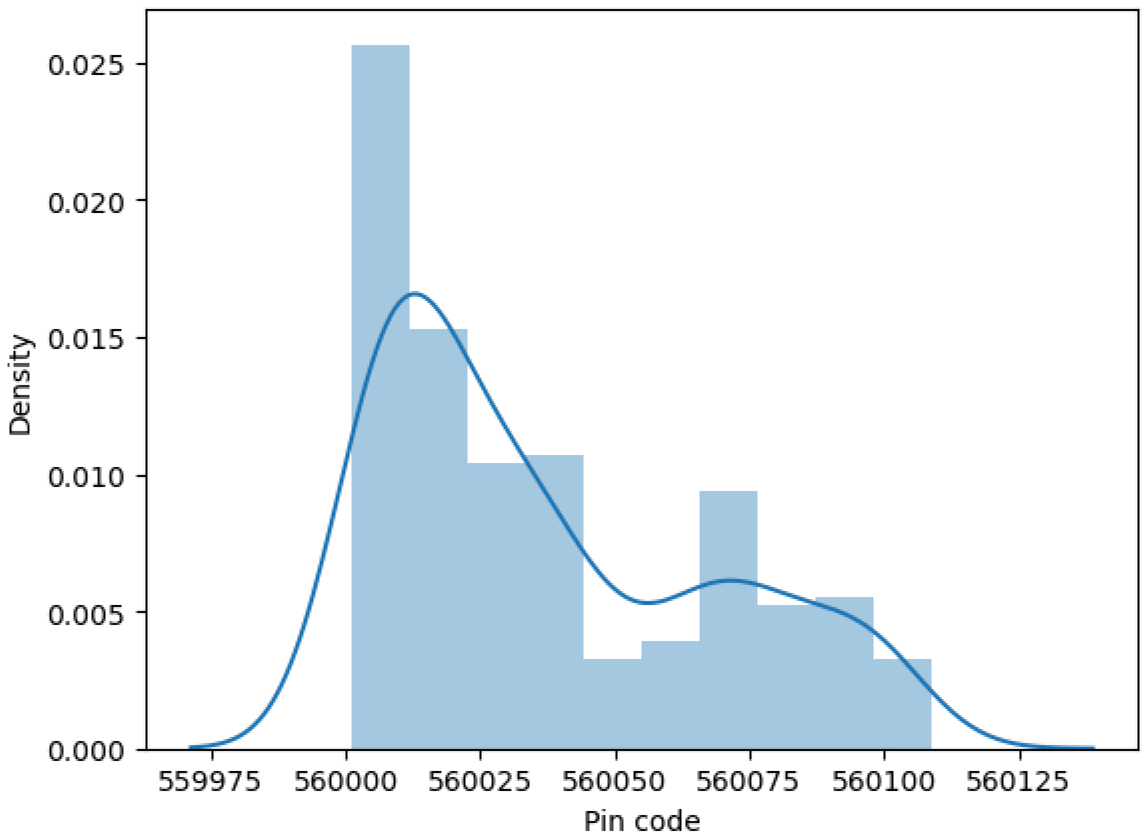

0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
0
1
2
3
4
5
6
7
Family size



Density
latitude
0
2
4
6
8
10
12.85
12.90
12.95
13.00
13.05
13.10
13.15



Density
longitude
0
2
4
6
8
10
77.45
77.50
77.55
77.60
77.65
77.70
77.75
77.80



Step-9: Outlier Analysis

In [27]:
```
numerical
```

Out[27]:
```
Index(['Age', 'Family size', 'latitude', 'longitude', 'Pin code'], dtype='o
bject')
```

In [28]:
```
for i in food_df.columns:
    print("*" * 30, f"{i}", "*" * 30)  # Print column name dynamically

    # Print the number of unique values
    print(f"Number of unique values in '{i}': {food_df[i].nunique()}")

    # Print the unique values
    print(f"Unique values in '{i}': {food_df[i].unique()}")

    print("*" * 50)
    print("\n")
```

```
****************************** Age ******************************
Number of unique values in 'Age': 16
Unique values in 'Age': [20 24 22 27 23 21 28 25 32 30 31 26 18 19 33 29]
**************************************************


****************************** Gender ******************************
Number of unique values in 'Gender': 2
Unique values in 'Gender': ['Female' 'Male']
**************************************************


****************************** Marital Status ******************************
Number of unique values in 'Marital Status': 3
Unique values in 'Marital Status': ['Single' 'Married' 'Prefer not to say']
**************************************************


****************************** Occupation ******************************
Number of unique values in 'Occupation': 4
Unique values in 'Occupation': ['Student' 'Employee' 'Self Employeed' 'House
wife']
**************************************************


****************************** Monthly Income ******************************
Number of unique values in 'Monthly Income': 5
Unique values in 'Monthly Income': ['No Income' 'Below Rs.10000' 'More than
50000' '10001 to 25000'
 '25001 to 50000']
**************************************************


****************************** Educational Qualifications ******************
************
Number of unique values in 'Educational Qualifications': 5
Unique values in 'Educational Qualifications': ['Post Graduate' 'Graduate'
'Ph.D' 'Uneducated' 'School']
**************************************************


****************************** Family size ******************************
Number of unique values in 'Family size': 6
Unique values in 'Family size': [4 3 6 2 5 1]
**************************************************


****************************** latitude ******************************
Number of unique values in 'latitude': 77
Unique values in 'latitude': [12.9766 12.977  12.9551 12.9473 12.985  12.929
9 12.9828 12.9854 12.8988
 12.9438 12.8893 12.9783 12.982  13.0298 12.9983 12.9925 12.9306 12.9353
 12.9155 13.0019 12.9698 12.9261 12.9119 12.9662 12.9565 13.0206 12.9635
 13.0067 12.8845 13.0158 12.9343 13.0012 12.9442 13.0487 12.9889 12.9335
 13.102  12.9048 12.9337 12.9037 13.0289 12.9561 12.9579 13.014  13.0138
 12.9537 12.998  13.0496 13.0166 13.0503 12.9883 13.0626 12.957  12.8652
```

```
 12.9757 12.9621 12.9217 13.0223 13.0262 13.0078 12.9105 12.8834 12.9149
 12.9706 13.0103 13.0641 12.9369 13.0809 12.9859 12.9866 12.9847 12.989
 12.9251 12.9967 13.0734 12.9515 12.9719]
**************************************************


****************************** longitude ******************************
Number of unique values in 'longitude': 76
Unique values in 'longitude': [77.5993 77.5773 77.6593 77.5616 77.5533 77.68
48 77.6131 77.7081 77.5764
 77.5738 77.6399 77.6408 77.6256 77.6047 77.6409 77.5633 77.5434 77.5585
 77.5135 77.5713 77.75   77.6221 77.6446 77.6068 77.5484 77.6479 77.5821
 77.545  77.6036 77.539  77.6044 77.5995 77.6076 77.5923 77.5741 77.5691
 77.5864 77.6821 77.59   77.5376 77.54   77.5921 77.6309 77.5658 77.5877
 77.6176 77.6227 77.4941 77.6804 77.5529 77.5987 77.5284 77.5637 77.524
 77.5586 77.5936 77.7132 77.62   77.5577 77.4842 77.5486 77.5635 77.6529
 77.5796 77.5931 77.6407 77.5565 77.6713 77.4904 77.5491 77.5332 77.4992
 77.7582 77.5464 77.4921 77.5128]
**************************************************


****************************** Pin code ******************************
Number of unique values in 'Pin code': 77
Unique values in 'Pin code': [560001 560009 560017 560019 560010 560103 5600
42 560048 560078 560004
 560068 560038 560008 560032 560033 560021 560085 560050 560098 560003
 560066 560034 560102 560025 560026 560043 560002 560086 560076 560096
 560029 560046 560030 560024 560020 560028 560064 560036 560011 560061
 560022 560027 560007 560012 560006 560047 560005 560073 560016 560013
 560051 560015 560018 560109 560023 560104 560041 560049 560045 560055
 560060 560062 560070 560075 560080 560092 560095 560097 560093 560091
 560100 560079 560059 560067 560014 560056 560072]
**************************************************


****************************** Output ******************************
Number of unique values in 'Output': 2
Unique values in 'Output': ['Yes' 'No']
**************************************************


****************************** Feedback ******************************
Number of unique values in 'Feedback': 2
Unique values in 'Feedback': ['Positive' 'Negative ']
**************************************************


****************************** Unnamed: 12 ******************************
Number of unique values in 'Unnamed: 12': 2
Unique values in 'Unnamed: 12': ['Yes' 'No']
**************************************************
```

In [29]:
```python
bal_data=food_df['Age']
# calculate the 1st and 3rd quartile
```

```python
q1=round(np.quantile(bal_data,0.25),2)
q3=round(np.quantile(bal_data, 0.75),2)
# Compute the IQR and the lower and upper bounds
IQR=q3-q1
lb=q1-1.5*IQR
ub=q3+1.5*IQR
con1=food_df['Age']>lb
con2=food_df['Age']<ub
con3=con1&con2
count=len(food_df[con3])
non_outliers_data=food_df[con3]
non_outliers_data
```

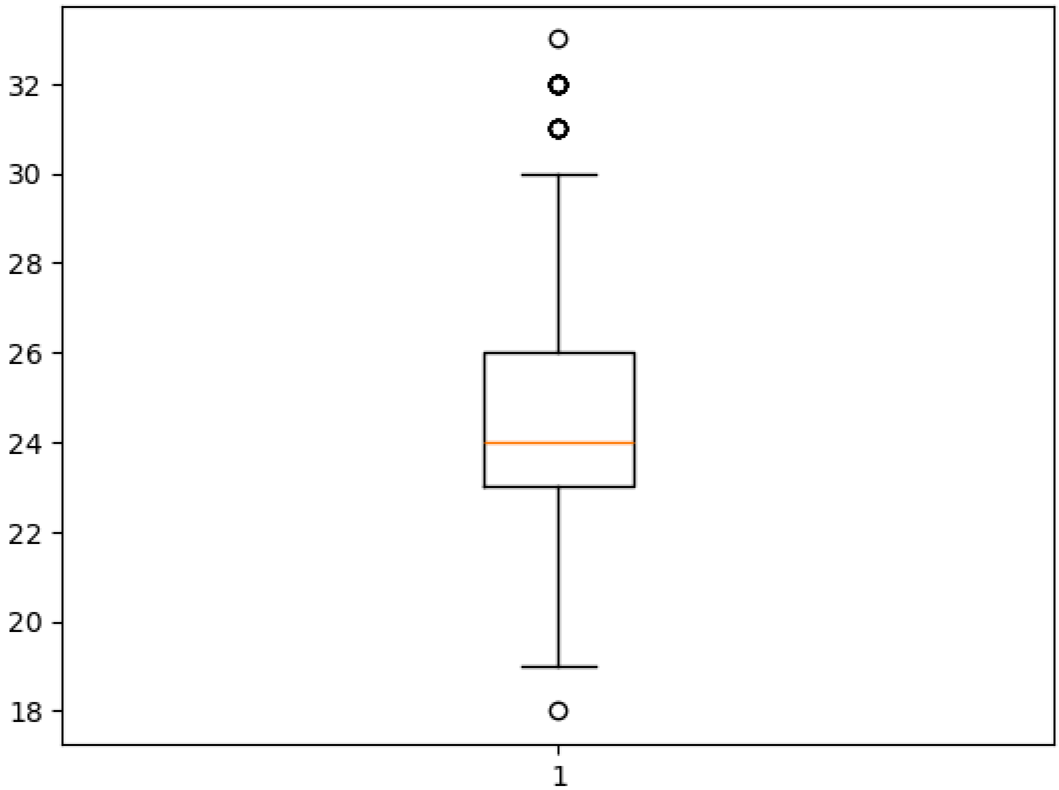
Out[29]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitu
0	20	Female	Single	Student	No Income	Post Graduate	4	12.9
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	12.9
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	12.9
3	22	Female	Single	Student	No Income	Graduate	6	12.9
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	12.9
...	...	...	...	...	...	...	...	
352	29	Female	Married	Employee	25001 to 50000	Graduate	4	12.9
355	21	Male	Single	Student	No Income	Graduate	2	13.0
369	30	Male	Married	Employee	More than 50000	Post Graduate	6	12.9
374	21	Male	Single	Student	No Income	Graduate	3	13.0
386	23	Male	Single	Student	Below Rs.10000	Post Graduate	2	12.9

264 rows × 13 columns

In [30]:

```python
plt.boxplot(food_df['Age'])
plt.show()
```

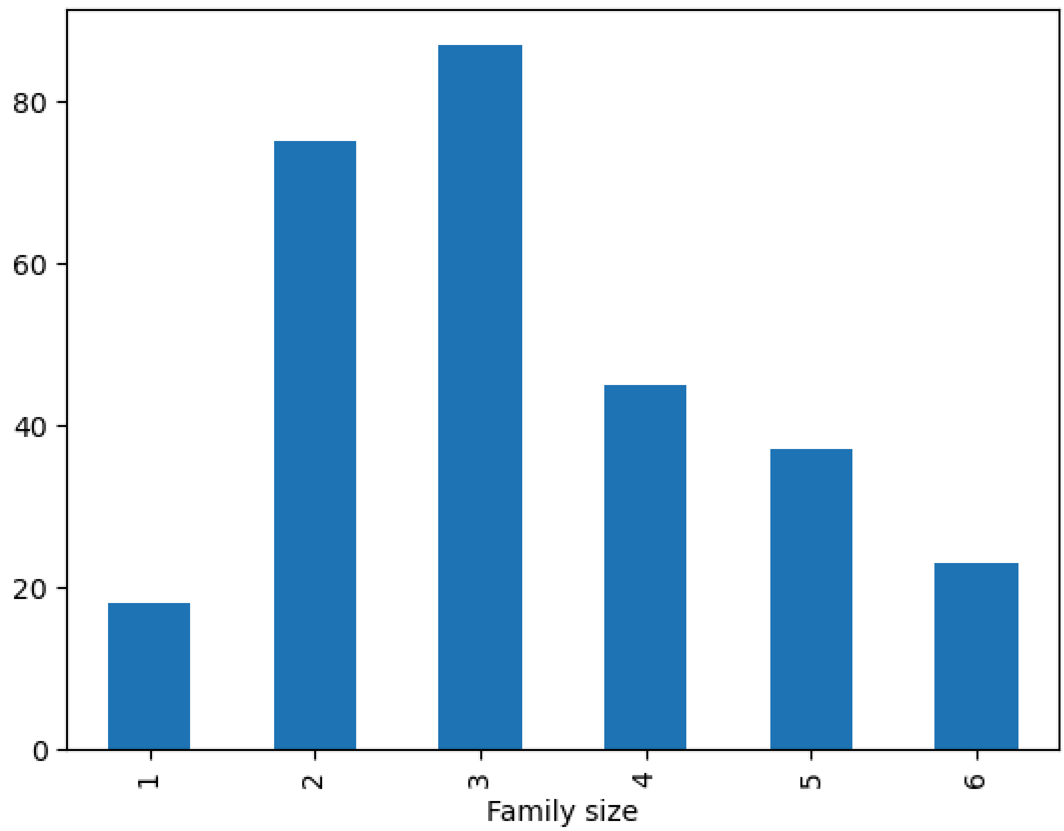


In [2]:
```
for i in numerical:
    plt.boxplot(food_df[i])
    plt.show()
```

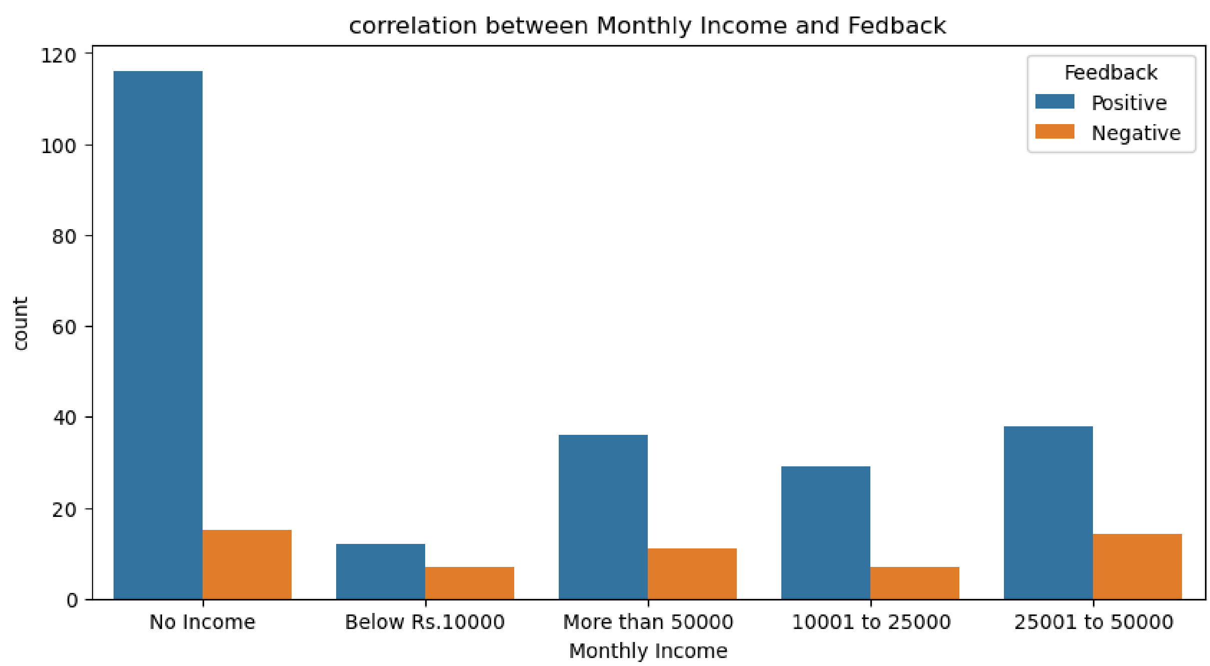
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 for i in numerical:
      2     plt.boxplot(food_df[i])
      3     plt.show()

NameError: name 'numerical' is not defined
```

In [32]:
```
food_df["Family size"].value_counts().sort_index().plot(kind= "bar")
```

Out[32]:
```
<Axes: xlabel='Family size'>
```



In [33]:
```
plt.figure(figsize=(10,5))
sns.countplot(data = food_df ,x= "Monthly Income" , hue ="Feedback")
plt.title("correlation between Monthly Income and Fedback")
plt.show()
```



In [34]:
```
# food_df["Target"] =  food_df.loc[:,"Unnamed: 12"]
food_df.drop("Unnamed: 12", axis=1 , inplace=True)
```

In [35]:
```
food_df.head()
```

Out[35]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	latitude
0	20	Female	Single	Student	No Income	Post Graduate	4	12.9766
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	12.9770
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	12.9551
3	22	Female	Single	Student	No Income	Graduate	6	12.9473
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	12.9850

In [36]:
```
food_df.drop("latitude", axis=1 , inplace=True)
food_df.drop("longitude", axis=1 , inplace=True)
```

In [37]:
```
# lat_yes=food_df[food_df.Target == "Yes"  ]["latitude"]
# lat_no =food_df[food_df.Target == "No" ] ["latitude"]

# lo_yes=food_df[food_df.Target == "Yes"  ]["longitude"]
# lo_no =food_df[food_df.Target == "No" ] ["longitude"]
```

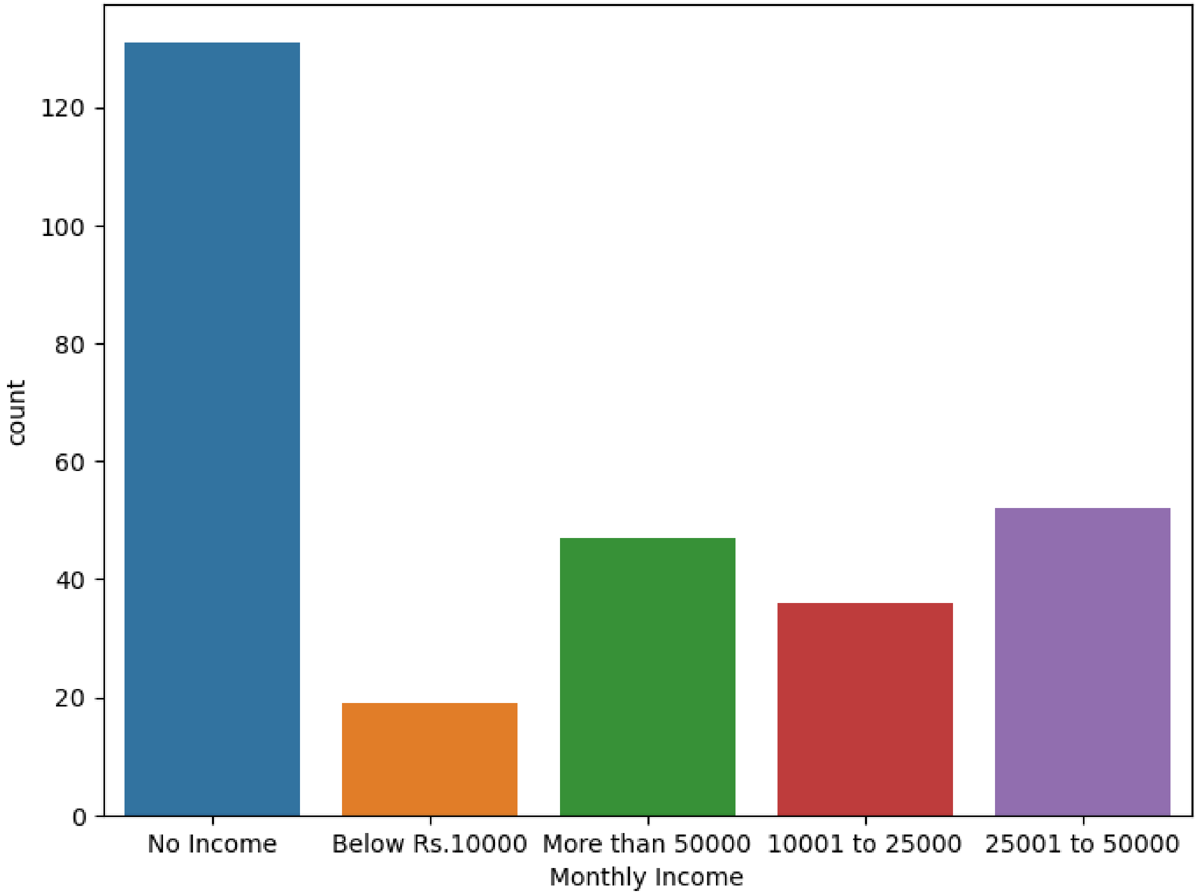
In [38]:
```
# sns.scatterplot(data = food_df ,x="latitude" , y="longitude"  ,hue='Target
```

In [39]:
```
# sns.histplot(data= food_df  , x='Age' ,hue="Target",cumulative=False)
```

In [40]:
```
# food_df.drop(columns= ["Target"] ,inplace=True )
```

In [41]:
```
plt.figure(figsize=(8,6))
sns.countplot(x= food_df["Monthly Income"])
```

Out[41]:
```
<Axes: xlabel='Monthly Income', ylabel='count'>
```



Data Prepartion

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

Out[42]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	Pin code
0	20	Female	Single	Student	No Income	Post Graduate	4	560001
1	24	Female	Single	Student	Below Rs.10000	Graduate	3	560009
2	22	Male	Single	Student	Below Rs.10000	Post Graduate	3	560017
3	22	Female	Single	Student	No Income	Graduate	6	560019
4	22	Male	Single	Student	Below Rs.10000	Post Graduate	4	560010

Spiltting Data into x & Y

In [43]:
```
x= food_df.drop("Output" , axis =1)
```

```
y= food_df["Output"]
```

Label encoding on categorical columns on x & y

In [44]:
```
# performing label encoding on categorical columns
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=pd.DataFrame(le.fit_transform(y))

for i in x.columns:
    if i!= 'Age':
        x[i]=le.fit_transform(x[i])
    else:
        continue
```

In [45]:
```
x.head(1)
```

Out[45]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size	Pin code	F
0	20	0	2	3	4	2	3	0	

In [46]:
```
y.head(1)
```

Out[46]:

	0
0	1

In [58]:
```
# ordinary_data  =['Monthly Income','Educational Qualifications']
```

In [59]:
```
# cat1_=['Gender','Marital Status', 'Feedback', 'Occupation']
```

In [61]:
```
# lb= LabelEncoder()
# for col in ordinary_data:
#     x[col]=lb.fit_transform(x[col])
```

In [67]:
```
# x.info()
```

In [ ]:
```
#x=pd.get_dummies(x,drop_first=True)
```

In [ ]:
```

```

Spiltting x & y into x_train ,x_test , y_train , y_test

In [47]:
```
# splitting data into train test
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
```

standard scaling on numerical column in x_train

In [ ]:
```
# standard scaling on numerical column in x_train
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train['Age']=(ss.fit_transform(x_train[['Age']]))
```

standard scaling on numerical column in x_test

In [70]:
```
# standard scaling on numerical column in x_test
x_test['Age']=(ss.fit_transform(x_test[['Age']]))
```

In [71]:
```
x_train.head()
```

Out[71]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size
65	0.124063	1	2	3	4	2	5
101	-0.533758	1	2	3	4	2	1
386	-0.533758	1	2	3	2	2	1
73	-0.533758	1	2	3	4	2	1
288	0.124063	0	2	3	4	2	2

In [72]:
```
x_test.head()
```

Out[72]:

	Age	Gender	Marital Status	Occupation	Monthly Income	Educational Qualifications	Family size
69	-0.297560	0	0	0	3	1	3
239	-0.630126	0	2	3	4	2	2
62	-0.630126	1	2	3	1	2	0
224	0.035007	0	0	3	4	2	1
131	2.362973	0	0	0	3	0	0

Modeling(Preprocessing)

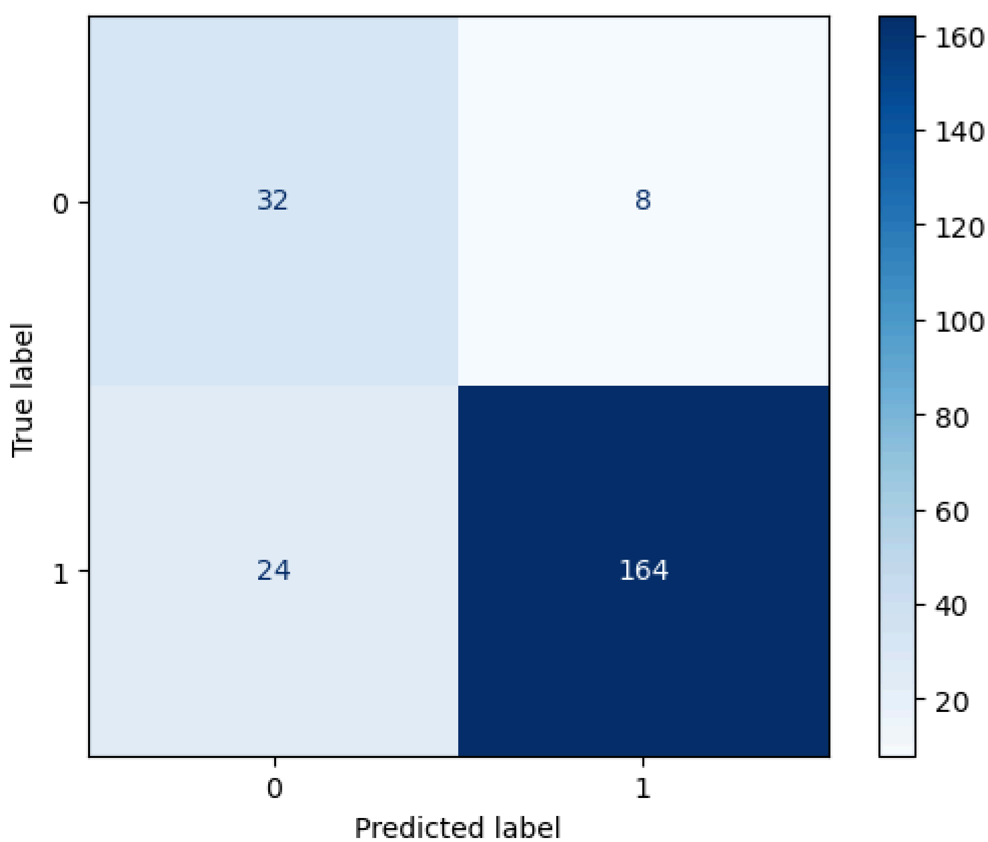
LogisticRegression

In [48]:
```
model1 =LogisticRegression()
model1.fit(x_train,y_train)
```

Out[48]:
```
▾ LogisticRegression
LogisticRegression()
```

In [74]:
```
y_pred_train = model1.predict(x_train)
cm = confusion_matrix(y_pred_train,y_train )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for train : " , recall_score(y_pred_train,y_train))
print("precision for train : " ,precision_score(y_pred_train,y_train))
print("f1_score for train : " ,f1_score(y_pred_train,y_train))
print("acc : " ,accuracy_score(y_pred_train,y_train))
disp.plot(cmap='Blues')

plt.show()
```

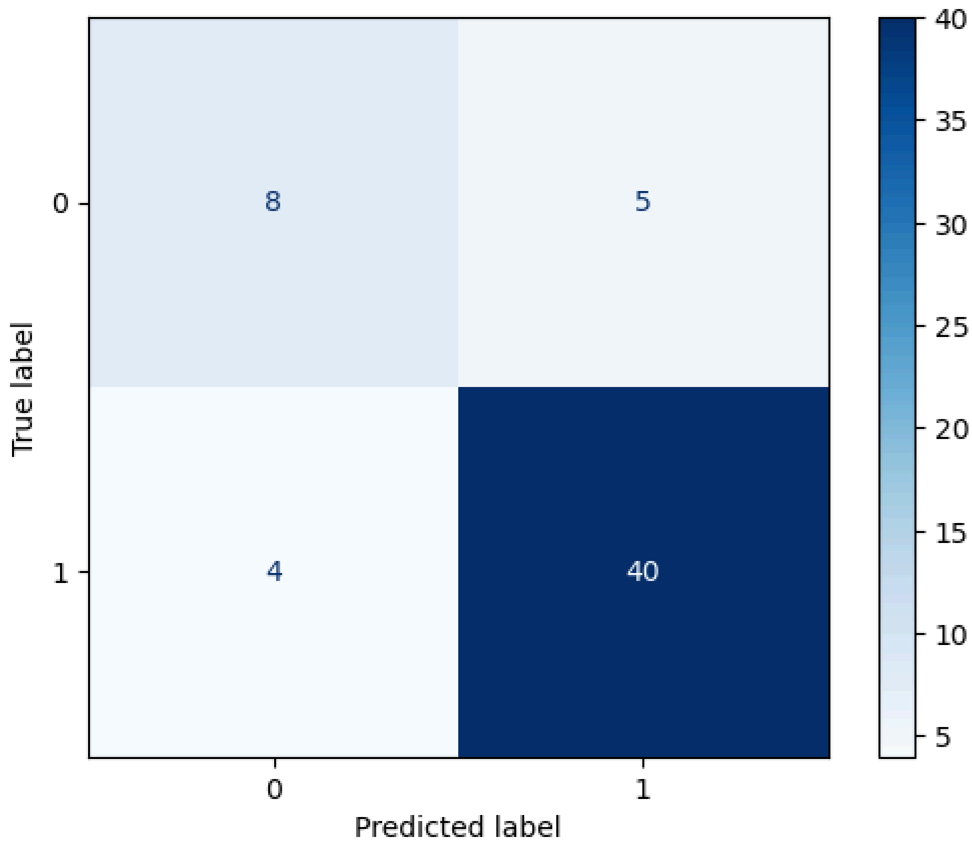
```
recall acc for train :  0.8723404255319149
precision for train :  0.9534883720930233
f1_score for train :  0.9111111111111112
acc :  0.8596491228070176
```



In [75]:
```
y_pred_test = model1.predict(x_test)
cm = confusion_matrix(y_pred_test,y_test )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
print("recall acc for train : " , recall_score(y_pred_test,y_test))
print("precision for train : " ,precision_score(y_pred_test,y_test))
print("f1_score for train : " ,f1_score(y_pred_test,y_test))
print("acc : " ,accuracy_score(y_pred_test,y_test))

disp.plot(cmap='Blues')

plt.show()
```

```
recall acc for train :  0.9090909090909091
precision for train :  0.8888888888888888
f1_score for train :  0.8988764044943819
acc :  0.8421052631578947
```



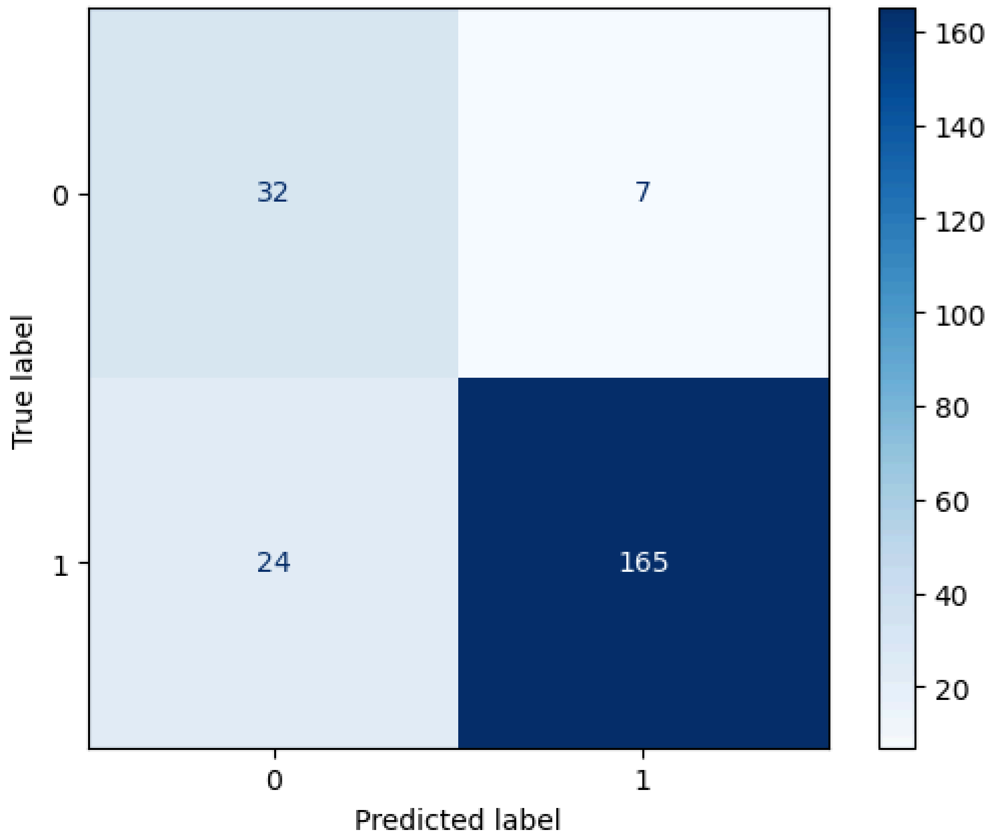
Decision Tree

In [76]:

```
#Decision Tree
model2 =DecisionTreeClassifier(max_depth=2,criterion='entropy')
model2.fit(x_train,y_train)

y_pred_train = model2.predict(x_train)
cm = confusion_matrix(y_pred_train,y_train )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for train : " , recall_score(y_pred_train,y_train))
print("precision for train : " ,precision_score(y_pred_train,y_train))
print("f1_score for train : " ,f1_score(y_pred_train,y_train))
```

```
print("acc : " ,accuracy_score(y_pred_train,y_train))
disp.plot(cmap='Blues')

plt.show()
```

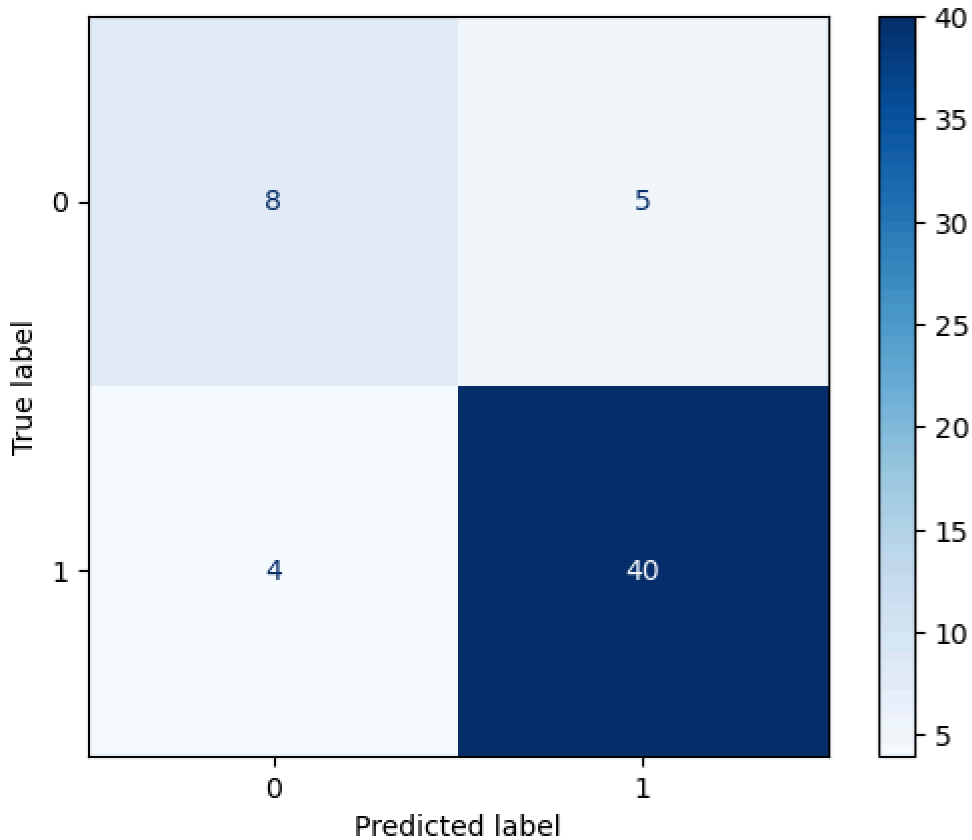
```
recall acc for train :  0.873015873015873
precision for train :  0.9593023255813954
f1_score for train :  0.9141274238227148
acc :  0.8640350877192983
```



In [78]:

```
y_pred_test = model1.predict(x_test)
cm = confusion_matrix(y_pred_test,y_test )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for train : " , recall_score(y_pred_test,y_test))
print("precision for train : " ,  precision_score(y_pred_test,y_test))
print("f1_score for train : " ,   f1_score(y_pred_test,y_test))
print("acc : " ,accuracy_score(y_pred_test,y_test))

disp.plot(cmap='Blues')

plt.show()
```

```
recall acc for train :  0.9090909090909091
precision for train :  0.8888888888888888
f1_score for train :  0.8988764044943819
acc :  0.8421052631578947
```



Random Forest

In [79]:
```
# Create Random Forest Classifier instance
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
rf_classifier.fit(x_train, y_train)
```

Out[79]:
```
▾         RandomForestClassifier
RandomForestClassifier(random_state=42)
```

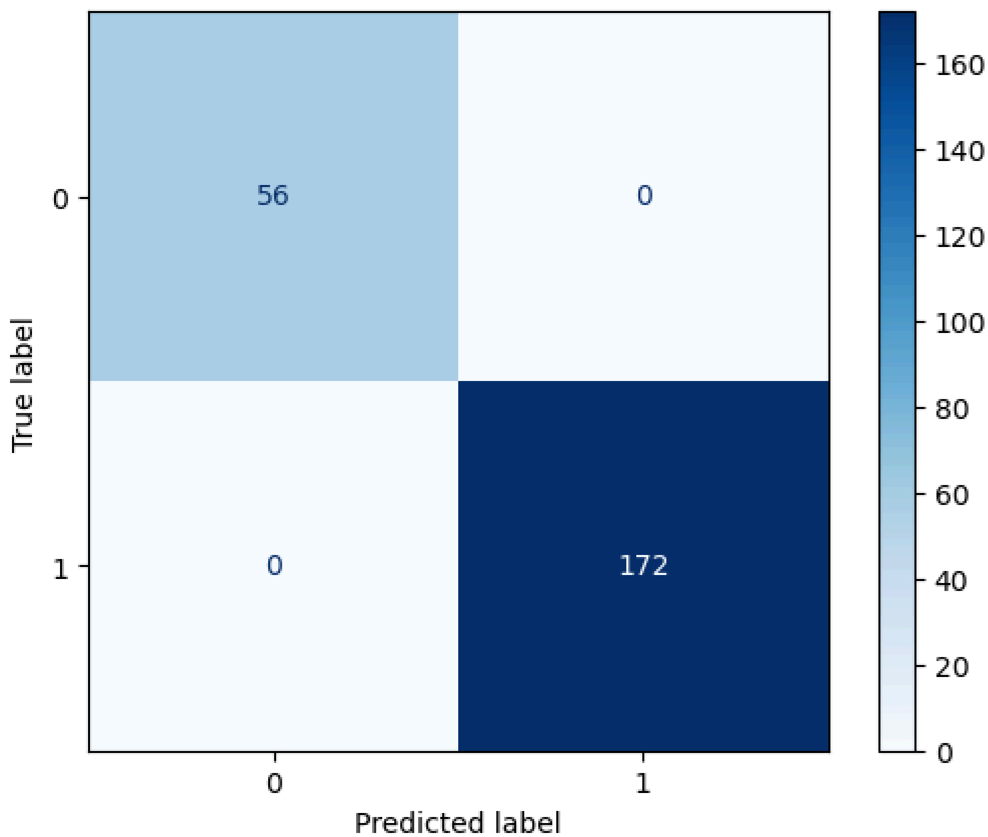
In [49]:
```
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
```

```
x_train shape: (228, 9)
y_train shape: (228, 1)
```

In [80]:
```
rf_classifier.fit(x_train,y_train)

y_pred_train = rf_classifier.predict(x_train)
cm = confusion_matrix(y_pred_train,y_train )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for train : " , recall_score(y_pred_train,y_train))
print("precision for train : " ,precision_score(y_pred_train,y_train))
```

```
print("f1_score for train : " ,f1_score(y_pred_train,y_train))
print("acc : " ,accuracy_score(y_pred_train,y_train))
disp.plot(cmap='Blues')

plt.show()
```

```
recall acc for train :  1.0
precision for train :  1.0
f1_score for train :  1.0
acc :  1.0
```

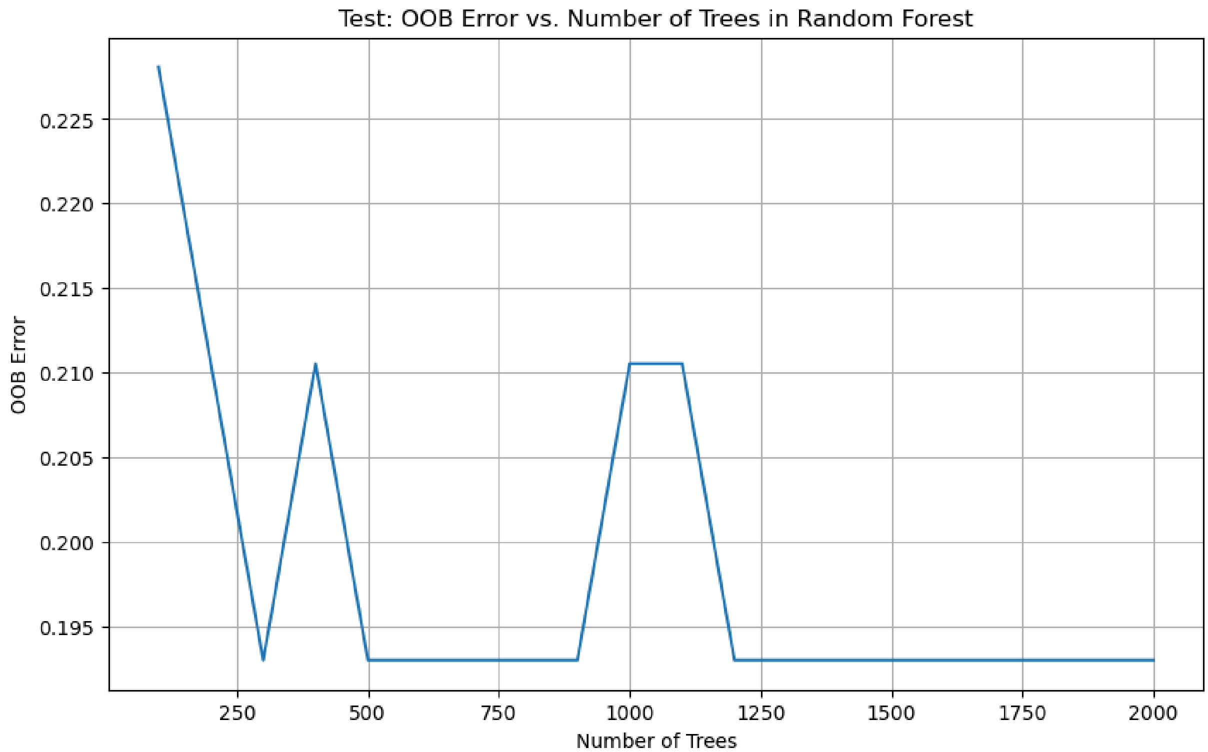


In [81]:

```
number_trees=[i for i in range(100,2100,100)]
oob_errors=[]

for i in number_trees:
    rfc=RandomForestClassifier(n_estimators=i, oob_score=True, random_state=
    rfc.fit(x_test,y_test)
    y_test_predict=rfc.predict(x_test)
    oob_errors.append(1 - rfc.oob_score_)
```

In [82]:

```
plt.figure(figsize=(10, 6))
plt.plot(number_trees, oob_errors)
plt.title("Test: OOB Error vs. Number of Trees in Random Forest")
plt.xlabel("Number of Trees")
plt.ylabel("OOB Error")
plt.grid()
plt.show()
```



In [56]:

```
import numpy as np
def calculate_acc(xtrain ,x_test ,y_train ,y_test):
    models =[LogisticRegression(),DecisionTreeClassifier(),RandomForestClass
    data_frame = pd.DataFrame()
    acc =[]
    recall =[]
    precision =[]
    f1=[]
    for mod in models :
        model_ = mod
        model_.fit(x_train ,y_train)

        y_pred_test =model_.predict(x_test)
        acc.append(np.round(accuracy_score(y_pred_test,y_test),2))
        recall.append(np.round(recall_score(y_pred_test,y_test),2))
        precision.append(precision_score(y_pred_test,y_test))
        f1.append(f1_score(y_pred_test,y_test).round(2))


    tabel =pd.DataFrame(index=["LogisticRegression","DecisionTreeClassifier"
                        columns=["acc" ,"recall","precision","F1"] )
    tabel["acc"]    = acc
    tabel["recall"] =recall
    tabel["precision"] = precision
    tabel["F1"] =f1
    return tabel
    print("Accuracy Measurement")
calculate_acc(x_train, x_test, y_train, y_test)
```

Out[56]:

	acc	recall	precision	F1
LogisticRegression	0.84	0.91	0.888889	0.90
DecisionTreeClassifier	0.74	0.86	0.800000	0.83
RandomForestClassifier	0.82	0.87	0.911111	0.89

In [11]:
```
df.head()
```

Out[11]:
```
0    20
1    24
2    22
3    22
4    22
Name: Age, dtype: int64
```

In [12]:
```
print(sw)
```

```
0.8098768724480562
```

In [ ]: