Problem 1: Linear Regression

Problem Statement: ¶

You are hired by a company Gem Stones co ltd, which is a cubic zirconia manufacturer. You are provided with the dataset containing the prices and other attributes of almost 27,000 cubic zirconia (which is an inexpensive diamond alternative with many of the same qualities as a diamond). The company is earning different profits on different prize slots. You have to help the company in predicting the price for the stone on the bases of the details given in the dataset so it can distinguish between higher profitable stones and lower profitable stones so as to have better profit share. Also, provide them with the best 5 attributes that are most important.

Data Dictionary:

1. Carat: Carat weight of the cubic zirconia.
2. Cut: Describe the cut quality of the cubic zirconia. Quality is increasing order Fair, Good, Very Good, Premium, Ideal.
3. Color: Colour of the cubic zirconia.With D being the best and J the worst.
4. Clarity: cubic zirconia Clarity refers to the absence of the Inclusions and Blemishes. (In order from Best to Worst, IF = flawless, I1= level 1 inclusion) IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1
5. Depth: The Height of cubic zirconia, measured from the Culet to the table, divided by its average Girdle Diameter.
6. Table: The Width of the cubic zirconia's Table expressed as a Percentage of its Average Diameter.
7. Price: The Price of the cubic zirconia.
8. X: Length of the cubic zirconia in mm.
9. Y: Width of the cubic zirconia in mm.
10. Z: Height of the cubic zirconia in mm.

In [1]:
```
# First Let's import all the required libraries and packages

import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
```

1.1. Read the data and do exploratory data analysis. Describe the data briefly. (Check the null values, Data types, shape, EDA). Perform Univariate and Bivariate Analysis.

In [2]:
```
#Reading the dataset

df = pd.read_csv('D:\\SHUBHANK !\\GL\\6. TOPIC 5 - Predictive Modeling\\Final
 Project\\cubic_zirconia.csv', index_col=[0])

#Checking the head of the dataset

df.head(10)
```

Out[2]:

	carat	cut	color	clarity	depth	table	x	y	z	price
1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779
6	1.02	Ideal	D	VS2	61.5	56.0	6.46	6.49	3.99	9502
7	1.01	Good	H	SI1	63.7	60.0	6.35	6.30	4.03	4836
8	0.50	Premium	E	SI1	61.5	62.0	5.09	5.06	3.12	1415
9	1.21	Good	H	SI1	63.8	64.0	6.72	6.63	4.26	5407
10	0.35	Ideal	F	VS2	60.5	57.0	4.52	4.60	2.76	706

In [3]:
```
#Checking the shape of the data

df.shape
```

Out[3]: (26967, 10)

In [4]:
```
#Describing the dataset

df.describe()
```

Out[4]:

	carat	depth	table	x	y	z	
count	26967.000000	26270.000000	26967.000000	26967.000000	26967.000000	26967.000000	269
mean	0.798375	61.745147	57.456080	5.729854	5.733569	3.538057	39
std	0.477745	1.412860	2.232068	1.128516	1.166058	0.720624	40
min	0.200000	50.800000	49.000000	0.000000	0.000000	0.000000	3
25%	0.400000	61.000000	56.000000	4.710000	4.710000	2.900000	9
50%	0.700000	61.800000	57.000000	5.690000	5.710000	3.520000	23
75%	1.050000	62.500000	59.000000	6.550000	6.540000	4.040000	53
max	4.500000	73.600000	79.000000	10.230000	58.900000	31.800000	188

We can see minimum price is 326 and maximum is 18818. maximum carat is 4.50 and minimum is 0.20.

In [5]:
```
#Let see some more info

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26967 entries, 1 to 26967
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    26967 non-null  float64
 1   cut      26967 non-null  object 
 2   color    26967 non-null  object 
 3   clarity  26967 non-null  object 
 4   depth    26270 non-null  float64
 5   table    26967 non-null  float64
 6   x        26967 non-null  float64
 7   y        26967 non-null  float64
 8   z        26967 non-null  float64
 9   price    26967 non-null  int64  
dtypes: float64(6), int64(1), object(3)
memory usage: 2.3+ MB
```

There are non-null values present only in depth column. There are 6 float types, 1 integer type and 3 object types columns in the dataset.

In [6]:
```
#What is the shape of the data?

df.shape
```

Out[6]:
```
(26967, 10)
```

In [7]:
```
#Are there any null values present ? Let see.

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

Out[7]:
```
carat        0
cut          0
color        0
clarity      0
depth      697
table        0
x            0
y            0
z            0
price        0
dtype: int64
```

697 null values are present in depth column.

In [8]:
```
#Let see if there are any duplicate rows in the dataset

df.duplicated().sum()
```

Out[8]:
```
34
```

In [9]:
```
#There are 34 duplicate records found. Let's remove it from the dataset.

df.drop_duplicates(inplace=True)
```

In [10]:
```
#Let's check the shape again after dropping the duplicates

df.shape
```

Out[10]:
```
(26933, 10)
```

In [11]:
```
#Let's check the data types

df.dtypes
```

Out[11]:
```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
x          float64
y          float64
z          float64
price        int64
dtype: object
```

In [12]:
```
#Now, let's check the object data types

df.select_dtypes(include='object').head()
```

Out[12]:

	cut	color	clarity
1	Ideal	E	SI1
2	Premium	G	IF
3	Very Good	E	VVS2
4	Ideal	F	VS1
5	Ideal	F	VVS1

cut, color and clarity are the columns with the object data type.

In [13]:
```
#Let get some more details about object data type columns:

print('Number of unique values in "cut" variable:',df['cut'].nunique())
print('Those unique values are:',df['cut'].unique())
print('\nValue Counts:\n', df.cut.value_counts())
print('_____________________________________________')
print('Number of unique values in "color" variable:',df['color'].nunique())
print('Those unique values are:',df['color'].unique())
print('\nValue Counts:\n', df.color.value_counts().sort_index(ascending=True))
print('_____________________________________________')
print('Number of unique values in "clarity" variable:',df['clarity'].nunique
())
print('Those unique values are:',df['clarity'].unique())
print('\nValue Counts:\n', df.clarity.value_counts())
```

```
Number of unique values in "cut" variable: 5
Those unique values are: ['Ideal' 'Premium' 'Very Good' 'Good' 'Fair']

Value Counts:
 Ideal        10805
Premium       6886
Very Good     6027
Good          2435
Fair           780
Name: cut, dtype: int64

_____________________________________________
Number of unique values in "color" variable: 7
Those unique values are: ['E' 'G' 'F' 'D' 'H' 'J' 'I']

Value Counts:
 D    3341
E    4916
F    4723
G    5653
H    4095
I    2765
J    1440
Name: color, dtype: int64

_____________________________________________
Number of unique values in "clarity" variable: 8
Those unique values are: ['SI1' 'IF' 'VVS2' 'VS1' 'VVS1' 'VS2' 'SI2' 'I1']

Value Counts:
 SI1     6565
VS2     6093
SI2     4564
VS1     4087
VVS2    2530
VVS1    1839
IF       891
I1       364
Name: clarity, dtype: int64
```

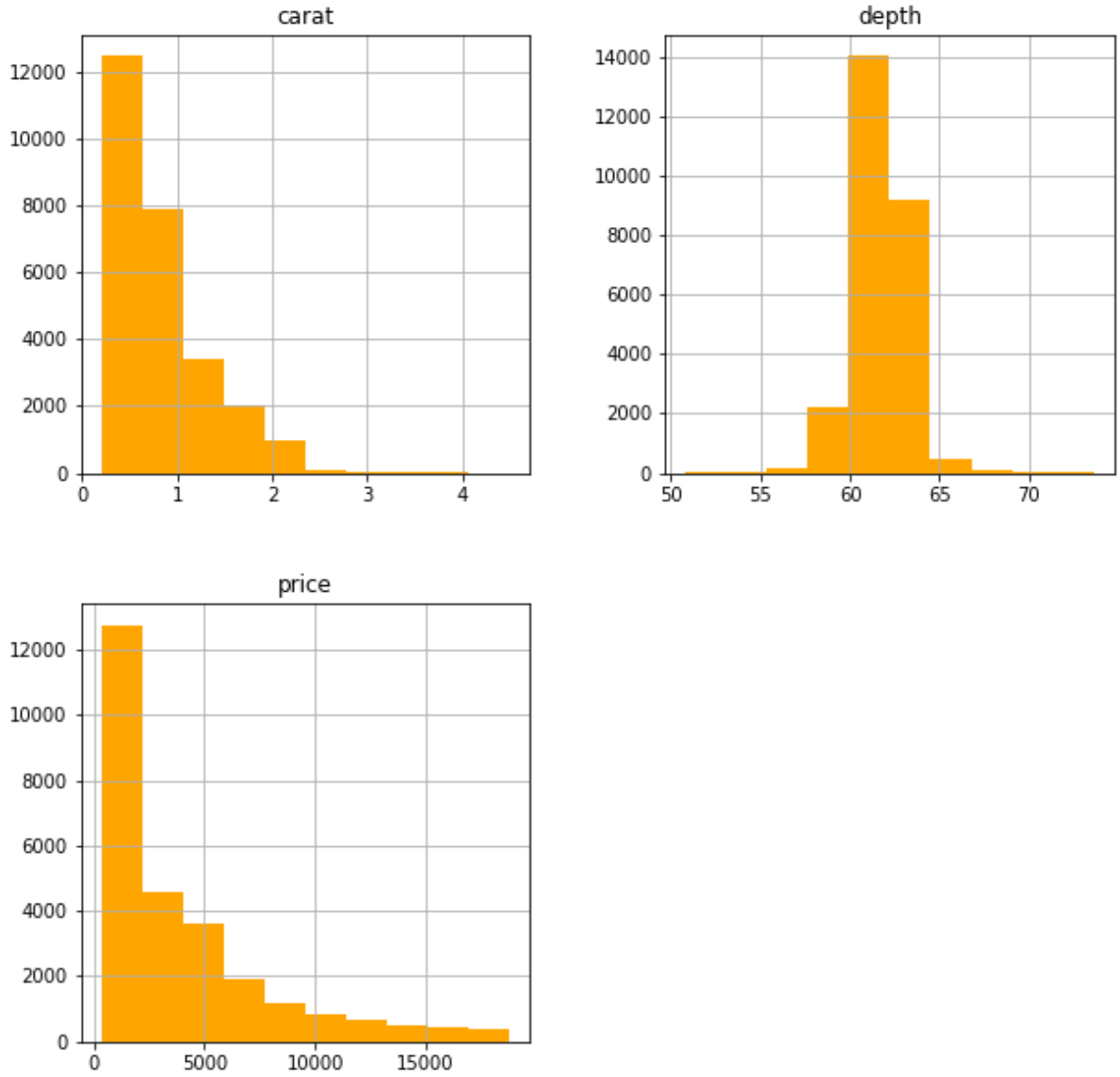
'Ideal' quality stones are in majority and 'Fair' quality stones are less.

Histogram:

In [14]:
```
#Creating the histogram for Carat, depth and price.

df[['carat','depth','price']].hist(figsize=(10,10), color='orange')
```

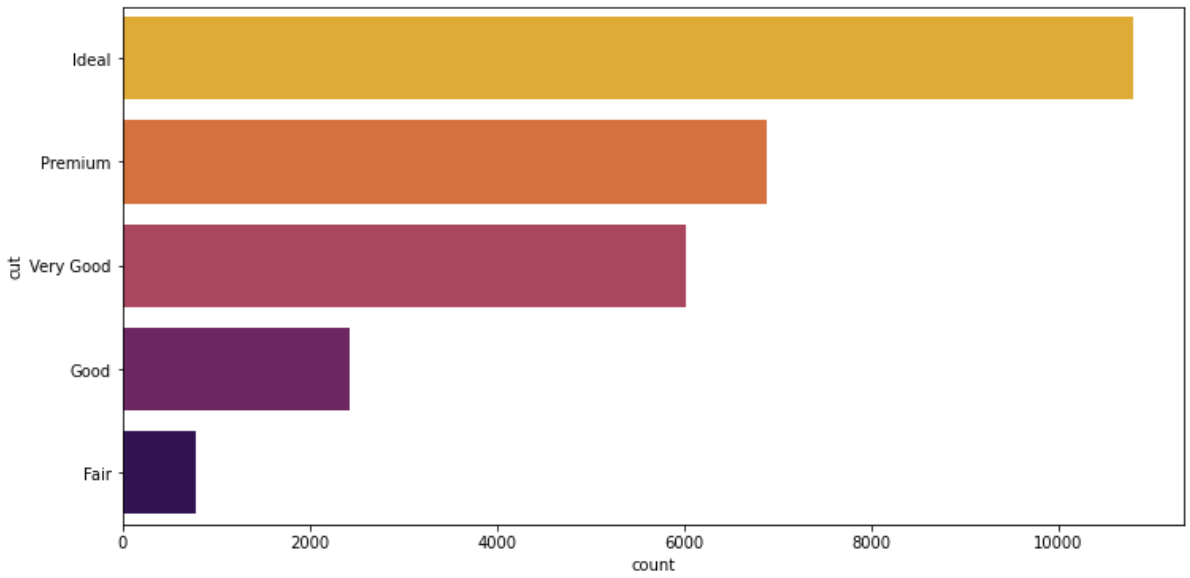
```
C:\Users\user\anaconda3\lib\site-packages\pandas\plotting\_matplotlib\tools.p
y:331: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be remove
d two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

Out[14]:
```
array([[<AxesSubplot:title={'center':'carat'}>,
        <AxesSubplot:title={'center':'depth'}>],
       [<AxesSubplot:title={'center':'price'}>, <AxesSubplot:>]],
      dtype=object)
```

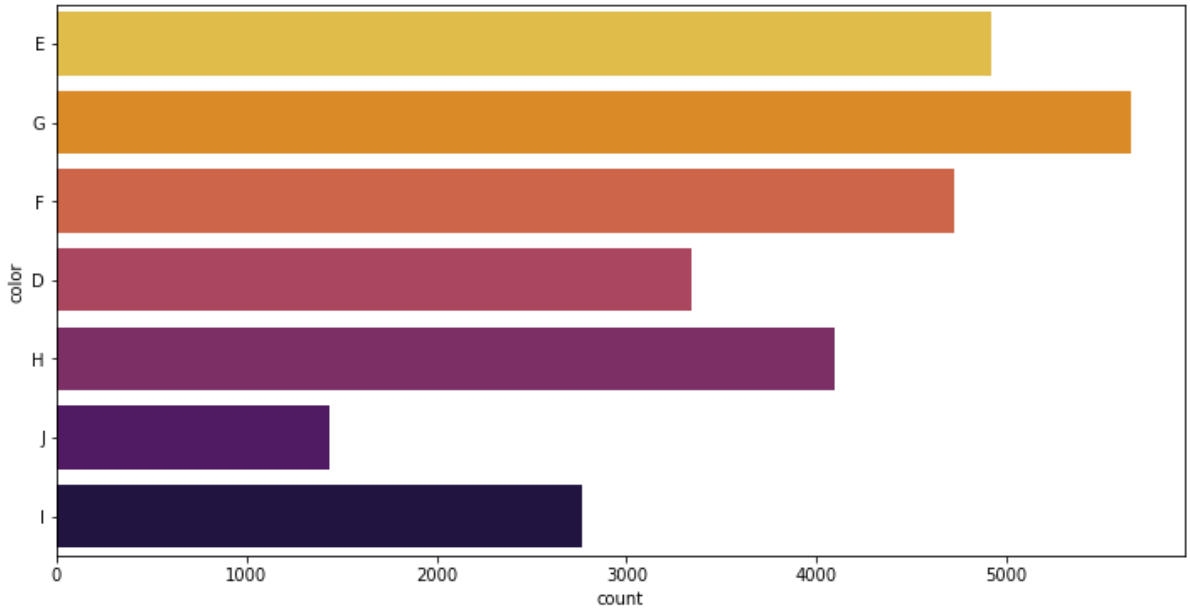


Countplot:

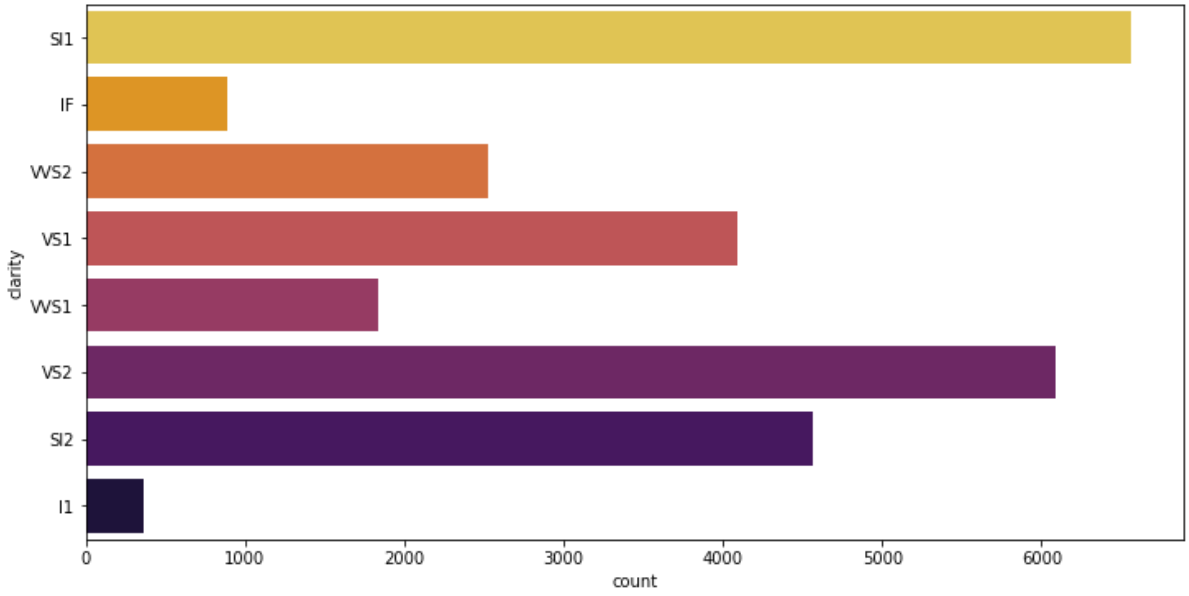
In [15]:
```
#Lets check the count of cut :

plt.figure(figsize=(12,6))
sns.countplot(y=(df.cut),palette='inferno_r');
```



In [16]:
```
#Lets check the count of different available colors :

plt.figure(figsize=(12,6))
sns.countplot(y=(df.color),palette='inferno_r');
```



In [17]:
```
##Lets check the count of different clarity :

plt.figure(figsize=(12,6))
sns.countplot(y=(df.clarity),palette='inferno_r');
```



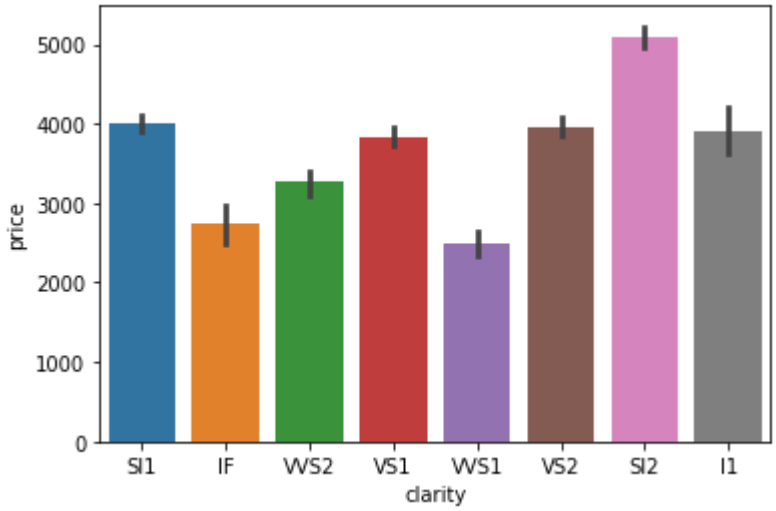
In [18]:
```
df.price.describe()
```

Out[18]:
```
count    26933.000000
mean      3937.526120
std       4022.551862
min        326.000000
25%        945.000000
50%       2375.000000
75%       5356.000000
max      18818.000000
Name: price, dtype: float64
```

Barplot

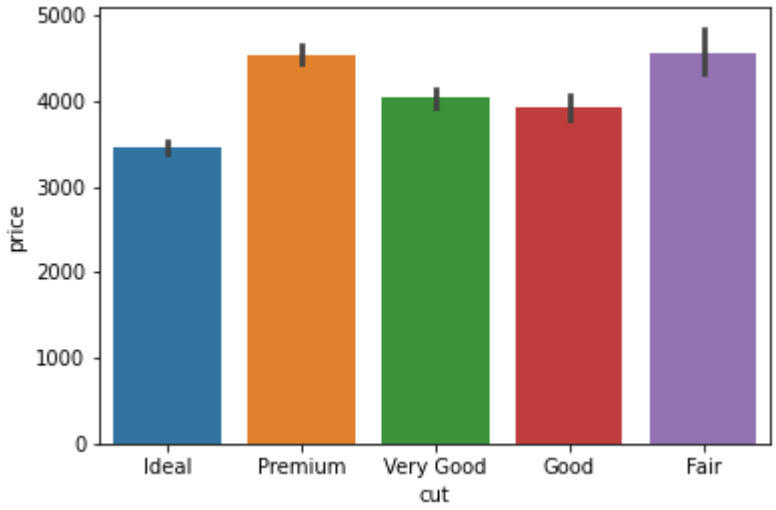
In [19]:
```
sns.barplot(data = df, x='clarity',y='price')
```

Out[19]: <AxesSubplot:xlabel='clarity', ylabel='price'>



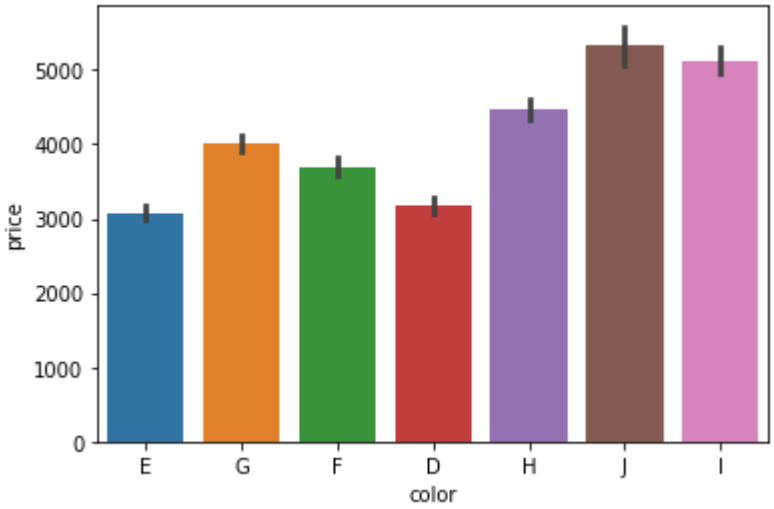
In [20]:
```
sns.barplot(data = df, x='cut',y='price')
```

Out[20]: <AxesSubplot:xlabel='cut', ylabel='price'>



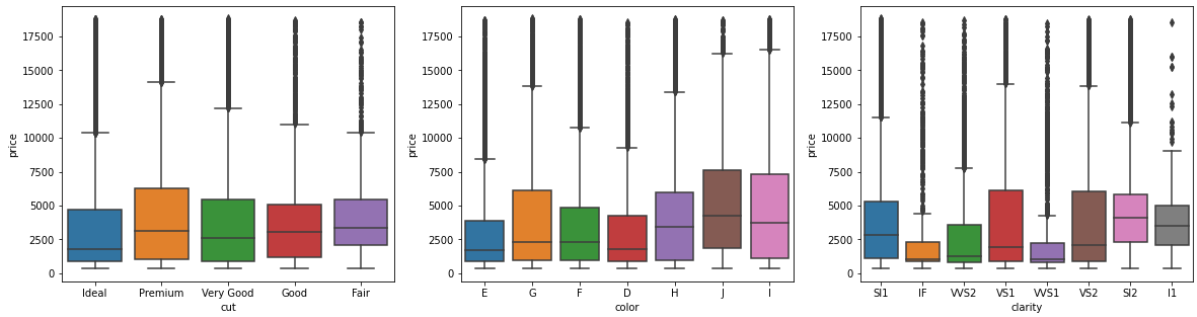
In [21]:
```
sns.barplot(data = df, x='color',y='price')
```

Out[21]: <AxesSubplot:xlabel='color', ylabel='price'>

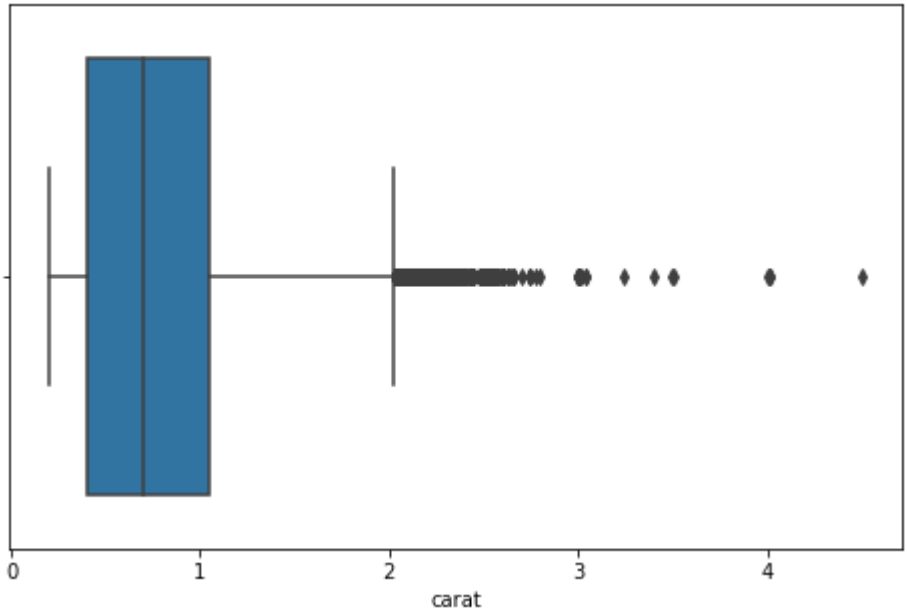


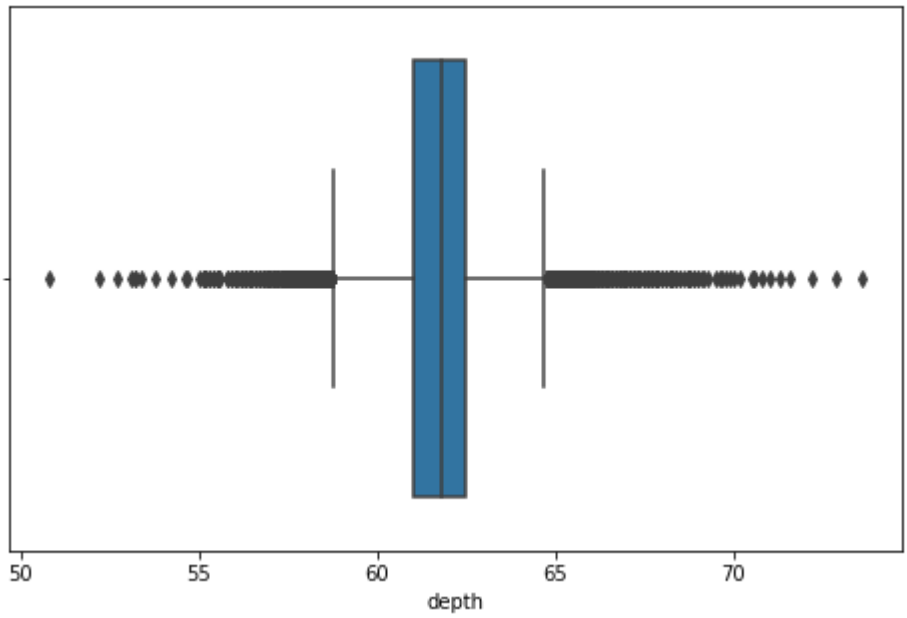
Boxplot

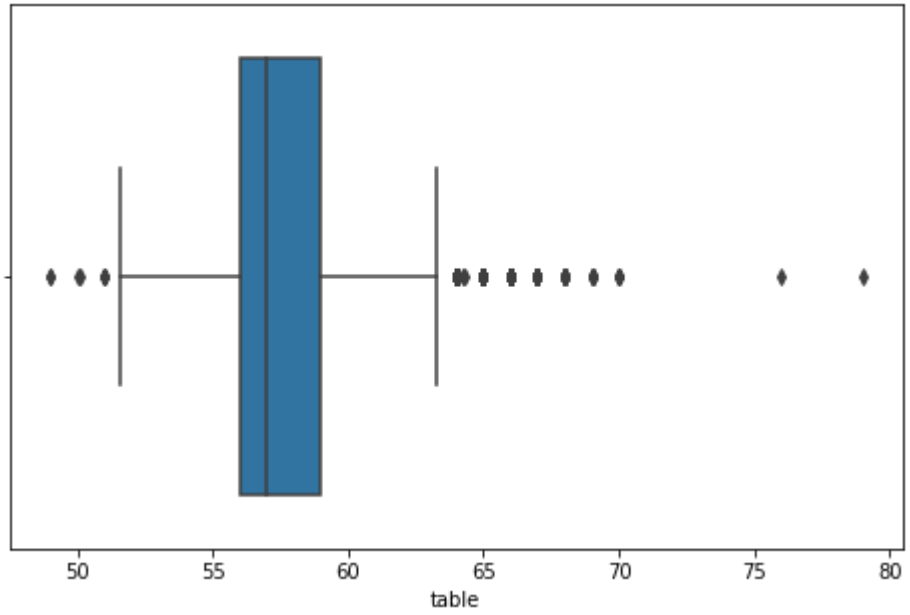
In [22]:
```
#Let's check the outliers in cut,color, clarity with respect to price variabl
e:

plt.rcParams['figure.figsize']=20,5
plt.subplot(131)
sns.boxplot(df['cut'], df['price'])

plt.subplot(132)
sns.boxplot(df['color'], df['price'])

plt.subplot(133)
sns.boxplot(df['clarity'], df['price'])
plt.show()
```

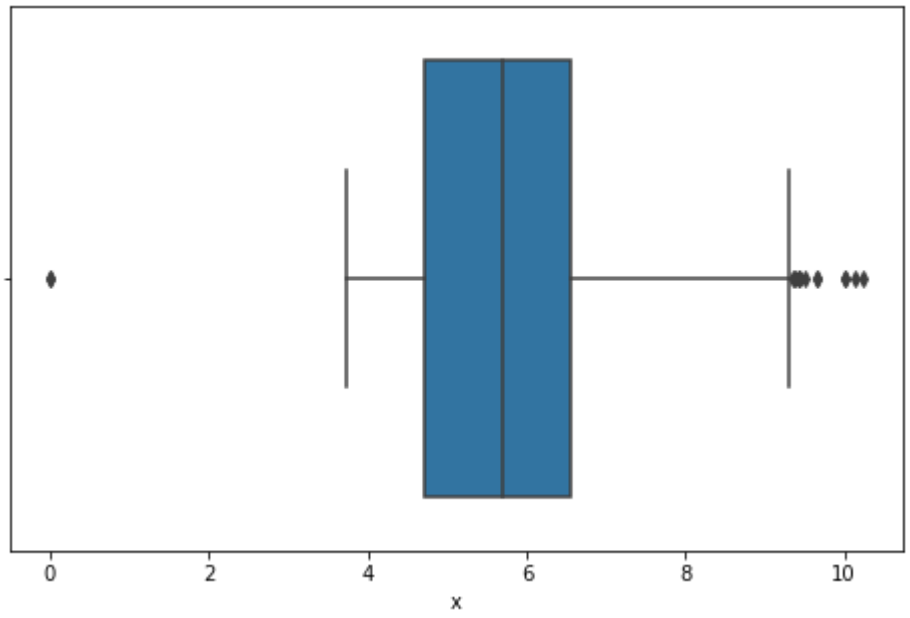


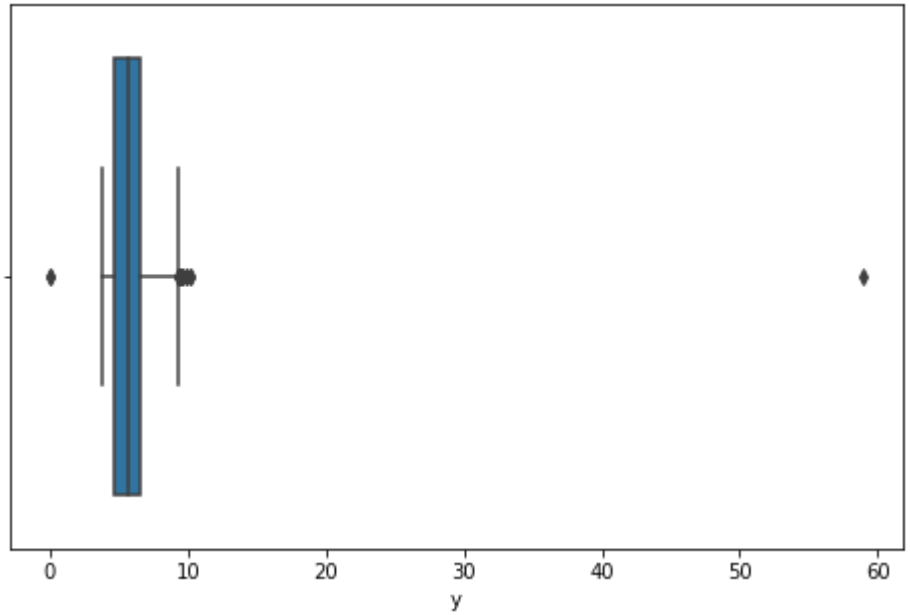
In [23]:
```
#Let's check the Outliers in all the features

columns = ['carat' ,'depth', 'table', 'x', 'y', 'z',
       'price']


for x in columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(df[x])
    plt.show()
```

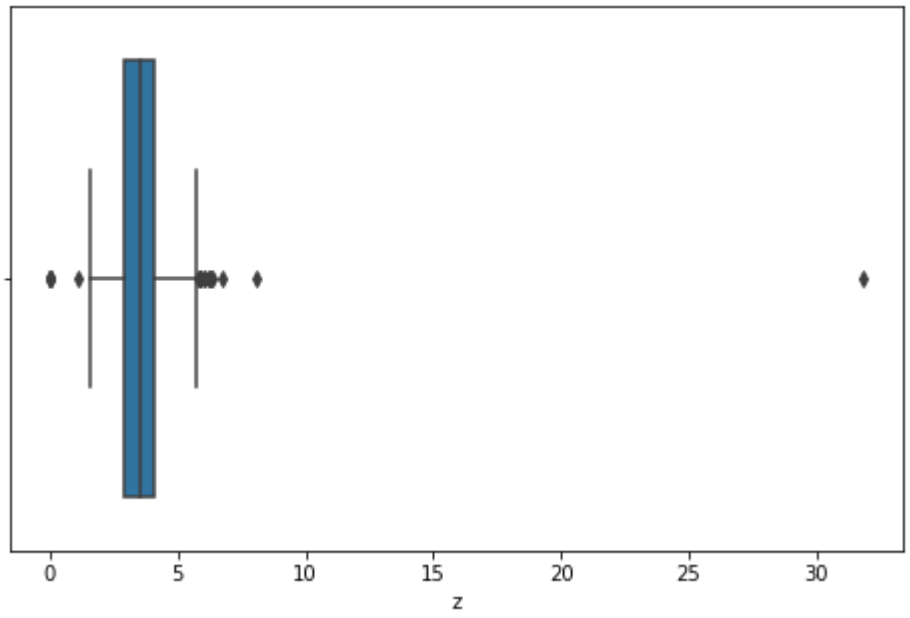

0
1
2
3
4
carat

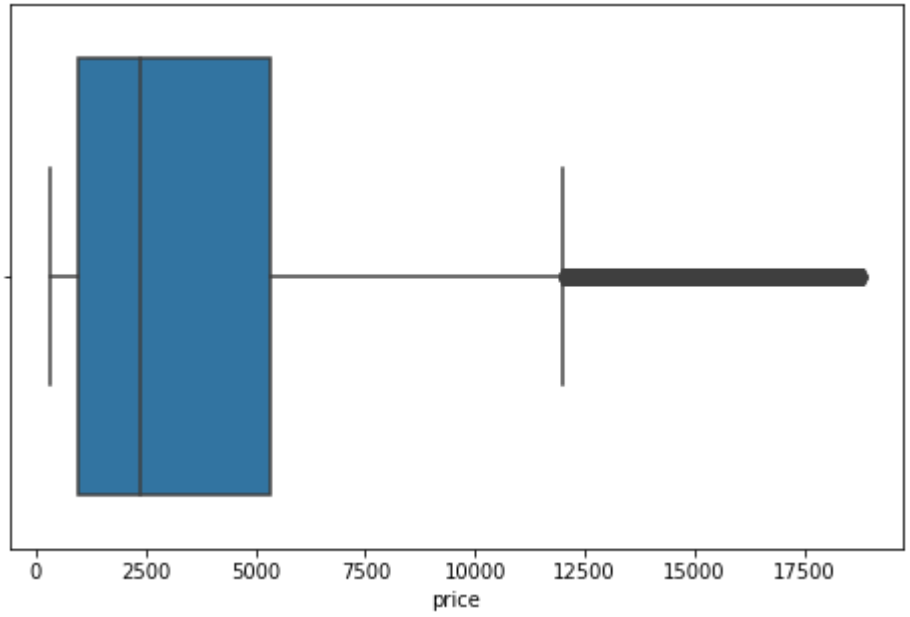

50
55
60
65
70
depth


50
55
60
65
70
75
80
table


0
2
4
6
8
10
x

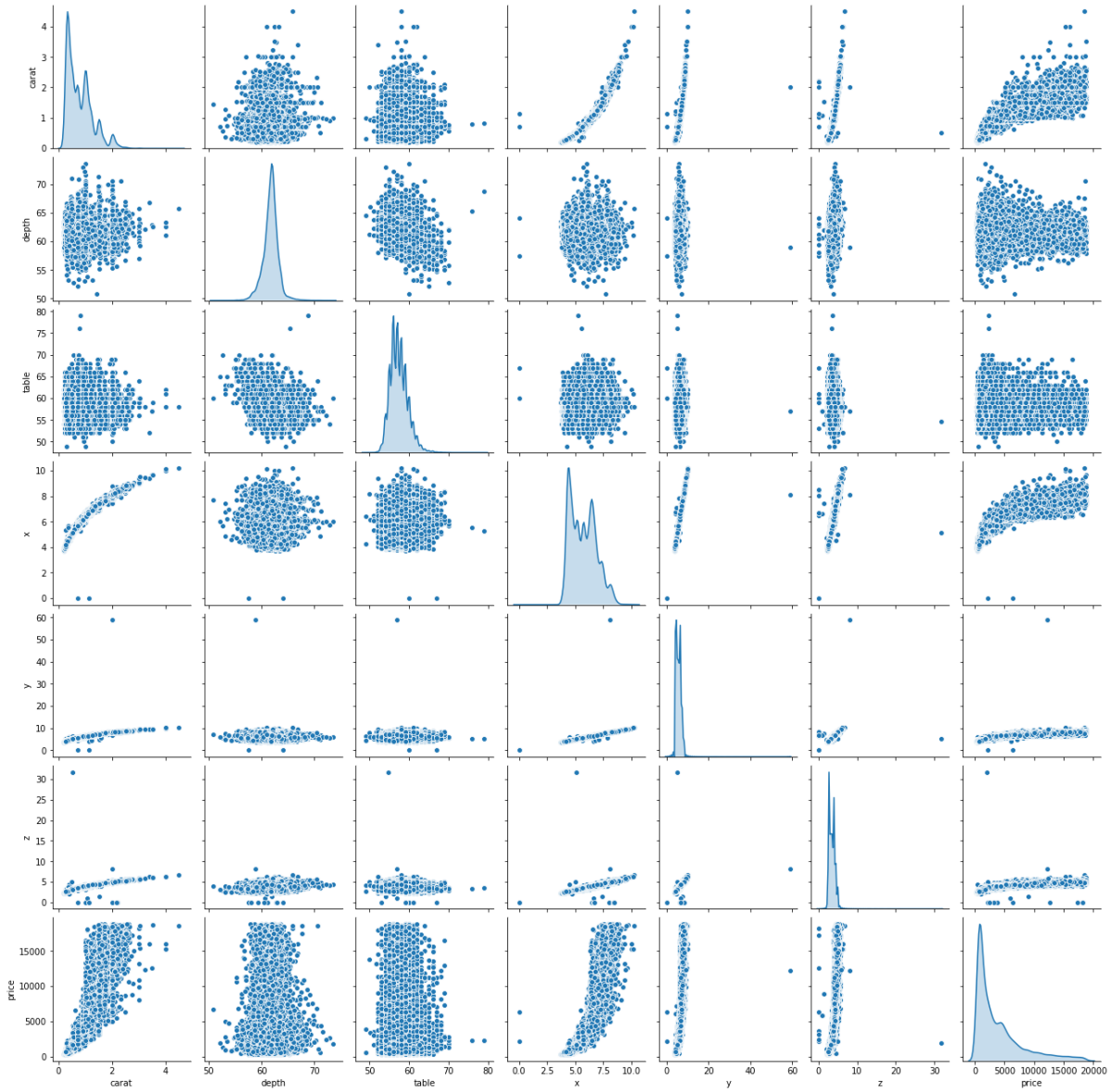

0
10
20
30
40
50
60
y


0
5
10
15
20
25
30
z



Pairplot

In [24]:
```
sns.pairplot(df, diag_kind='kde')
plt.show()
```



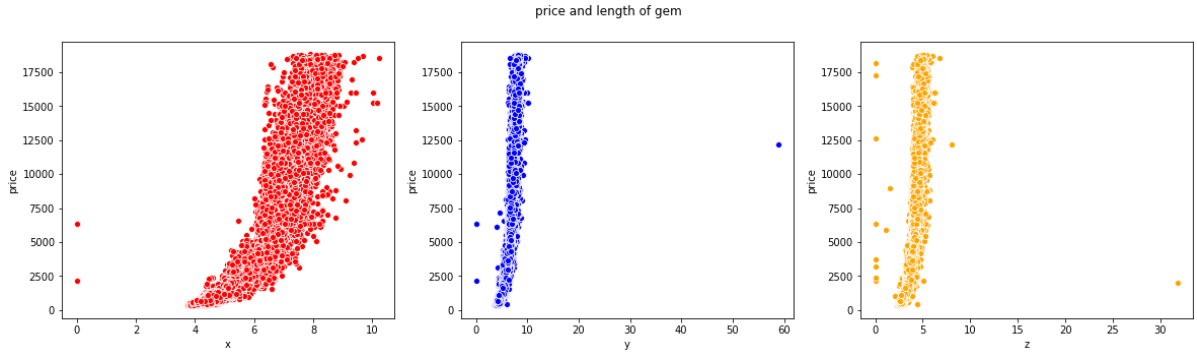
Correlation and Heatmap

In [25]:
```
corr = df.corr()
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True, fmt = '.2f')
```

Out[25]: <AxesSubplot:>

We have given that the X is length , Y is width and Z is height in mm, so let's check whether these factors affecting the price of the product or not.

In [26]:
```
#creating the Scatter plot to see the affect of the dimensions on the produc
t's price

plt.rcParams['figure.figsize']=20,5
plt.subplot(131)
sns.scatterplot(df['x'], df['price'], color='red')

plt.subplot(132)
sns.scatterplot(df['y'], df['price'], color='blue')

plt.subplot(133)
sns.scatterplot(df['z'], df['price'], color='orange')

plt.suptitle('price and length of gem')
plt.show()
```



We can see that dimensions does not affect much on price, so we can remove those columns

1.2 Impute null values if present, also check for the values which are equal to zero. Do they have any meaning or do we need to change them or drop them? Do you think scaling is necessary in this case?

In [27]:
```
df.isnull().sum()
```

Out[27]:
```
carat        0
cut          0
color        0
clarity      0
depth      697
table        0
x            0
y            0
z            0
price        0
dtype: int64
```

In [28]:
```
#Let's fill these 697 null place with mean value of depth:

df['depth'].fillna(df['depth'].mean(), inplace=True)

#Again Checking the info

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

Out[28]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
price      0
dtype: int64
```

Checking and dropping the zeros from X,Y,Z columns if available:

In [29]:
```
#Let's check the location of presence of zeros in x,y,z columns

df.loc[((df['x'] == 0) | (df['y'] == 0) | (df['z'] == 0))]
```

Out[29]:

	carat	cut	color	clarity	depth	table	x	y	z	price
5822	0.71	Good	F	SI2	64.1	60.0	0.00	0.00	0.0	2130
6035	2.02	Premium	H	VS2	62.7	53.0	8.02	7.95	0.0	18207
10828	2.20	Premium	H	SI1	61.2	59.0	8.42	8.37	0.0	17265
12499	2.18	Premium	H	SI2	59.4	61.0	8.49	8.45	0.0	12631
12690	1.10	Premium	G	SI2	63.0	59.0	6.50	6.47	0.0	3696
17507	1.14	Fair	G	VS1	57.5	67.0	0.00	0.00	0.0	6381
18195	1.01	Premium	H	I1	58.1	59.0	6.66	6.60	0.0	3167
23759	1.12	Premium	G	I1	60.4	59.0	6.71	6.67	0.0	2383

In [30]:
```
#Let's drop these rows where zeros are available

df.drop(df[((df['x'] == 0) | (df['y'] == 0) | (df['z'] == 0))].index, inplace=
True)
```

In [31]:
```
#Let's check again

df.loc[((df['x'] == 0) | (df['y'] == 0) | (df['z'] == 0))]
```

Out[31]:

carat	cut	color	clarity	depth	table	x	y	z	price

Scaling the data

In [32]:
```
#Using standard scaler:

int_type = df.select_dtypes(exclude=['object'])

std_sc = StandardScaler()

#Scaling the integer columnns by fit_transform
df[int_type.columns] = std_sc.fit_transform(int_type)
```

In [33]:
```
df.head()
```

Out[33]:

	carat	cut	color	clarity	depth	table	x	y	z	
1	-1.043484	Ideal	E	SI1	0.254366	0.244117	-1.296010	-1.240036	-1.224888	-0.85
2	-0.980601	Premium	G	IF	-0.678601	0.244117	-1.162802	-1.093962	-1.169137	-0.73
3	0.214178	Very Good	E	VVS2	0.326132	1.140461	0.275843	0.332401	0.336153	0.58
4	-0.791951	Ideal	F	VS1	-0.104468	-0.652227	-0.807581	-0.801816	-0.806752	-0.70
5	-1.022523	Ideal	F	VVS1	-0.965668	0.692289	-1.224966	-1.119740	-1.238826	-0.78

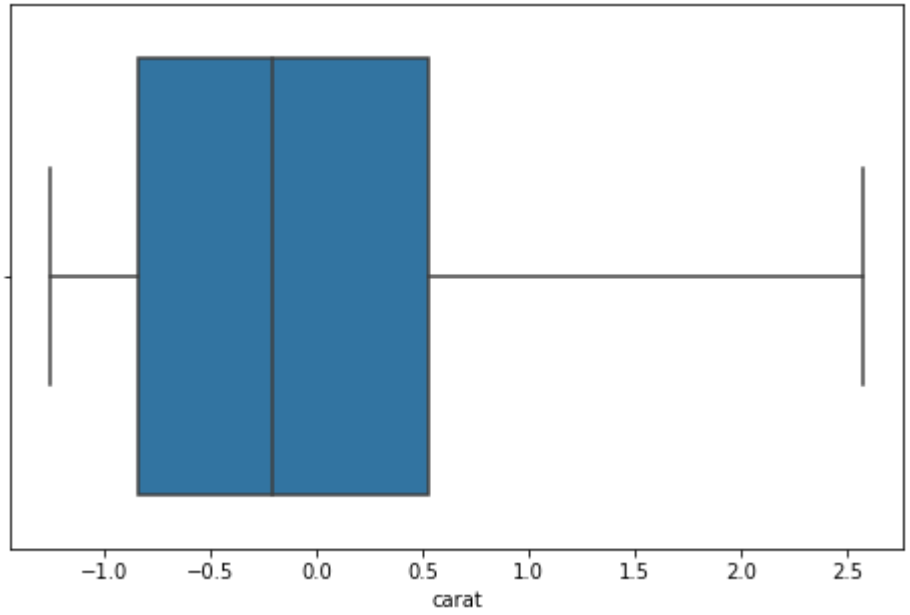
Removing the outliers:

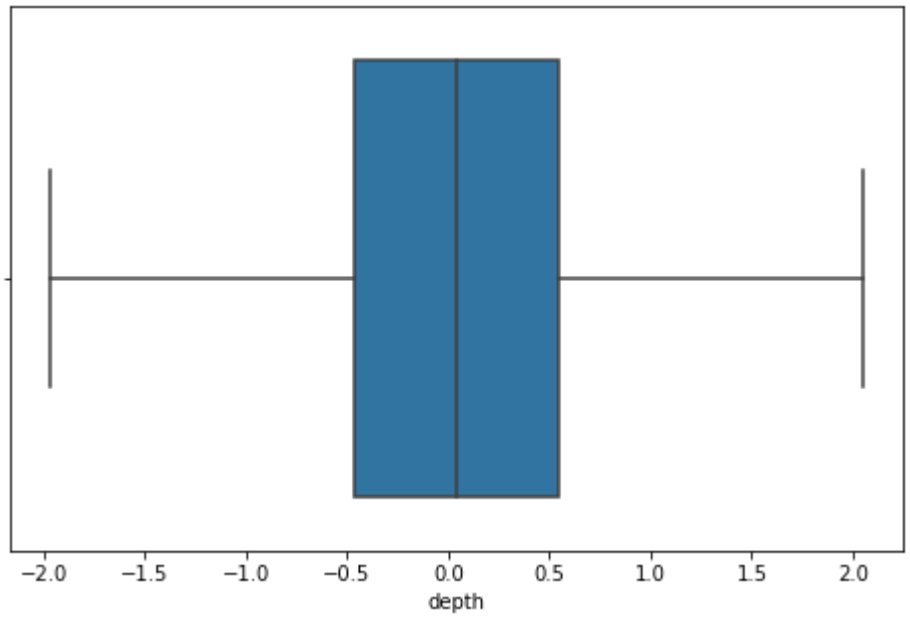
In [34]:
```
#Seperating the continuous variables

continuous_variables = df.dtypes[(df.dtypes!='uint8') & (df.dtypes!='object')]
.index
```

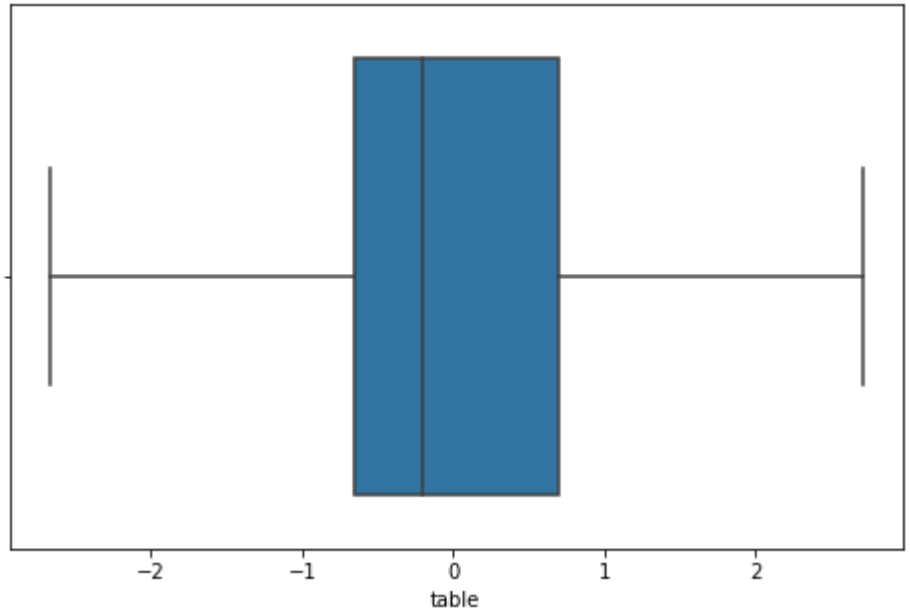
In [35]:
```
#Defining a function named treat_outliers to removing all the outliers from th
e continuos variables:

def treat_outliers(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

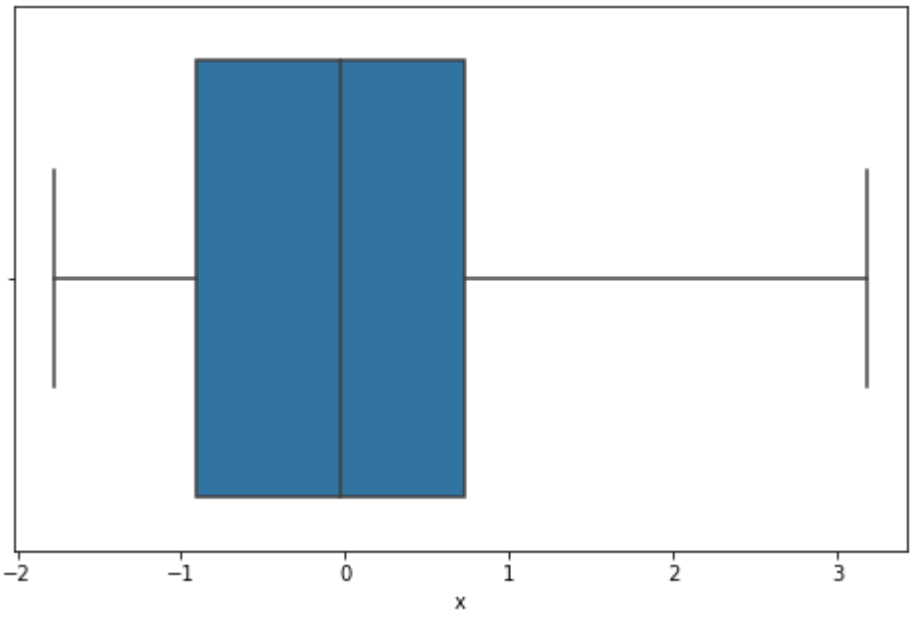
In [36]:
```
for column in df[continuous_variables].columns:
    lr,ur=treat_outliers(df[column])
    df[column]=np.where(df[column]>ur,ur,df[column])
    df[column]=np.where(df[column]<lr,lr,df[column])
```

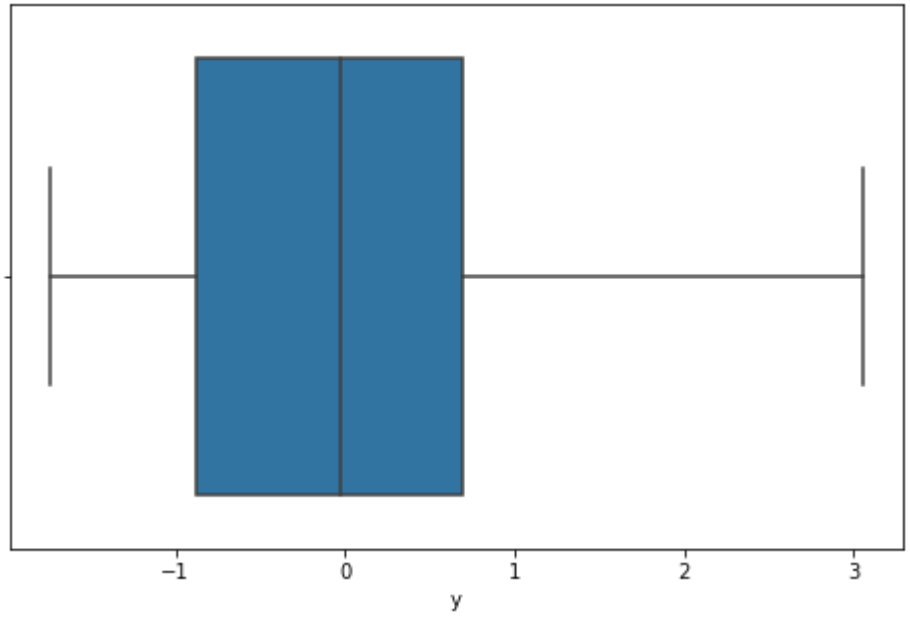
In [37]:
```
columns = ['carat' ,'depth', 'table', 'x', 'y', 'z',
       'price']

for x in columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(df[x])
    plt.show()
```

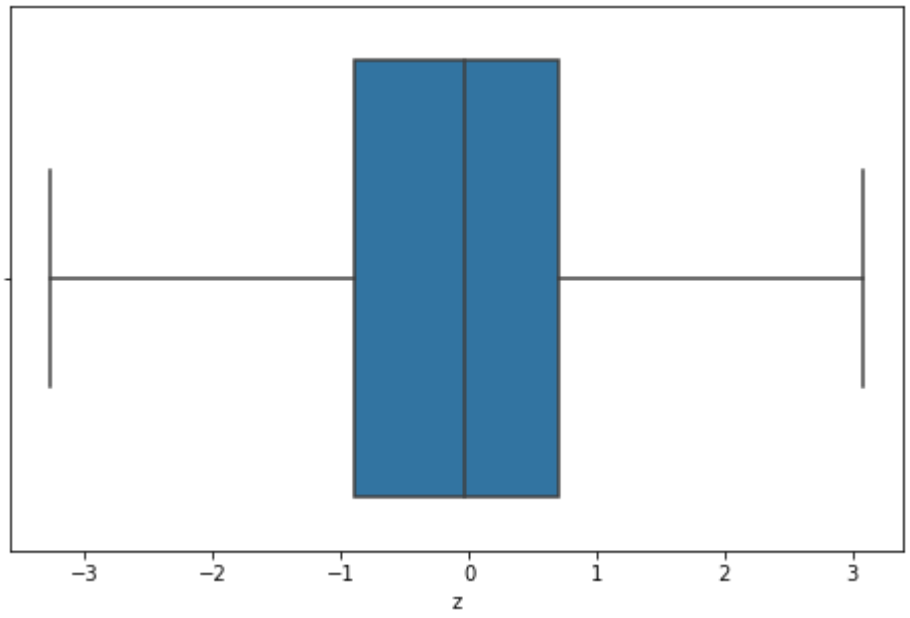

−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
carat

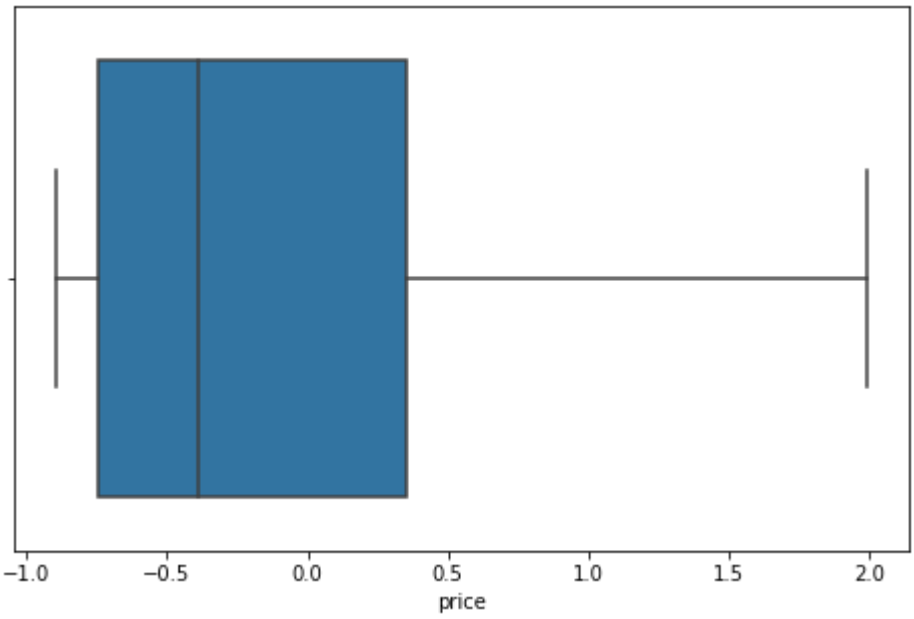

−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
depth


−2
−1
0
1
2
table


−2
−1
0
1
2
3
x


−1
0
1
2
3
y


−3
−2
−1
0
1
2
3
z



1.3 Encode the data (having string values) for Modelling. Data Split: Split the data into train and test (70:30). Apply Linear regression. Performance Metrics: Check the performance of Predictions on Train and Test sets using Rsquare, RMSE.

In [38]:
```
#First we will convert categorical to dummy variables

new_df = pd.get_dummies(df, columns=['cut','color','clarity'],drop_first=True)
```

In [39]:
```
new_df.head()
```

Out[39]:

	carat	depth	table	x	y	z	price	cut_Good	cut_Ide
1	-1.043484	0.254366	0.244117	-1.296010	-1.240036	-1.224888	-0.854844	0	
2	-0.980601	-0.678601	0.244117	-1.162802	-1.093962	-1.169137	-0.734225	0	
3	0.214178	0.326132	1.140461	0.275843	0.332401	0.336153	0.585129	0	
4	-0.791951	-0.104468	-0.652227	-0.807581	-0.801816	-0.806752	-0.709852	0	
5	-1.022523	-0.965668	0.692289	-1.224966	-1.119740	-1.238826	-0.785208	0	

5 rows × 24 columns

In [40]:
```
#Let's rename few columns for better readability

new_df.rename(columns = {'cut_Very Good' : 'cut_Very_Good'}, inplace = True)
```

Spliting the dataset into Train and Test set

In [41]:
```
#'Price' is the Target Variable:

X = new_df.drop('price', axis=1)
y = new_df[['price']]
```

In [42]:
```
#As given in the question, Split train and test into 70:30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3 , rand
om_state=1)
```

In [43]:
```
#Now let's fit the X_train and y_train into the model

reg_model = LinearRegression()

reg_model.fit(X_train, y_train)
```

Out[43]: LinearRegression()

Checking score on train and test data

In [44]:
```
#Let's check the score for train data

reg_model.score(X_train, y_train)
```

Out[44]: 0.9404719027464119

In [45]:
```
#Let's check the score for test data

reg_model.score(X_test, y_test)
```

Out[45]: 0.9416169664411843

Checking RMSE on train and test data

In [46]:
```
#Check RMSE on train data

from sklearn import metrics

predicted_train=reg_model.fit(X_train, y_train).predict(X_train)

np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[46]: 0.20984203003055799

In [47]:
```
#Check RMSE on test data

predicted_test=reg_model.fit(X_train, y_train).predict(X_test)

np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[47]: 0.20942733183439705

Checking the coefficients

In [48]:
```
for idx, column in enumerate(X_train.columns):
    print("Coefficient for {} is {}".format(column, reg_model.coef_[0][idx]))
```

```
Coefficient for carat is 1.091608986444096
Coefficient for depth is 0.004292589689086406
Coefficient for table is -0.01280995363087446
Coefficient for x is -0.3297283160277515
Coefficient for y is 0.31326746642933856
Coefficient for z is -0.11428048662800115
Coefficient for cut_Good is 0.09632120352750682
Coefficient for cut_Ideal is 0.1566526304064449
Coefficient for cut_Premium is 0.1488898790105733
Coefficient for cut_Very_Good is 0.12494507083633381
Coefficient for color_E is -0.04697340540558105
Coefficient for color_F is -0.05750774333969632
Coefficient for color_G is -0.10223606541857169
Coefficient for color_H is -0.20679844309213313
Coefficient for color_I is -0.3308004746003582
Coefficient for color_J is -0.4629825913578346
Coefficient for clarity_IF is 0.9936103167577837
Coefficient for clarity_SI1 is 0.6306802090826207
Coefficient for clarity_SI2 is 0.42581742996624716
Coefficient for clarity_VS1 is 0.8344180308840299
Coefficient for clarity_VS2 is 0.7640466334789668
Coefficient for clarity_VVS1 is 0.9393141121734576
Coefficient for clarity_VVS2 is 0.9367999621686632
```

In [49]:
```
# Let us check the intercept for the model

intercept = reg_model.intercept_[0]

print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is -0.7571488566897918
```

Linear Regression using statsmodels

In [50]:
```
# Concatenate X and y into a single dataframe:

data_train = pd.concat([X_train, y_train], axis=1)
data_test  = pd.concat([X_test,y_test],axis=1)
```

In [51]:
```
data_train.head()
```

Out[51]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut
5031	0.633398	1.115566	-0.652227	0.710988	0.727658	0.851854	1	0	
12109	0.444749	1.617933	-0.652227	0.506736	0.555807	0.726413	0	0	
20182	-0.267926	-0.750368	1.767902	-0.114900	-0.080041	-0.179548	1	0	
4713	-0.079277	-1.970402	2.484977	0.284723	0.203513	-0.095921	1	0	
2549	0.444749	0.756732	0.692289	0.568900	0.521437	0.628848	0	0	

5 rows × 24 columns

In [52]:
```
data_train.columns
```

Out[52]:
```
Index(['carat', 'depth', 'table', 'x', 'y', 'z', 'cut_Good', 'cut_Ideal',
       'cut_Premium', 'cut_Very_Good', 'color_E', 'color_F', 'color_G',
       'color_H', 'color_I', 'color_J', 'clarity_IF', 'clarity_SI1',
       'clarity_SI2', 'clarity_VS1', 'clarity_VS2', 'clarity_VVS1',
       'clarity_VVS2', 'price'],
      dtype='object')
```

In [53]:
```
expr = 'price ~ carat + depth + table + x+ y + z + cut_Good + cut_Ideal + cut_
Premium + cut_Very_Good + color_E + color_F + color_G + color_H + color_I + co
lor_J + clarity_IF + clarity_SI1 + clarity_SI2 + clarity_VS1 + clarity_VS2 + c
larity_VVS1 + clarity_VVS2'
```

In [54]:
```
import statsmodels.formula.api as smf

lm1 = smf.ols(formula= expr, data = data_train).fit()

lm1.params
```

Out[54]:
```
Intercept       -0.757149
carat            1.091609
depth            0.004293
table           -0.012810
x               -0.329728
y                0.313267
z               -0.114280
cut_Good         0.096321
cut_Ideal        0.156653
cut_Premium      0.148890
cut_Very_Good    0.124945
color_E         -0.046973
color_F         -0.057508
color_G         -0.102236
color_H         -0.206798
color_I         -0.330800
color_J         -0.462983
clarity_IF       0.993610
clarity_SI1      0.630680
clarity_SI2      0.425817
clarity_VS1      0.834418
clarity_VS2      0.764047
clarity_VVS1     0.939314
clarity_VVS2     0.936800
dtype: float64
```

In [55]:
```
print(lm1.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.940
Model:	OLS	Adj. R-squared:	0.940
Method:	Least Squares	F-statistic:	1.293e+04
Date:	Sat, 31 Jul 2021	Prob (F-statistic):	0.00
Time:	20:48:47	Log-Likelihood:	2685.0
No. Observations:	18847	AIC:	-5322.
Df Residuals:	18823	BIC:	-5134.
Df Model:	23		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.7571	0.016	-46.528	0.000	-0.789	-0.725
carat	1.0916	0.009	118.886	0.000	1.074	1.110
depth	0.0043	0.004	1.184	0.237	-0.003	0.011
table	-0.0128	0.002	-6.021	0.000	-0.017	-0.009
x	-0.3297	0.038	-8.626	0.000	-0.405	-0.255
y	0.3133	0.040	7.835	0.000	0.235	0.392
z	-0.1143	0.023	-4.886	0.000	-0.160	-0.068
cut_Good	0.0963	0.011	8.796	0.000	0.075	0.118
cut_Ideal	0.1567	0.011	14.701	0.000	0.136	0.178
cut_Premium	0.1489	0.010	14.582	0.000	0.129	0.169
cut_Very_Good	0.1249	0.011	11.898	0.000	0.104	0.146
color_E	-0.0470	0.006	-8.318	0.000	-0.058	-0.036
color_F	-0.0575	0.006	-10.046	0.000	-0.069	-0.046
color_G	-0.1022	0.006	-18.267	0.000	-0.113	-0.091
color_H	-0.2068	0.006	-34.647	0.000	-0.218	-0.195
color_I	-0.3308	0.007	-49.715	0.000	-0.344	-0.318

```
color_J          -0.4630      0.008    -56.819      0.000      -0.479      -
0.447
clarity_IF        0.9936      0.016     61.555      0.000       0.962
1.025
clarity_SI1       0.6307      0.014     45.631      0.000       0.604
0.658
clarity_SI2       0.4258      0.014     30.639      0.000       0.399
0.453
clarity_VS1       0.8344      0.014     59.087      0.000       0.807
0.862
clarity_VS2       0.7640      0.014     54.934      0.000       0.737
0.791
clarity_VVS1      0.9393      0.015     62.758      0.000       0.910
0.969
clarity_VVS2      0.9368      0.015     64.387      0.000       0.908
0.965
==============================================================================
=
Omnibus:                     4642.691   Durbin-Watson:                   2.00
2
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17342.71
2
Skew:                           1.197   Prob(JB):                         0.0
0
Kurtosis:                       7.043   Cond. No.                         67.
8
==============================================================================
=

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
```

In [56]:
```
# Calculate MSE
MSE = np.mean((lm1.predict(data_train.drop('price',axis=1))-data_train['price'
])**2)
print('Mean Square Error:',MSE)
print('_____________________________')
#Root Mean Squared Error - RMSE
print('Root mean square error:',np.sqrt(MSE))
```

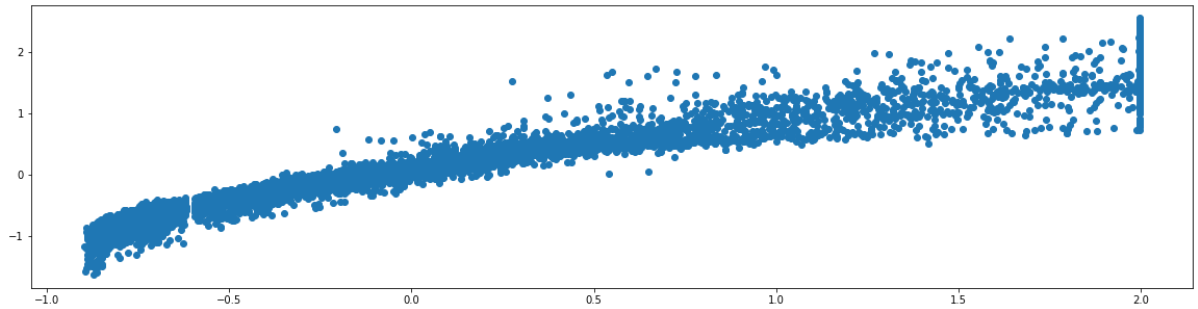
```
Mean Square Error: 0.04403367756734561
_____________________________
Root mean square error: 0.209842030030558
```

In [57]:
```
#model score

reg_model.score(X_test, y_test)
```

Out[57]: 0.9416169664411843

In [58]:
```
y_pred = reg_model.predict(X_test)
```

In [59]:
```
plt.scatter(y_test['price'], y_pred)
plt.show()
```



In [60]:
```
for i,j in np.array(lm1.params.reset_index()):
    print('({}) * {} +'.format(round(j,2),i),end=' ')
```

```
(-0.76) * Intercept + (1.09) * carat + (0.0) * depth + (-0.01) * table + (-0.
33) * x + (0.31) * y + (-0.11) * z + (0.1) * cut_Good + (0.16) * cut_Ideal +
(0.15) * cut_Premium + (0.12) * cut_Very_Good + (-0.05) * color_E + (-0.06) *
color_F + (-0.1) * color_G + (-0.21) * color_H + (-0.33) * color_I + (-0.46)
* color_J + (0.99) * clarity_IF + (0.63) * clarity_SI1 + (0.43) * clarity_SI2
+ (0.83) * clarity_VS1 + (0.76) * clarity_VS2 + (0.94) * clarity_VVS1 + (0.9
4) * clarity_VVS2 +
```

1.4 Inference: Basis on these predictions, what are the business insights and recommendations.

Insights and Observations:

1. The most preferred cut chosen by the customers is the Ideal cut, because it is cheaper than the other available cuts.
2. We can also see that the reason behind the 'Fair' being the least preffered is it's price.
3. The quality of Fair type cut is low and price is also high.
4. According to the given information, D being the best color while J being the worst but still it is most costly among all of them.
5. The most preferred color is G, average priced.
6. In terms of clarity, VS2 seems to be most chosen by the customers and SI2 seems to be the costliest clarity stone.
7. We have also observed from the pair plot that with the increase in carat, the price is also increasing.
8. However, the dimesnions of the stone does not effect much on the price of the stone.
9. Multicollinearity also present in the data.
10. Similarly, the company is gaining profits due to the colors H and I, also company is building profits from the Ideal, premium and very_good types of cut out of 5 available cuts.

Recommendations:

1) TOP LEVEL CUT. Based on the above few observations from the concluded result, we can say that cut and clarity holds the power of growing the profits for the company and that is the biggest benefit, because the upper level cuts like Ideal cut, premium cut and the very_good type of cut captures the higher contribution towards the company's growth, and so i recommend that we should display and prioritize those 3 cuts and company should make the best marketing strategy to boost the sales of these stones.

2) BETTER THE QUALITY, MORE THE PROFIT. Also, I can see that the clarity of the stones holds the power to boost the profit for the company. So I recommend that the company should focus more on the stone's clarity, because better the clarity, more the profit.

3) FOCUS LESS ON THE DIMENSIONS, MORE ON CARAT: Length, width and Height of the stones does not affect much on the price of the stones, but carat of the stones bring the profits. So I recommend that company should ask the production team to get the best of the quality to its stones, so that profit would be better in such cases.

_ THE END __

Problem 2: Logistic Regression and LDA

Problem Statement:

You are hired by a tour and travel agency which deals in selling holiday packages. You are provided details of 872 employees of a company. Among these employees, some opted for the package and some didn't. You have to help the company in predicting whether an employee will opt for the package or not on the basis of the information given in the data set. Also, find out the important factors on the basis of which the company will focus on particular employees to sell their packages.

Data Dictionary:

1. Holiday_Package: Opted for Holiday Package yes/no?
2. Salary: Employee salary
3. age: Age in years
4. edu: Years of formal education
5. no_young_children: The number of young children (younger than 7 years)
6. no_older_children: Number of older children
7. foreign: foreigner Yes/No

In [61]:
```
import numpy as np, pandas as pd, seaborn as sns, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,conf
usion_matrix,plot_confusion_matrix
```

2.1 Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it. Perform Univariate and Bivariate Analysis. Do exploratory data analysis.

Loading and Reading the dataset

In [62]:
```
holiday = pd.read_csv('D:\\SHUBHANK !\\GL\\6. TOPIC 5 - Predictive Modeling\\Final Project\\Holiday_Package.csv', index_col=[0])

holiday
```

Out[62]:

	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
1	no	48412	30	8	1	1	no
2	yes	37207	45	8	0	1	no
3	no	58022	46	9	0	0	no
4	no	66503	31	11	2	0	no
5	no	66734	44	12	0	2	no
...	...	...	...	...	...	...	...
868	no	40030	24	4	2	1	yes
869	yes	32137	48	8	0	0	yes
870	no	25178	24	6	2	0	yes
871	yes	55958	41	10	0	1	yes
872	no	74659	51	10	0	0	yes

872 rows × 7 columns

Let's rename the column first for better readability

In [63]:
```
holiday.rename(columns = {'Holliday_Package':'Holiday_Package','age':'Age', 'educ':'Education(In Years)', 'no_young_children':'Num_Young_Children(<7 yrs)', 'no_older_children':'Num_Older_Children', 'foreign':'Foreign'}, inplace=True)
```

In [64]:
```
holiday
```

Out[64]:

	Holiday_Package	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children
1	no	48412	30	8	1	1
2	yes	37207	45	8	0	1
3	no	58022	46	9	0	0
4	no	66503	31	11	2	0
5	no	66734	44	12	0	2
...	...	...	...	...	...	...
868	no	40030	24	4	2	1
869	yes	32137	48	8	0	0
870	no	25178	24	6	2	0
871	yes	55958	41	10	0	1
872	no	74659	51	10	0	0

872 rows × 7 columns

In [65]:
```
#Let's check the shape

print('No. of rows:',holiday.shape[0])
print('No. of columns:',holiday.shape[1])
```

```
No. of rows: 872
No. of columns: 7
```

In [66]:
```
#Description about the dataset

holiday.describe()
```

Out[66]:

	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children
count	872.000000	872.000000	872.000000	872.000000	872.000000
mean	47729.172018	39.955275	9.307339	0.311927	0.982798
std	23418.668531	10.551675	3.036259	0.612870	1.086786
min	1322.000000	20.000000	1.000000	0.000000	0.000000
25%	35324.000000	32.000000	8.000000	0.000000	0.000000
50%	41903.500000	39.000000	9.000000	0.000000	1.000000
75%	53469.500000	48.000000	12.000000	0.000000	2.000000
max	236961.000000	62.000000	21.000000	3.000000	6.000000

The maximum age is 62 and the minimum age is 20.

In [67]:
```
#Some more info about the data

holiday.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 872 entries, 1 to 872
Data columns (total 7 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Holiday_Package             872 non-null    object
 1   Salary                      872 non-null    int64 
 2   Age                         872 non-null    int64 
 3   Education(In Years)         872 non-null    int64 
 4   Num_Young_Children(<7 yrs)  872 non-null    int64 
 5   Num_Older_Children          872 non-null    int64 
 6   Foreign                     872 non-null    object
dtypes: int64(5), object(2)
memory usage: 54.5+ KB
```

There are no non-null values. There are 2 columns with object datatype and 5 columns with integer datatype.

In [68]:
```
#Let's check for any missing values

holiday.isnull().sum()
```

Out[68]:
```
Holiday_Package             0
Salary                      0
Age                         0
Education(In Years)         0
Num_Young_Children(<7 yrs)  0
Num_Older_Children          0
Foreign                     0
dtype: int64
```

In [69]:
```
#Checking for Duplicate data

holiday.duplicated().sum()
```

Out[69]:
```
0
```

In [70]:
```
#Value Counts in object data type columns

holiday.Holiday_Package.value_counts()
```

Out[70]:
```
no     471
yes    401
Name: Holiday_Package, dtype: int64
```

In [71]:
```
holiday.Foreign.value_counts()
```

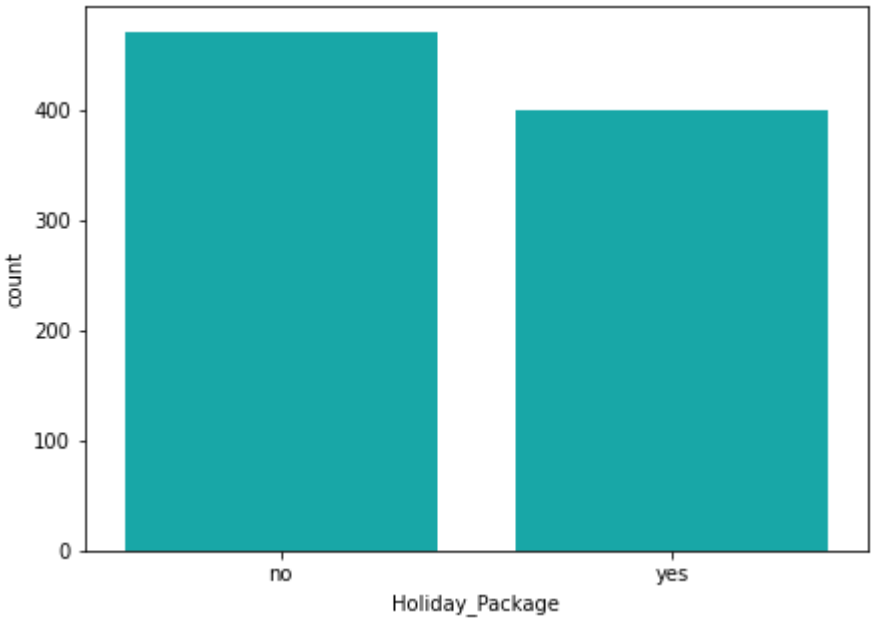
Out[71]:
```
no     656
yes    216
Name: Foreign, dtype: int64
```

In [72]:
```
holiday_df = holiday.copy()
```

Countplots for Categorical Variables

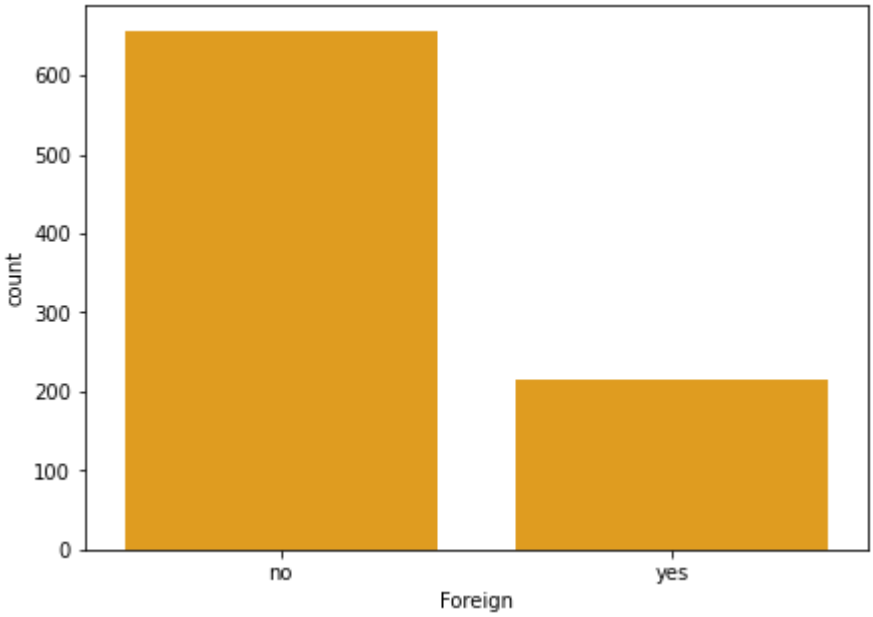
In [73]:
```
plt.figure(figsize=(7,5))
sns.countplot(x="Holiday_Package", data=holiday, color="c")
```

Out[73]:
```
<AxesSubplot:xlabel='Holiday_Package', ylabel='count'>
```



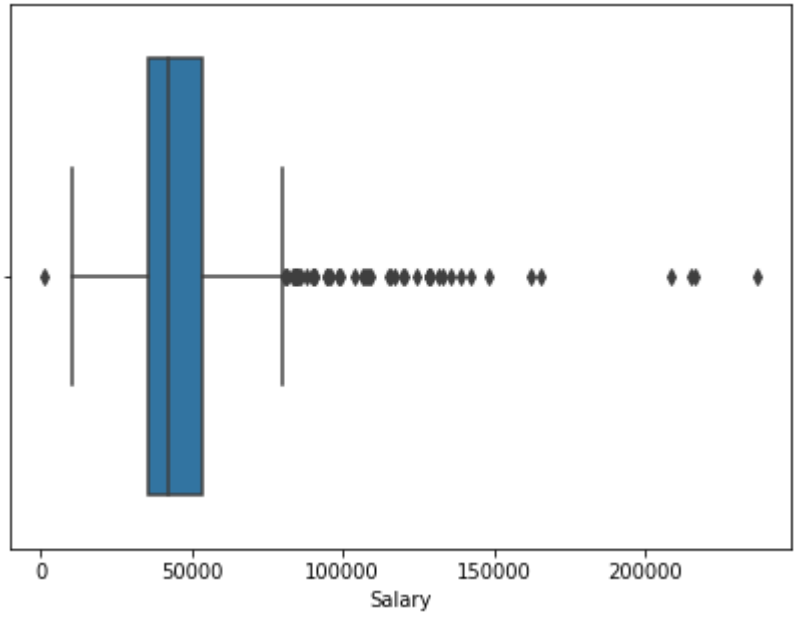
In [74]:
```
plt.figure(figsize=(7,5))
sns.countplot(x="Foreign", data=holiday, color="orange")
```

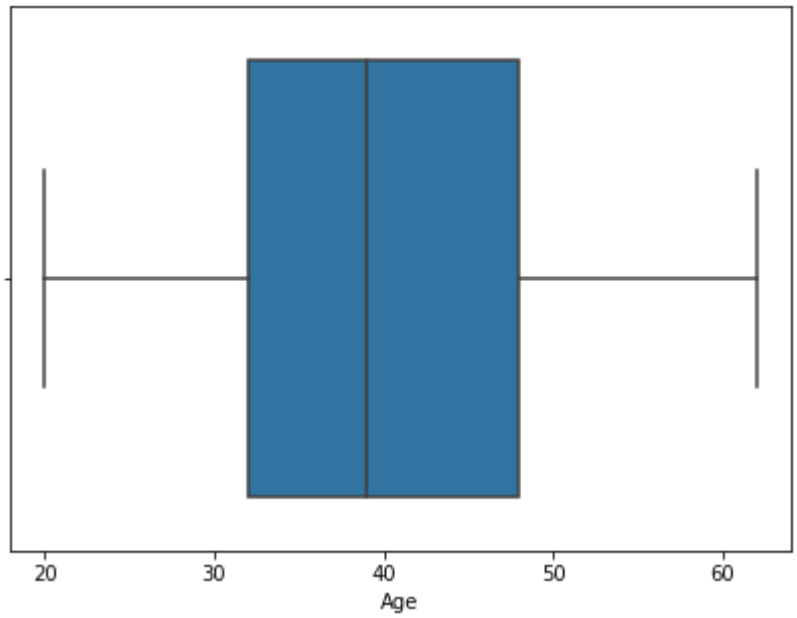
Out[74]: <AxesSubplot:xlabel='Foreign', ylabel='count'>

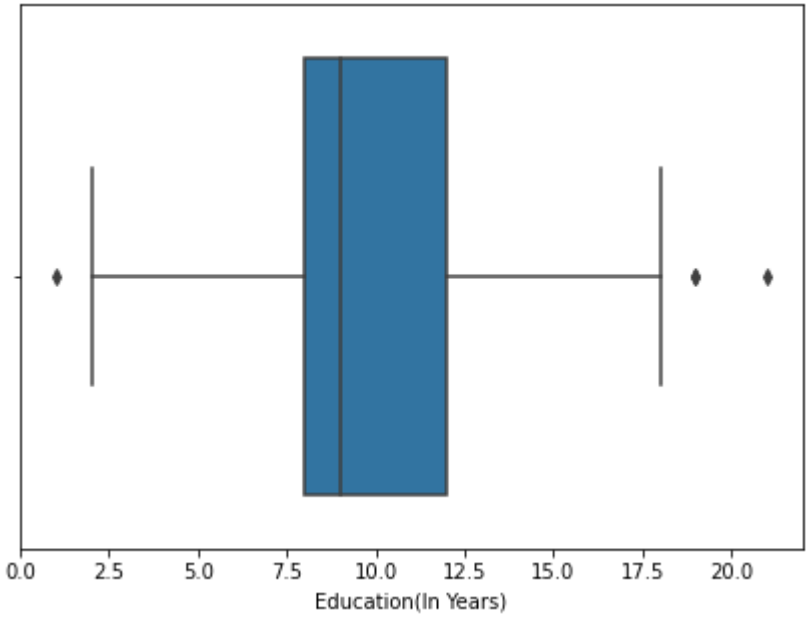


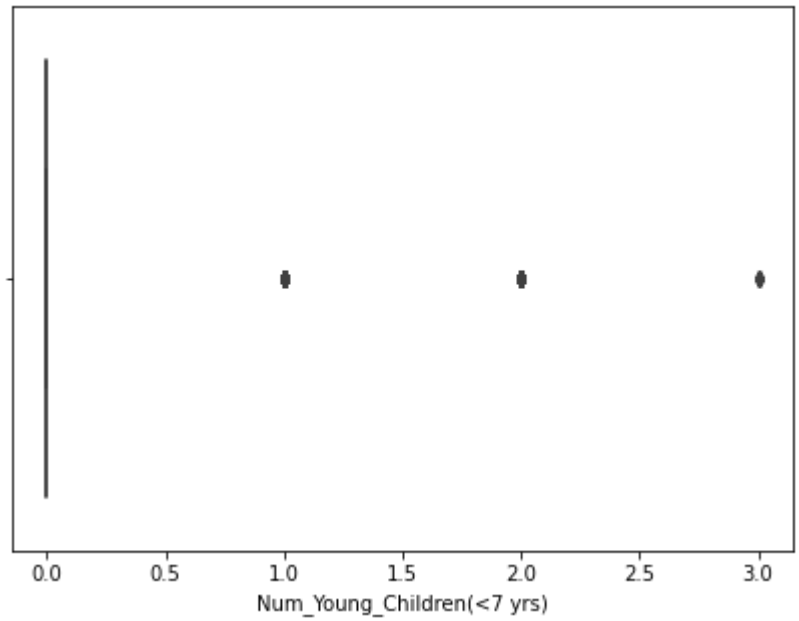
Boxplot for All Continuous Varibales

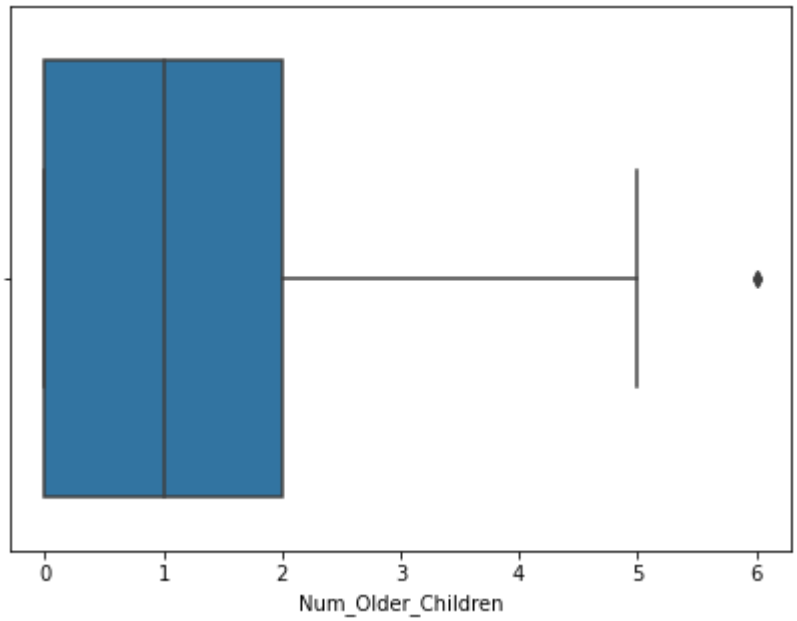
In [75]:
```
columns = ['Salary','Age','Education(In Years)','Num_Young_Children(<7 yrs)',
'Num_Older_Children']
for i in columns:
    plt.figure(figsize=(7,5))
    sns.boxplot(holiday[i])
    plt.show();
```


0
50000
100000
150000
200000
Salary


20
30
40
50
60
Age


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Education(In Years)


0.0
0.5
1.0
1.5
2.0
2.5
3.0
Num_Young_Children(<7 yrs)

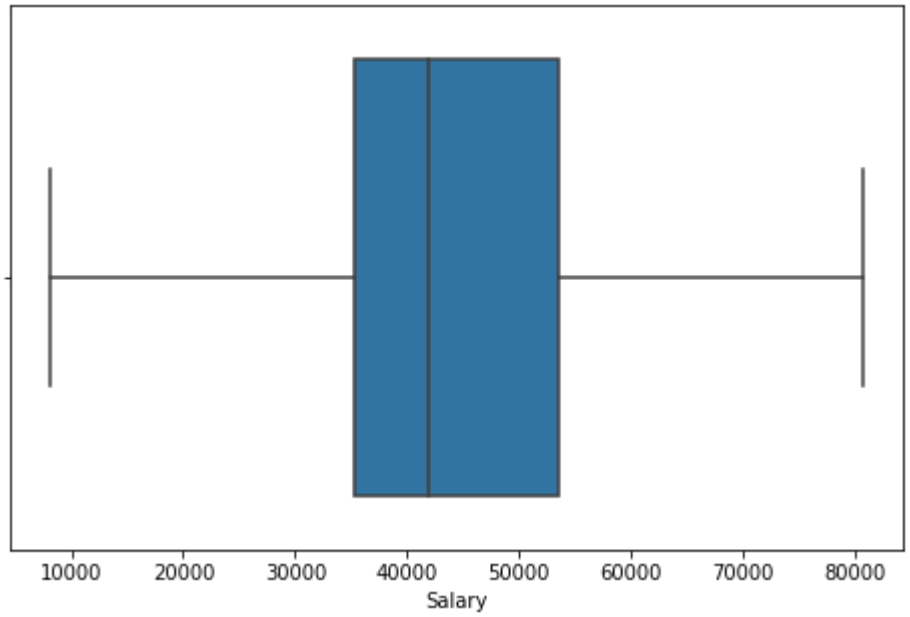


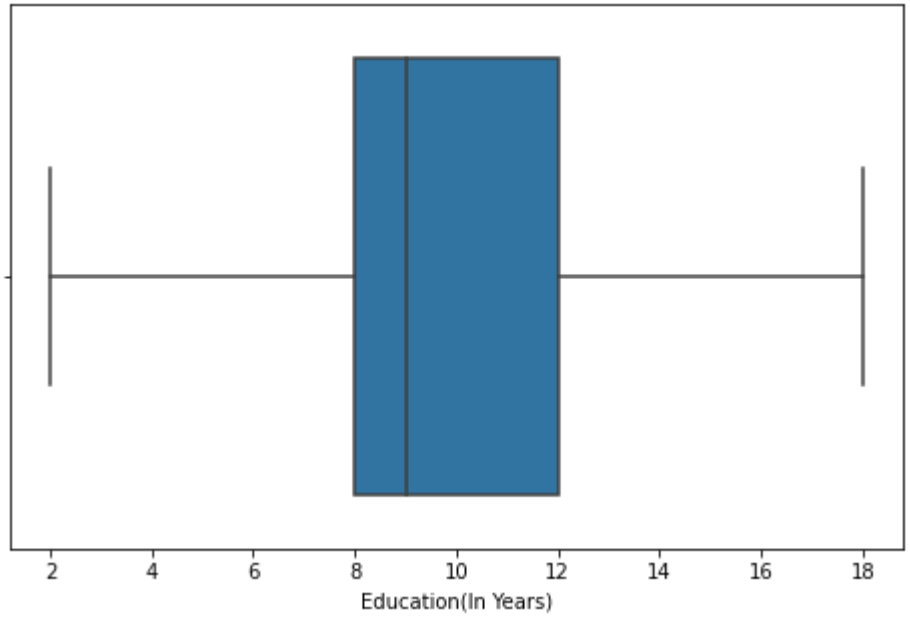
Treating Outliers

In [76]:
```
#We will only treat outliers from 'Salary' and 'Education'

def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

In [77]:
```
cont_variables = ['Salary','Education(In Years)']

for column in holiday[cont_variables].columns:
    lr,ur=remove_outlier(holiday[column])
    holiday[column]=np.where(holiday[column]>ur,ur,holiday[column])
    holiday[column]=np.where(holiday[column]<lr,lr,holiday[column])
```

In [78]:
```
for k in cont_variables:
    plt.figure(figsize=(8,5))
    sns.boxplot(holiday[k])
    plt.show()
```



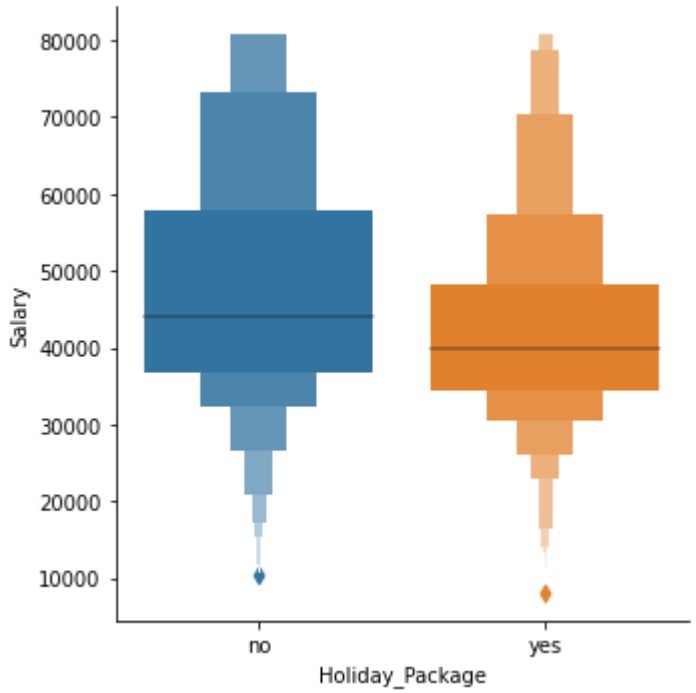


Now Outliers are removed from the above columns.

Catplots:

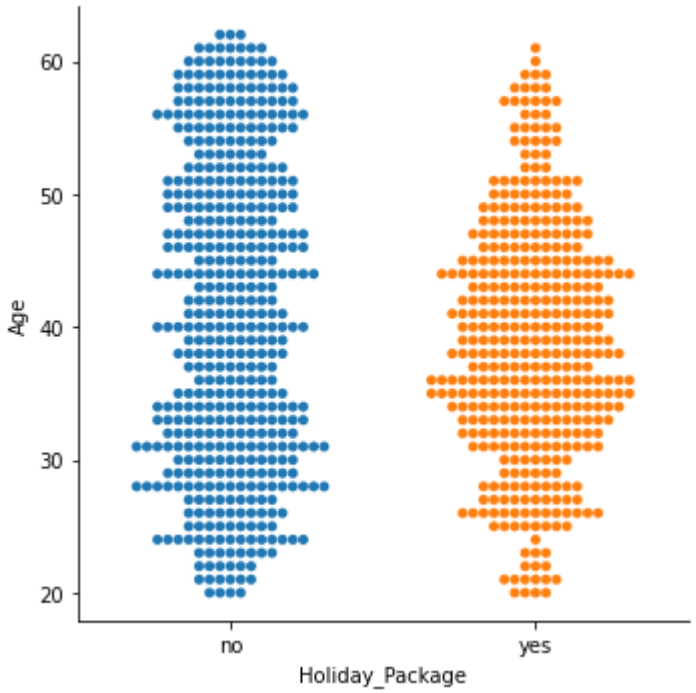
In [79]:
```
#Catplot for 'foreign' with respect to 'Salary'

sns.catplot(x='Holiday_Package', y='Salary',data=holiday, kind='boxen')
```

Out[79]: <seaborn.axisgrid.FacetGrid at 0x28d857cfa00>



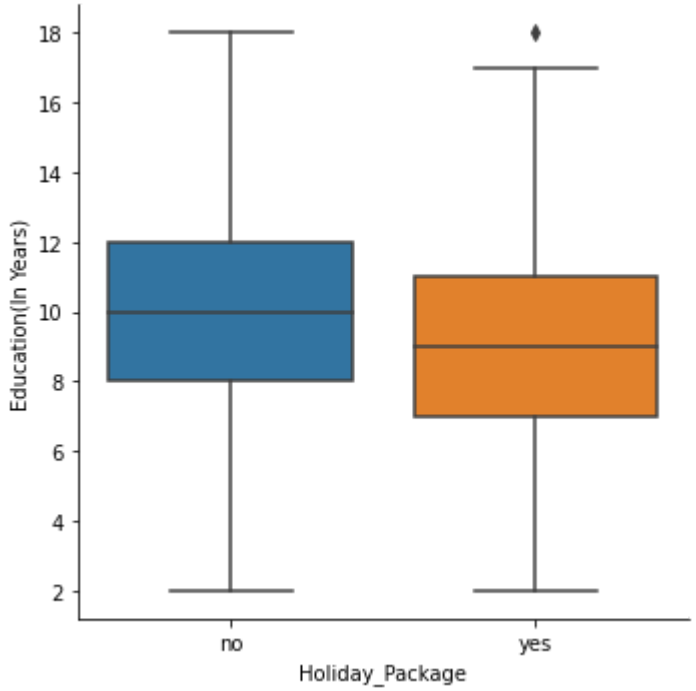
In [80]:
```
#Catplot for 'Holiday_Package' with respect to 'Age'

sns.catplot(x="Holiday_Package", y="Age",data=holiday, kind="swarm")
```

Out[80]: <seaborn.axisgrid.FacetGrid at 0x28d853a1850>



In [81]:
```
sns.catplot(x="Holiday_Package", y="Education(In Years)",data=holiday, kind="box")
```

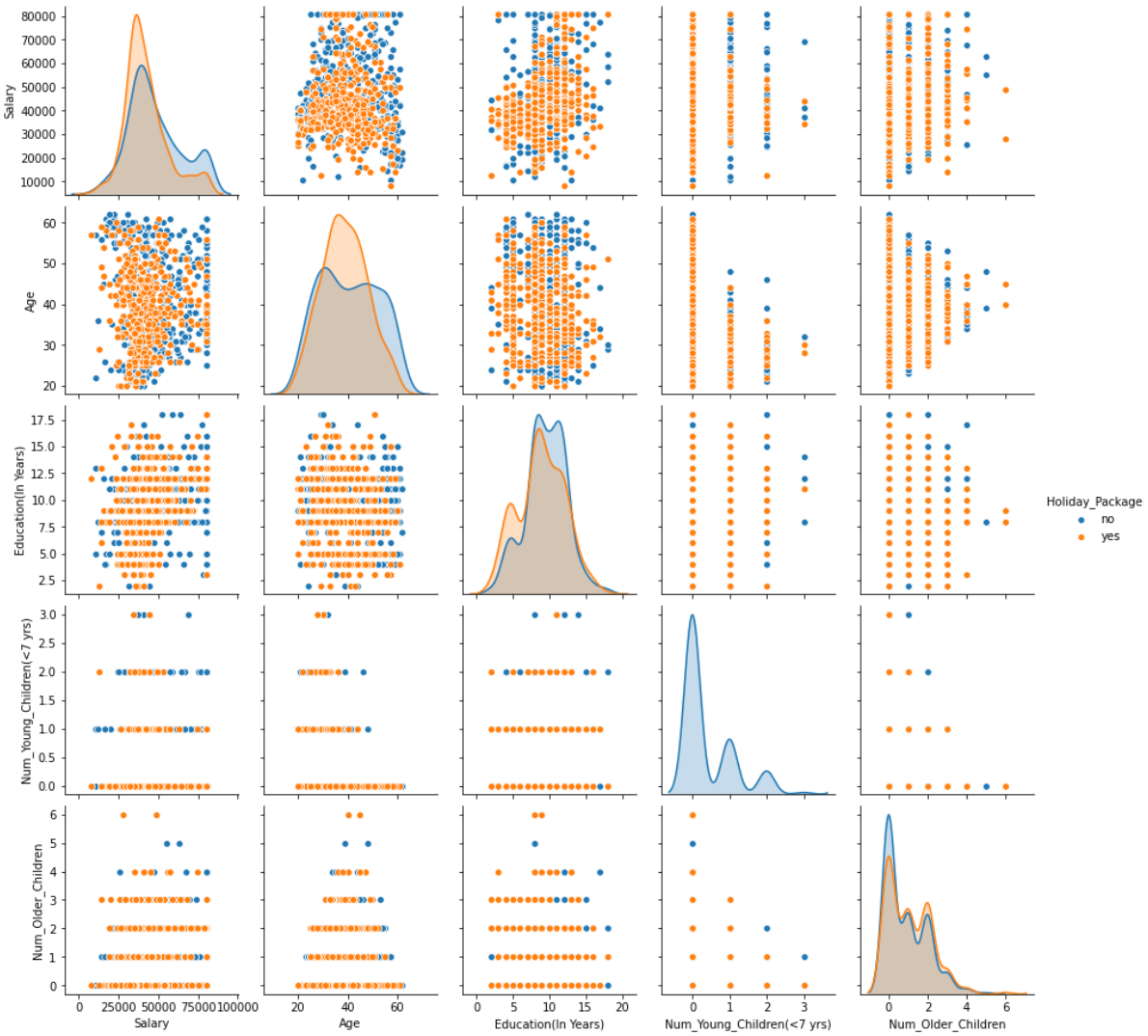
Out[81]: <seaborn.axisgrid.FacetGrid at 0x28d867ca910>



Pairplot:

In [82]:
```
sns.pairplot(data=holiday ,hue='Holiday_Package', diag_kind='kde');
```

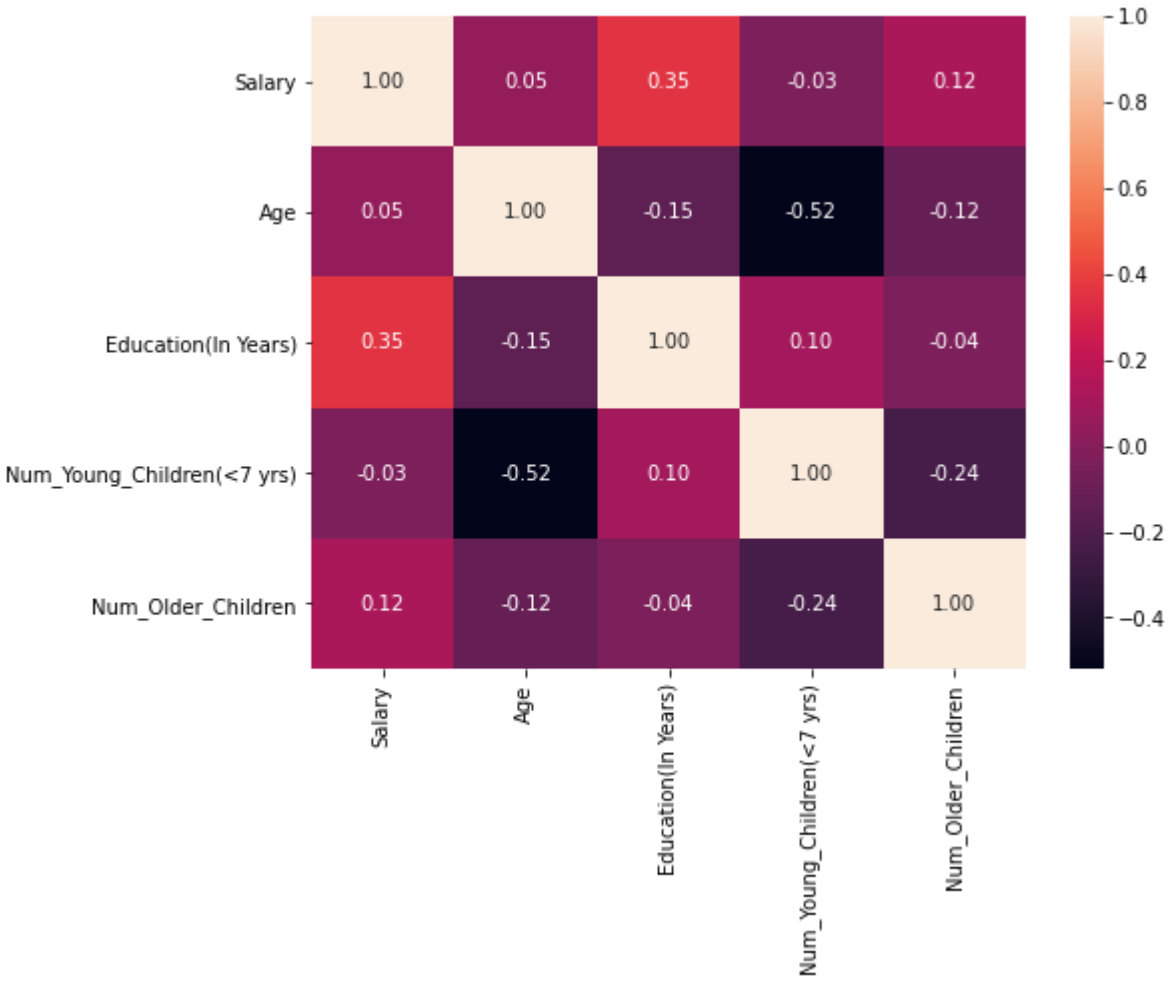
```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:369: UserW
arning: Default bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```



Heatmap and Correlation

In [83]:
```
holiday_corr = holiday.corr()
plt.figure(figsize=(8,6))
sns.heatmap(holiday_corr, annot=True, fmt = '.2f')
```

Out[83]: <AxesSubplot:>



There isn't any strong correlation in the data.

2.2 Do not scale the data. Encode the data (having string values) for Modelling. Data Split: Split the data into train and test (70:30). Apply Logistic Regression and LDA (linear discriminant analysis).

In [84]:
```
#Let's first keep the original dataframe separate

df = holiday.copy()

df.head()
```

Out[84]:

	Holiday_Package	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children	F
1	no	48412.0	30	8.0	1	1	
2	yes	37207.0	45	8.0	0	1	
3	no	58022.0	46	9.0	0	0	
4	no	66503.0	31	11.0	2	0	
5	no	66734.0	44	12.0	0	2	

In [85]:
```
#Get dummies

dummy_df = pd.get_dummies(df, columns=['Holiday_Package','Foreign'], drop_firs
t = True)
```

In [86]:
```
dummy_df.head()
```

Out[86]:

	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children	Holiday_Package_ye
1	48412.0	30	8.0	1	1	
2	37207.0	45	8.0	0	1	
3	58022.0	46	9.0	0	0	
4	66503.0	31	11.0	2	0	
5	66734.0	44	12.0	0	2	

In [87]:
```
dummy_df.shape
```

Out[87]:
```
(872, 7)
```

In [88]:
```
dummy_df.columns
```

Out[88]:
```
Index(['Salary', 'Age', 'Education(In Years)', 'Num_Young_Children(<7 yrs)',
       'Num_Older_Children', 'Holiday_Package_yes', 'Foreign_yes'],
      dtype='object')
```

Splitting into train and test set

In [89]:
```
#Holiday_Package_yes is the Target variable

X = dummy_df.drop('Holiday_Package_yes', axis=1)
y = dummy_df['Holiday_Package_yes']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , ran
dom_state=1,stratify=y)
```

In [90]:
```
y_train.value_counts(1)
```

Out[90]:
```
0    0.539344
1    0.460656
Name: Holiday_Package_yes, dtype: float64
```

In [91]:
```
y_test.value_counts(1)
```

Out[91]:
```
0    0.541985
1    0.458015
Name: Holiday_Package_yes, dtype: float64
```

Let's now build the Logistic Regression Model:

We are making some adjustments to the parameters in the Logistic Regression Class to get a better accuracy.

In [92]:
```
#We will use GridSearchCV for getting the best parameters for the model

from sklearn.model_selection import GridSearchCV

grid={'penalty':['l1','l2','none'], 'solver':['lbfgs', 'liblinear'], 'tol':[0.
0001,0.000001]}
```

In [93]:
```
# Fit the Logistic Regression model

model = LogisticRegression(max_iter=100000,n_jobs=2)

grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = 3,n_jobs
=-1,scoring='f1')
```

In [94]:
```
grid_search.fit(X_train, y_train)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1
353: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' is set
to 'liblinear'. Got 'n_jobs' = 2.
  warnings.warn("'n_jobs' > 1 does not have any effect when"
```

Out[94]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=100000, n_jobs=2),
             n_jobs=-1,
             param_grid={'penalty': ['l1', 'l2', 'none'],
                         'solver': ['lbfgs', 'liblinear'],
                         'tol': [0.0001, 1e-06]},
             scoring='f1')
```

In [95]:
```
#Getting the best parameters and best grid estimators:

grid_search.best_params_
```

Out[95]:
```
{'penalty': 'l1', 'solver': 'liblinear', 'tol': 1e-06}
```

In [96]:
```
best_model = grid_search.best_estimator_
best_model
```

Out[96]:
```
LogisticRegression(max_iter=100000, n_jobs=2, penalty='l1', solver='liblinea
r',
                   tol=1e-06)
```

Predicting on Training and Test dataset

In [97]:
```
ytrain_predict = best_model.predict(X_train)
ytest_predict = best_model.predict(X_test)
```

In [98]:
```
## Getting the probabilities on the test set

ytest_predict_prob = best_model.predict_proba(X_test)
```

In [99]:
```
#Checking the head from the predicted probability

pd.DataFrame(ytest_predict_prob).head(10)
```

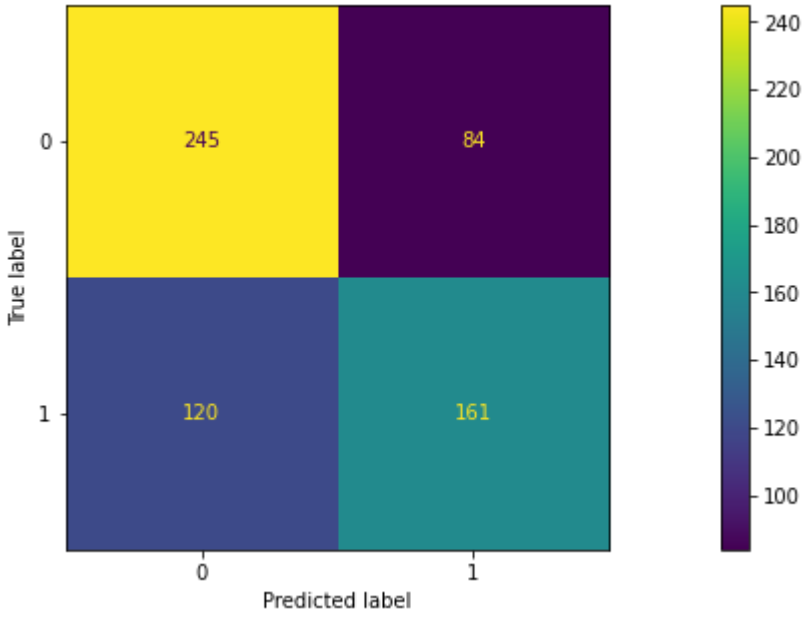
Out[99]:

	0	1
0	0.671228	0.328772
1	0.560763	0.439237
2	0.684892	0.315108
3	0.510392	0.489608
4	0.560310	0.439690
5	0.345337	0.654663
6	0.573157	0.426843
7	0.303291	0.696709
8	0.386480	0.613520
9	0.633587	0.366413

Confusion Matrix and Classification Report on the Training Data

In [100]:
```
## Confusion matrix on the training data

plot_confusion_matrix(best_model,X_train,y_train)
```

Out[100]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28d8622ad
00>
```



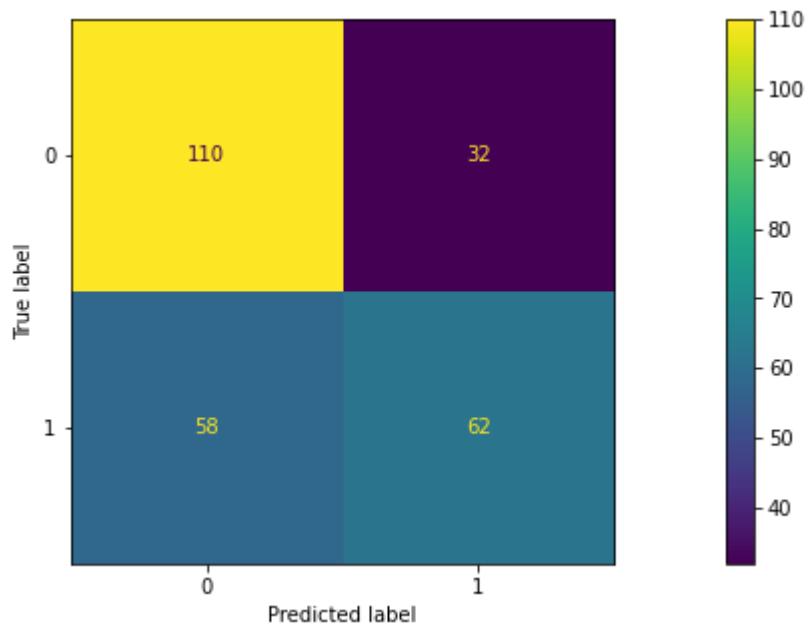
In [101]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.67      0.74      0.71       329
           1       0.66      0.57      0.61       281

    accuracy                           0.67       610
   macro avg       0.66      0.66      0.66       610
weighted avg       0.66      0.67      0.66       610
```

Confusion Matrix and Classification Report on Testing Data

In [102]:
```
## Confusion matrix on the test data

plot_confusion_matrix(best_model,X_test,y_test)
```

Out[102]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28d817ef2
e0>
```



In [103]:
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.65      0.77      0.71       142
           1       0.66      0.52      0.58       120

    accuracy                           0.66       262
   macro avg       0.66      0.65      0.64       262
weighted avg       0.66      0.66      0.65       262
```

Model Evaluation

In [104]:
```
# Accuracy score for training data

acc_train = best_model.score(X_train, y_train)

acc_train
```

Out[104]: 0.6655737704918033

AUC and ROC for the training data

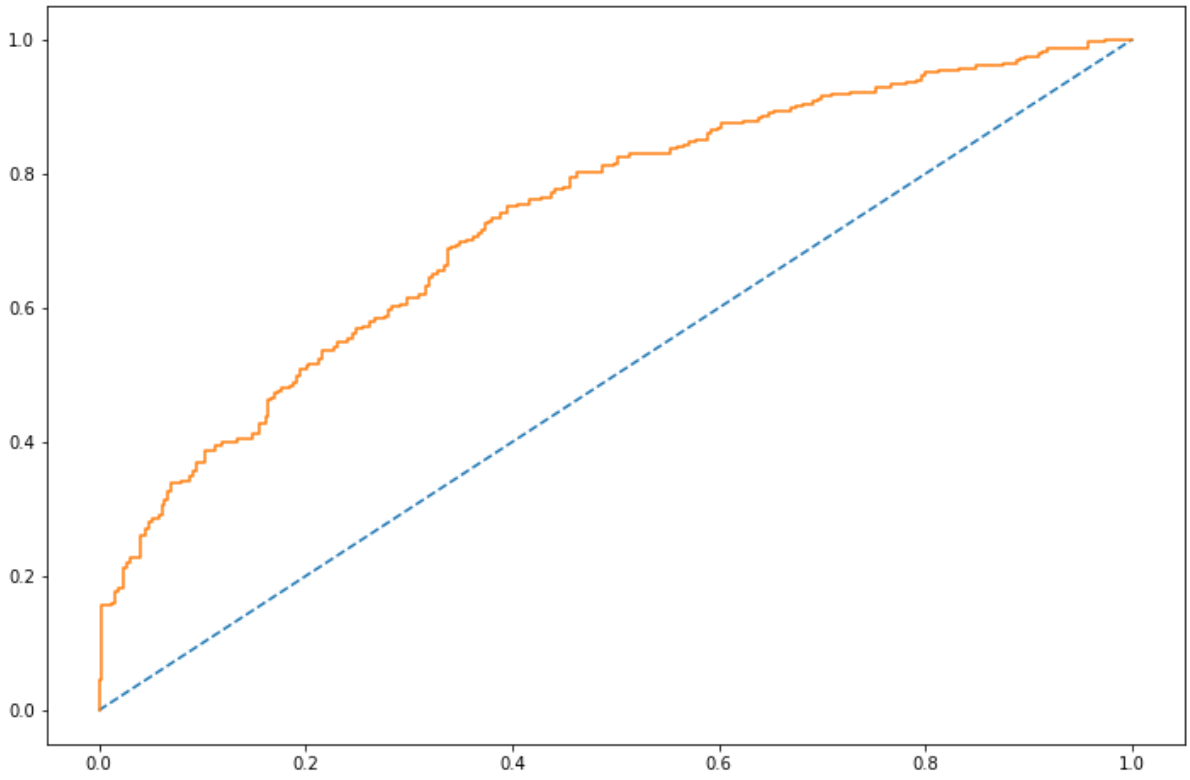
In [105]:
```
# predict probabilities

probs = best_model.predict_proba(X_train)
```

In [106]:
```
# keep probabilities for the positive outcome only

probs = probs[:, 1]
```

In [107]:
```
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
```

```
AUC: 0.733
```

In [108]:
```
# calculate roc curve

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.rcParams["figure.figsize"] = (12,8)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

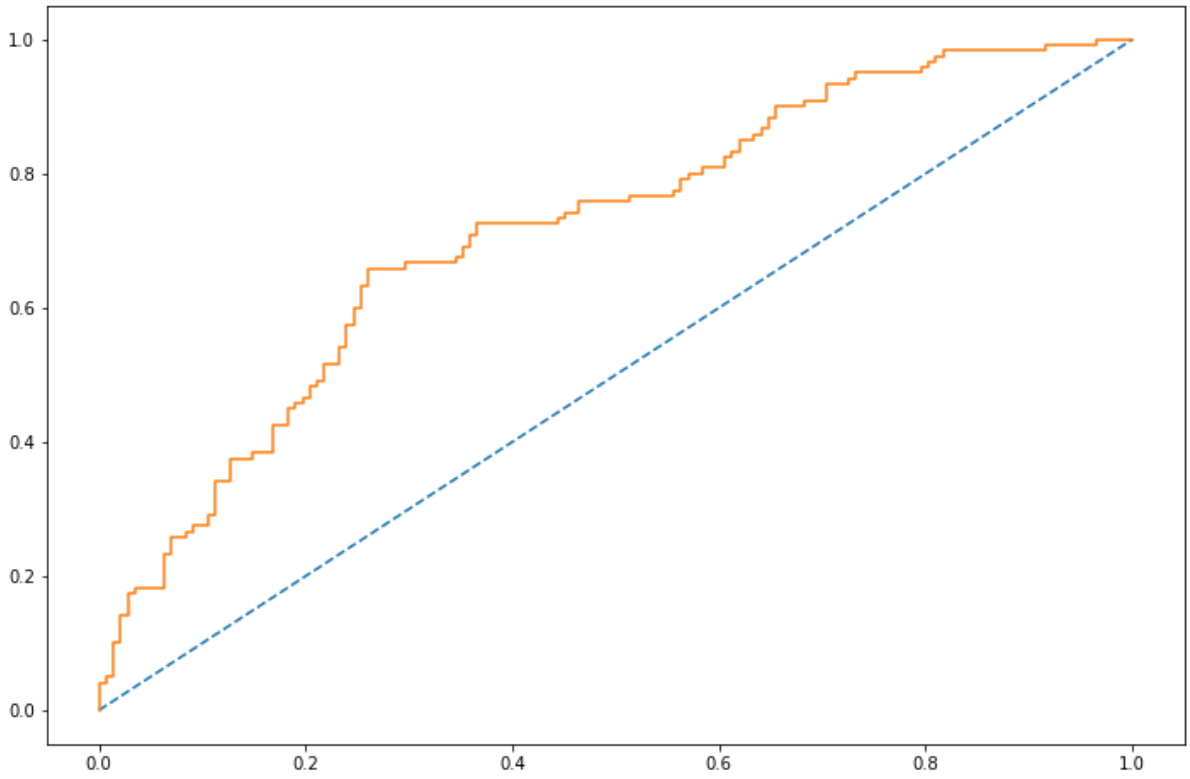


In [109]:
```
#Accuracy score for test data

acc_test = best_model.score(X_test, y_test)

acc_test
```

Out[109]: 0.6564885496183206

In [110]:
```
# predict probabilities
probs = best_model.predict_proba(X_test)

# Probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % test_auc)

# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.715
```



Linear Discriminant Analysis (LDA)

In [111]:
```
#Let's go back to our original dataframe 'holiday'

holiday_df.head()
```

Out[111]:

	Holiday_Package	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children	Fo
1	no	48412	30	8	1	1	
2	yes	37207	45	8	0	1	
3	no	58022	46	9	0	0	
4	no	66503	31	11	2	0	
5	no	66734	44	12	0	2	

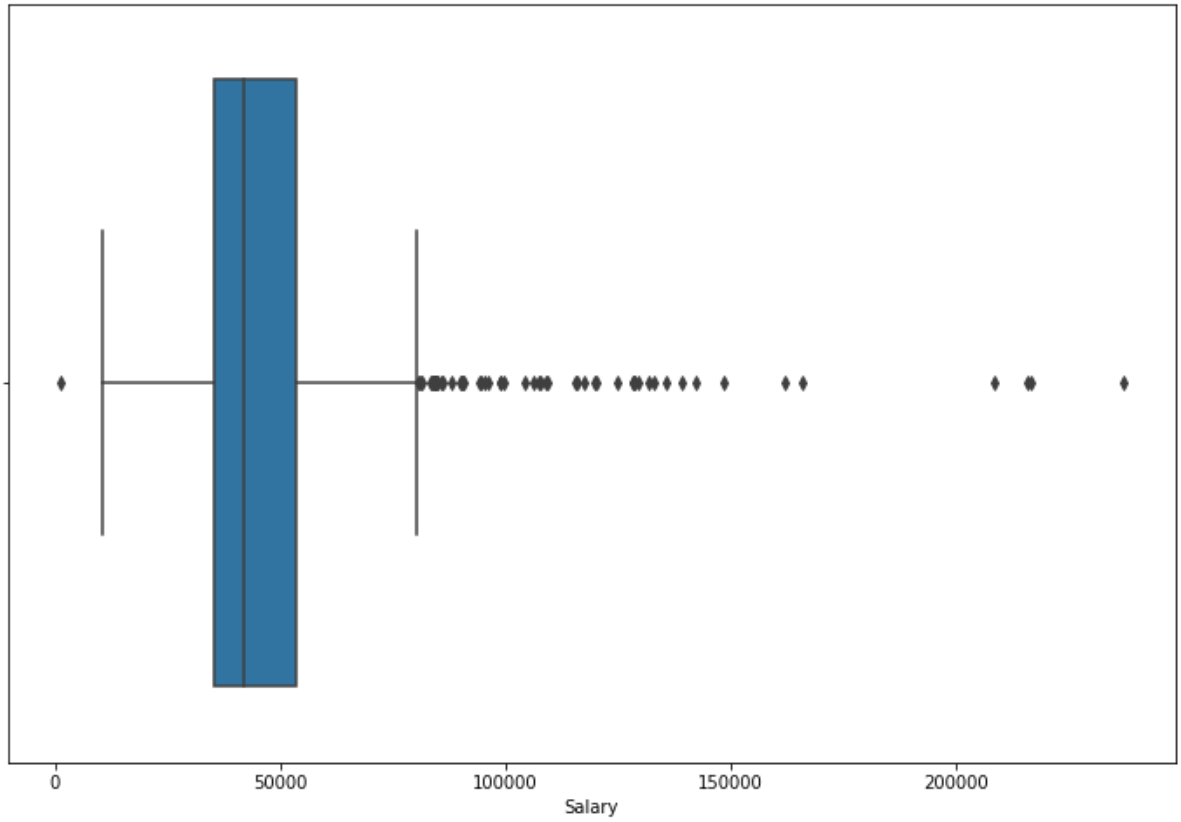
In [112]:
```
holiday_df.isnull().sum()
```

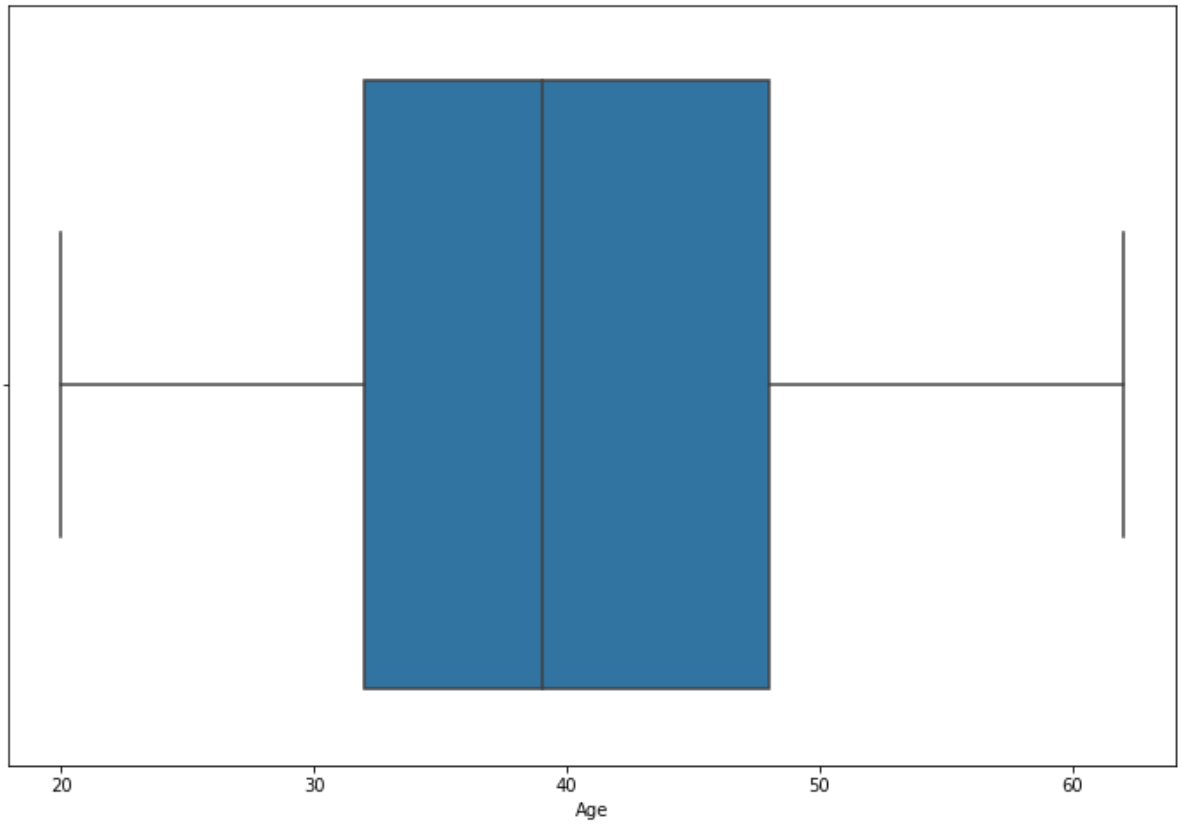
Out[112]:
```
Holiday_Package             0
Salary                      0
Age                         0
Education(In Years)         0
Num_Young_Children(<7 yrs)  0
Num_Older_Children          0
Foreign                     0
dtype: int64
```

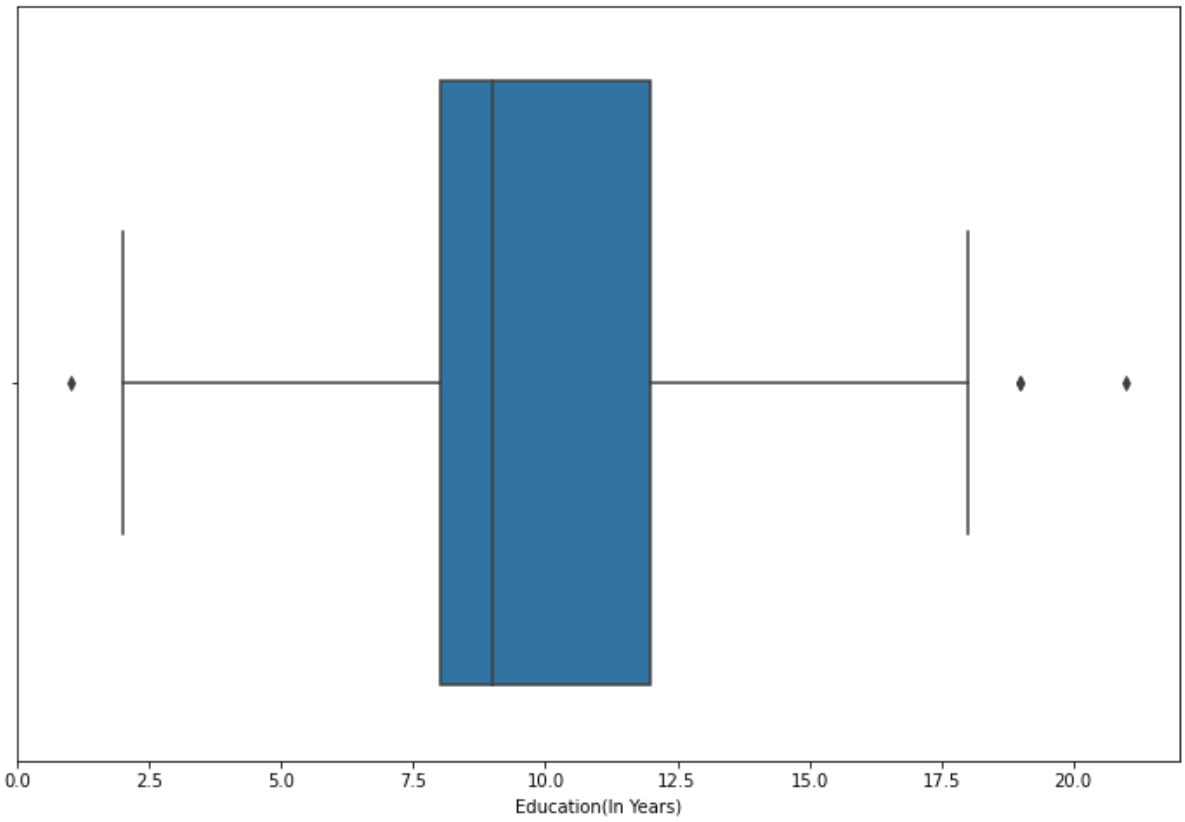
In [113]:
```
holiday_df.duplicated().sum()
```

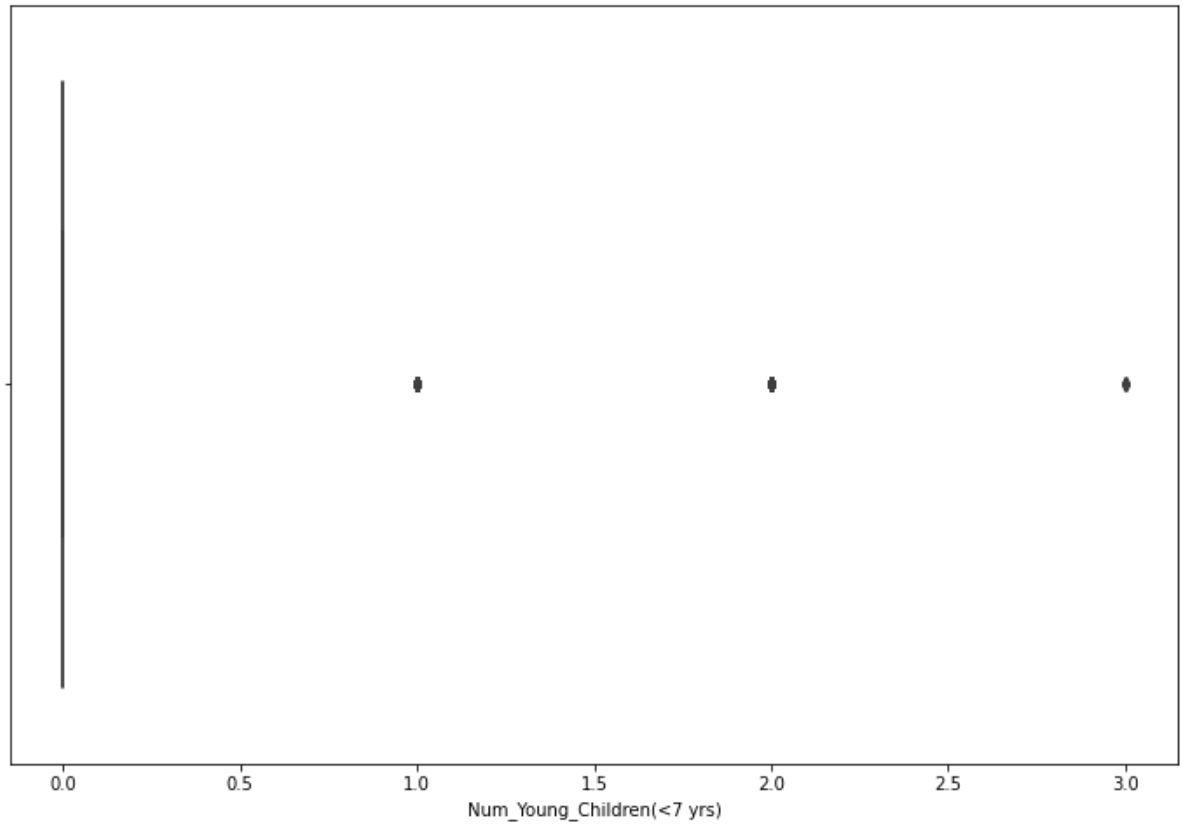
Out[113]:
```
0
```

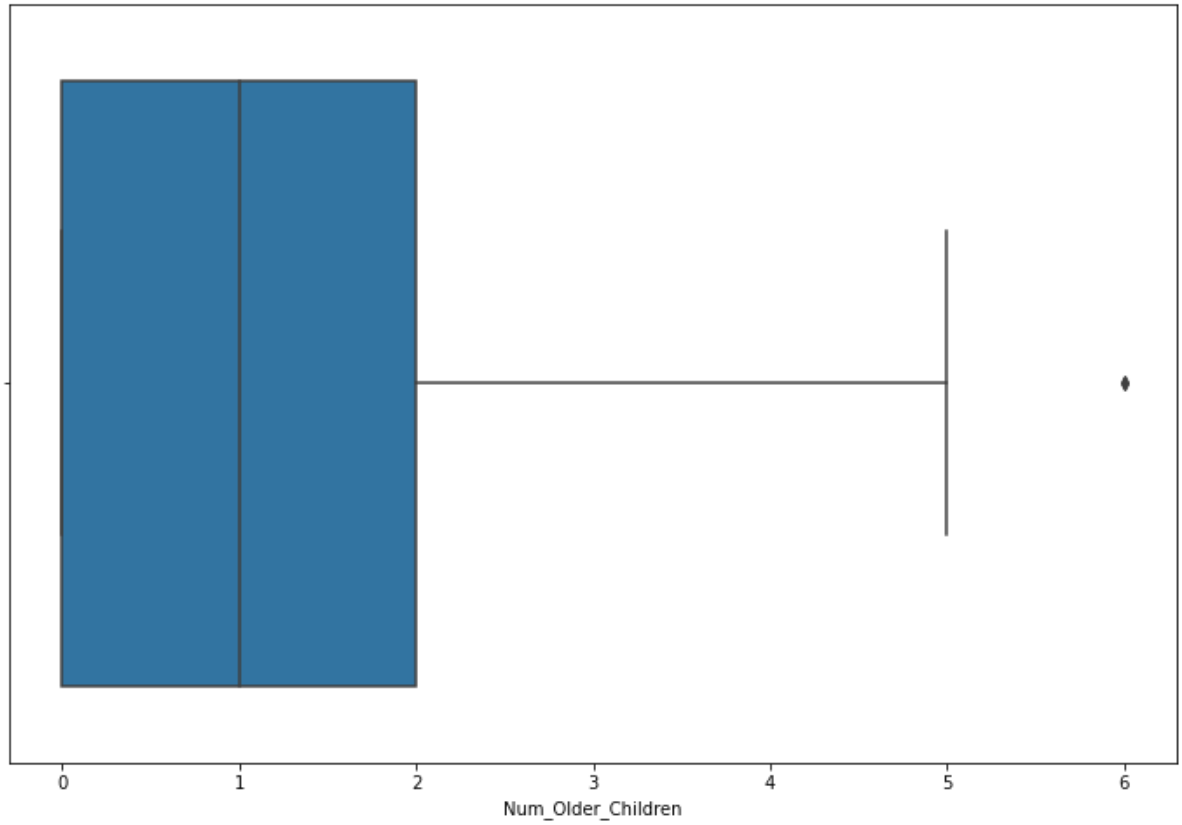
In [114]:
```
cols = ['Salary' ,'Age', 'Education(In Years)', 'Num_Young_Children(<7 yrs)',
'Num_Older_Children']

for i in cols:
    sns.boxplot(holiday_df[i])
    plt.show()
```


0
50000
100000
150000
200000
Salary


20
30
40
50
60
Age


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Education(In Years)

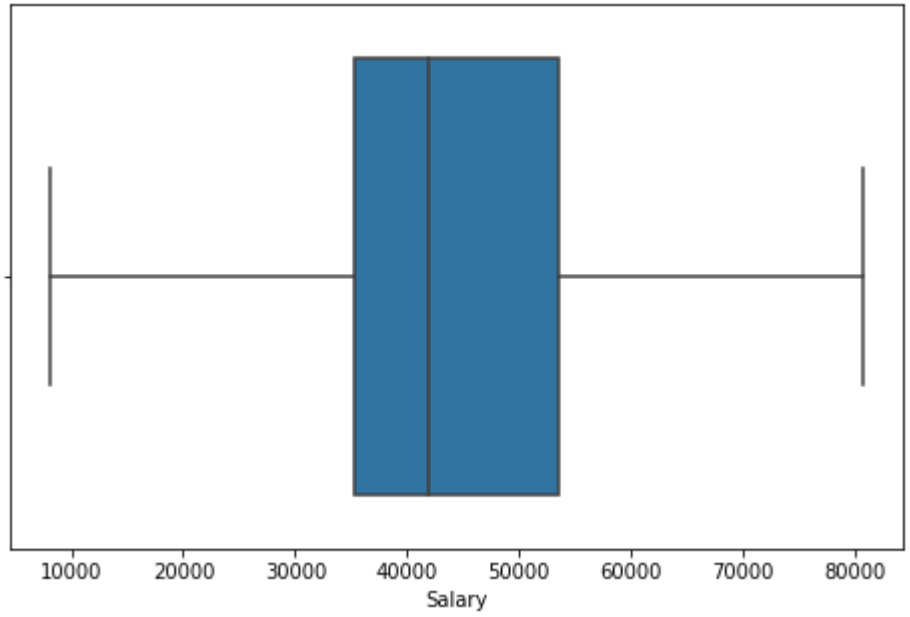

0.0
0.5
1.0
1.5
2.0
2.5
3.0
Num_Young_Children(<7 yrs)

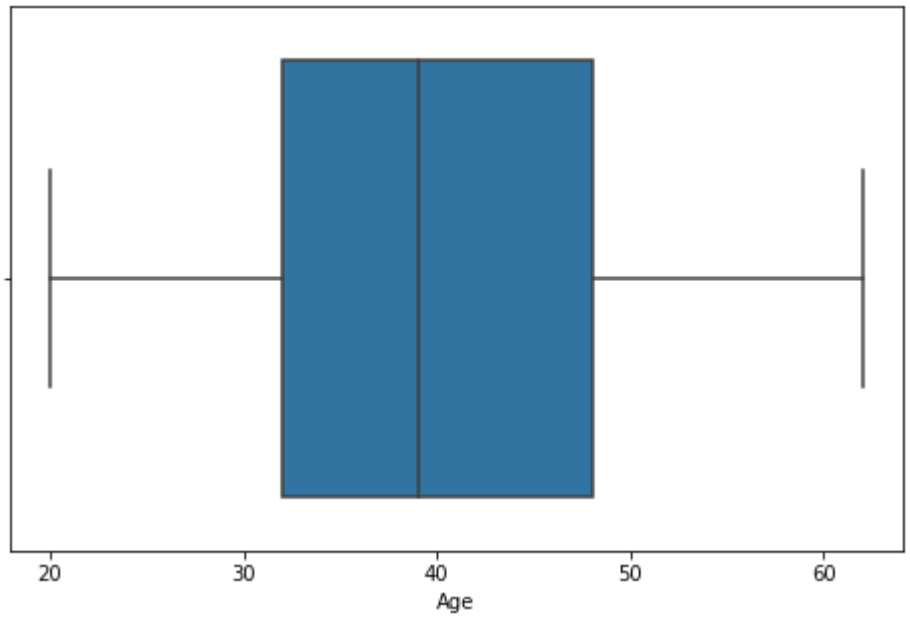


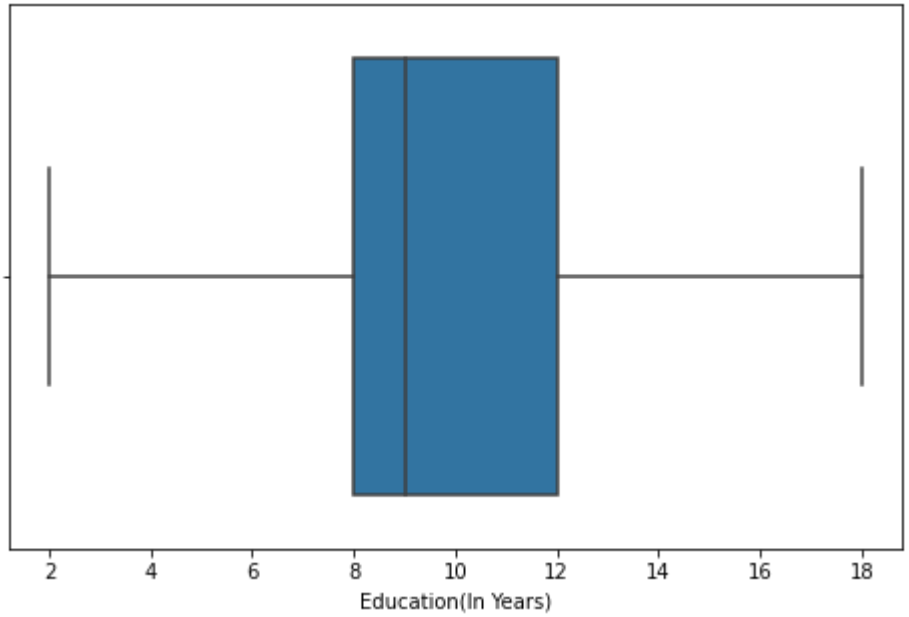
In [115]:
```
#Treating Outliers:

def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

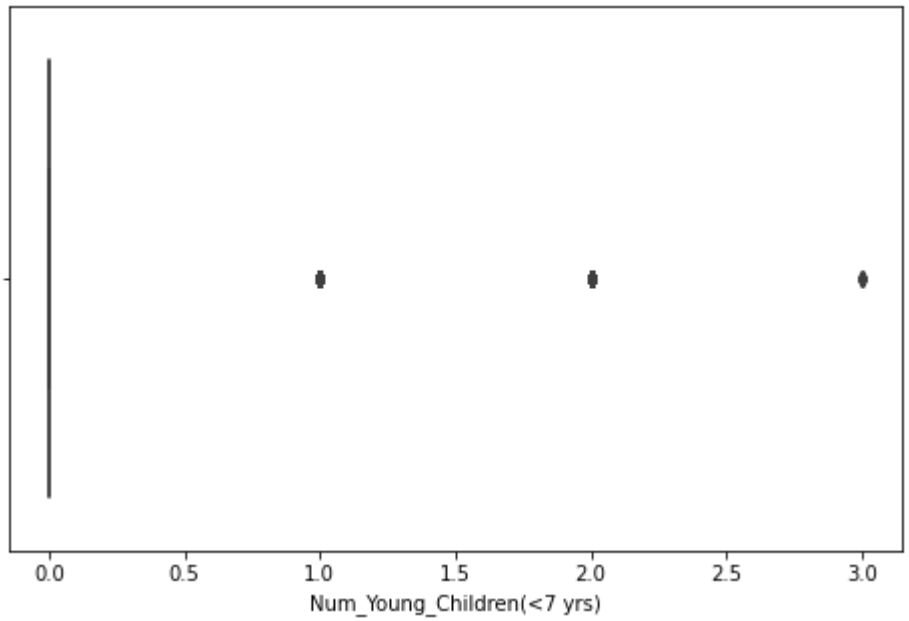
In [116]:
```
cont_variables = ['Salary' ,'Age', 'Education(In Years)', 'Num_Young_Children
(<7 yrs)', 'Num_Older_Children']

for column in holiday_df[cont_variables].columns:
    lr,ur=remove_outlier(holiday_df[column])
    holiday_df[column]=np.where(holiday_df[column]>ur,ur,holiday[column])
    holiday_df[column]=np.where(holiday_df[column]<lr,lr,holiday[column])
```

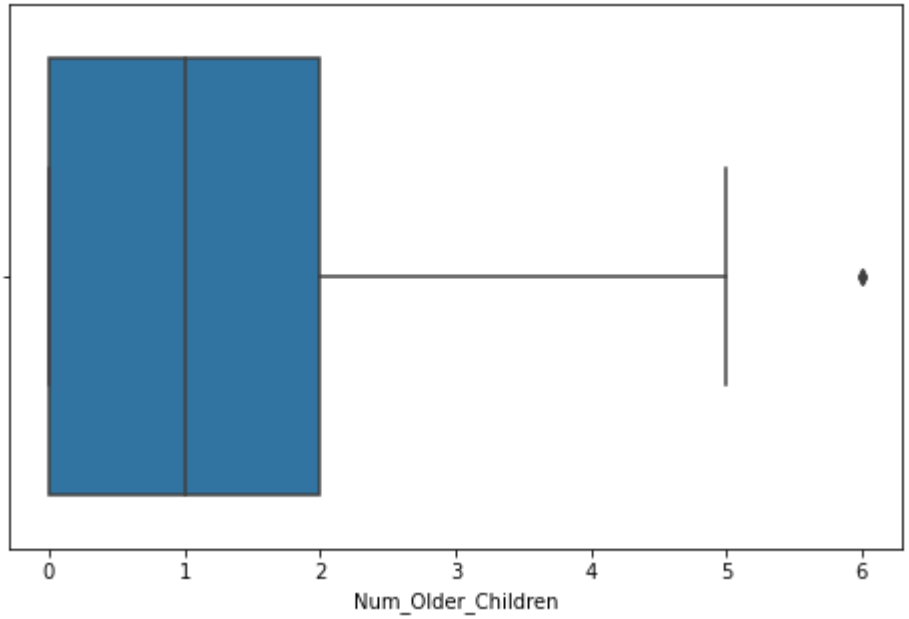
In [117]:
```
for k in cont_variables:
    plt.figure(figsize=(8,5))
    sns.boxplot(holiday_df[k])
    plt.show()
```


10000
20000
30000
40000
50000
60000
70000
80000
Salary


20
30
40
50
60
Age


2
4
6
8
10
12
14
16
18
Education(In Years)


0.0
0.5
1.0
1.5
2.0
2.5
3.0
Num_Young_Children(<7 yrs)



In [118]:
```
for i in holiday_df.columns:
    if holiday_df[i].dtype == 'object':
        print('\n')
        print('Column:',i)
        print(pd.Categorical(holiday_df[i].unique()))
        print(pd.Categorical(holiday_df[i].unique()).codes)
        holiday_df[i] = pd.Categorical(holiday_df[i]).codes
```

```
Column: Holiday_Package
[no, yes]
Categories (2, object): [no, yes]
[0 1]


Column: Foreign
[no, yes]
Categories (2, object): [no, yes]
[0 1]
```

Splitting the data into train and test sets for the LDA model

In [119]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import metrics, model_selection
from sklearn.preprocessing import scale
```

In [120]:
```
X = holiday_df.drop('Holiday_Package',axis=1)
Y = holiday_df.pop('Holiday_Package')
```

In [121]:
```
from sklearn.model_selection import train_test_split,GridSearchCV

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.30,random_sta
te=1,stratify = Y)
```

In [122]:
```
X_train.head()
```

Out[122]:

	Salary	Age	Education(In Years)	Num_Young_Children(<7 yrs)	Num_Older_Children	Foreign
822	38974.0	47.0	12.0	0.0	2.0	1
806	40270.0	33.0	8.0	2.0	0.0	1
323	32573.0	30.0	11.0	1.0	0.0	0
702	43839.0	43.0	11.0	0.0	1.0	1
774	33060.0	40.0	5.0	1.0	1.0	1

Let's build the LDA Model:

In [123]:
```
clf = LinearDiscriminantAnalysis()

LDA_model=clf.fit(X_train,Y_train)
```

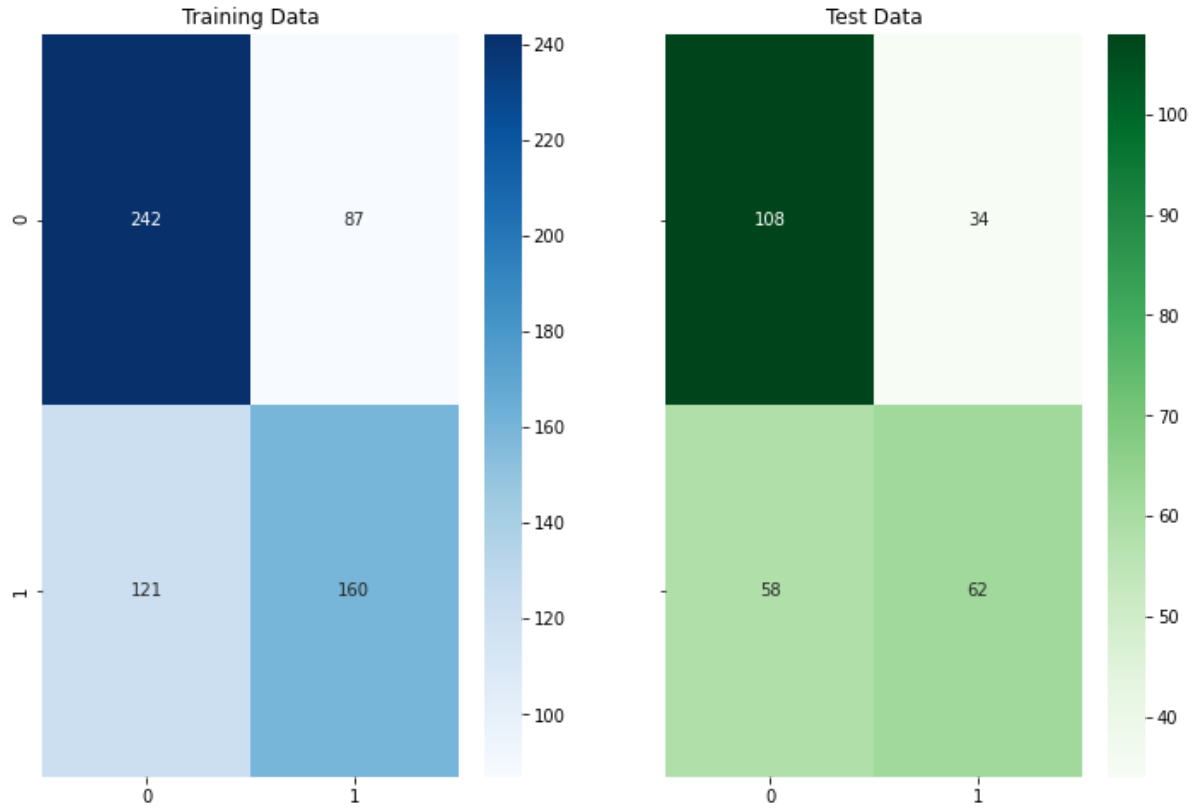
In [124]:
```
#Train data class prediction
#cut off value is 0.5

pred_class_train = LDA_model.predict(X_train)
```

In [125]:
```
# Test data class prediction
#cut-off value is 0.5

pred_class_test = LDA_model.predict(X_test)
```

2.3 Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model Final Model: Compare Both the models and write inference which model is best/optimized.

Comparison between train and test Confusion matrix

In [126]:
```
f,a =  plt.subplots(1,2,sharex=True,sharey=True,squeeze=False)

#Plotting confusion matrix for the different models for the Training Data

plot_0 = sns.heatmap((metrics.confusion_matrix(Y_train,pred_class_train)),anno
t=True,fmt='.5g',cmap='Blues',ax=a[0][0]);
a[0][0].set_title('Training Data')

plot_1 = sns.heatmap((metrics.confusion_matrix(Y_test,pred_class_test)),annot=
True,fmt='.5g',cmap='Greens',ax=a[0][1]);
a[0][1].set_title('Test Data');
```



Classification Report comparison

In [127]:
```
print('Classification Report of the training data:\n\n',metrics.classification
_report(Y_train,pred_class_train),'\n')
print('Classification Report of the test data:\n\n',metrics.classification_rep
ort(Y_test,pred_class_test),'\n')
```

```
Classification Report of the training data:

              precision    recall  f1-score   support

           0       0.67      0.74      0.70       329
           1       0.65      0.57      0.61       281

    accuracy                           0.66       610
   macro avg       0.66      0.65      0.65       610
weighted avg       0.66      0.66      0.66       610


Classification Report of the test data:

              precision    recall  f1-score   support

           0       0.65      0.76      0.70       142
           1       0.65      0.52      0.57       120

    accuracy                           0.65       262
   macro avg       0.65      0.64      0.64       262
weighted avg       0.65      0.65      0.64       262
```

Probability prediction for the training and test data

In [128]:
```
# Training Data Probability Prediction
pred_prob_train = LDA_model.predict_proba(X_train)

# Test Data Probability Prediction
pred_prob_test = LDA_model.predict_proba(X_test)
```

In [129]:
```
pred_prob_train[:,1]
```

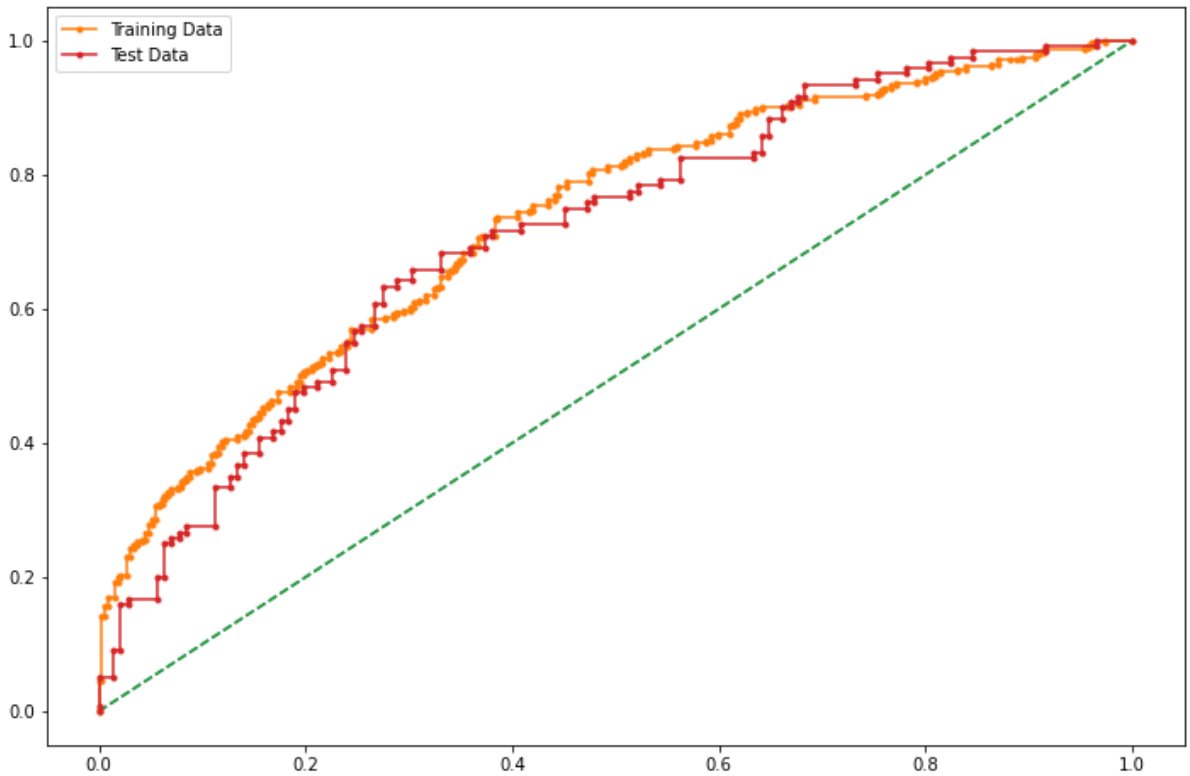
Out[129]:
```
array([0.73551768, 0.28353533, 0.39649058, 0.75922189, 0.47003003,
       0.40177036, 0.36986276, 0.3046299 , 0.60482453, 0.64266073,
       0.23173951, 0.25792362, 0.35900888, 0.04498625, 0.2829094 ,
       0.3643531 , 0.54782489, 0.30634075, 0.58914069, 0.6607117 ,
       0.62795172, 0.26530773, 0.88138953, 0.33411259, 0.08664023,
       0.8271775 , 0.19379564, 0.75733957, 0.53209164, 0.1892203 ,
       0.29258073, 0.3384893 , 0.3769479 , 0.38097484, 0.31056554,
       0.29449197, 0.10973564, 0.56147758, 0.48217964, 0.19474058,
       0.22634057, 0.7985467 , 0.50039742, 0.73794034, 0.7904354 ,
       0.3690023 , 0.30262021, 0.93345931, 0.43331819, 0.72155301,
       0.71677925, 0.45274194, 0.78744639, 0.37233837, 0.19187097,
       0.74091501, 0.25003165, 0.51612541, 0.67432724, 0.35616917,
       0.61002822, 0.52758233, 0.52261599, 0.40993742, 0.57008809,
       0.61654303, 0.12380369, 0.56372873, 0.40105364, 0.27636387,
       0.331064  , 0.74844526, 0.77913611, 0.21862757, 0.44518796,
       0.16803982, 0.49024512, 0.6626503 , 0.59088269, 0.49799747,
       0.4896793 , 0.59011486, 0.82395093, 0.4347367 , 0.64239389,
       0.73447375, 0.22668021, 0.47718918, 0.46780051, 0.30670499,
       0.53035844, 0.68305727, 0.82571857, 0.65940539, 0.50981099,
       0.73072076, 0.22020835, 0.57861358, 0.3906577 , 0.41315668,
       0.39961915, 0.39560786, 0.46210527, 0.52358073, 0.25991578,
       0.44942821, 0.39914056, 0.8268209 , 0.30317932, 0.34790508,
       0.59306708, 0.22910448, 0.68431629, 0.23729384, 0.28605156,
       0.38414222, 0.32996579, 0.602951  , 0.14240438, 0.50113204,
       0.30305369, 0.63206442, 0.7670921 , 0.49630286, 0.42679734,
       0.16259972, 0.21294065, 0.50379895, 0.50858594, 0.39256748,
       0.67108314, 0.29400098, 0.66906493, 0.14334749, 0.58857817,
       0.6341773 , 0.28993688, 0.33123118, 0.14568647, 0.27773523,
       0.73252377, 0.3766964 , 0.20462998, 0.74291733, 0.64088837,
       0.34611449, 0.66417313, 0.68281643, 0.44153834, 0.75802719,
       0.59735368, 0.92826221, 0.48449961, 0.0907812 , 0.64224122,
       0.45079297, 0.06498246, 0.61557792, 0.33989789, 0.54130797,
       0.30267298, 0.42870372, 0.34797438, 0.86021682, 0.61098396,
       0.08225338, 0.38793517, 0.70120088, 0.6000614 , 0.81252069,
       0.42746192, 0.35460828, 0.7482434 , 0.36201143, 0.32684319,
       0.4043595 , 0.5190445 , 0.37552252, 0.58866101, 0.29713676,
       0.37955487, 0.48804455, 0.88707586, 0.25145929, 0.38030449,
       0.72882309, 0.29958103, 0.3852379 , 0.40108409, 0.12454965,
       0.55881148, 0.43905053, 0.64610966, 0.51821849, 0.23201163,
       0.55470864, 0.77338469, 0.59854346, 0.80261939, 0.36876583,
       0.24913198, 0.28510202, 0.46447558, 0.54693731, 0.65309467,
       0.45631893, 0.21729382, 0.15431768, 0.85066908, 0.29997552,
       0.67300797, 0.31651641, 0.37950403, 0.52367112, 0.67000344,
       0.43079366, 0.56028724, 0.74648037, 0.50038392, 0.44137556,
       0.40563702, 0.24317694, 0.16001581, 0.64767069, 0.36223484,
       0.66290415, 0.21671679, 0.59801832, 0.3276567 , 0.44512562,
       0.67777101, 0.55704479, 0.35809111, 0.40109792, 0.6558441 ,
       0.29290504, 0.82652622, 0.81036194, 0.86649751, 0.2214361 ,
       0.57545803, 0.78589255, 0.64344643, 0.74449028, 0.15948405,
       0.38231562, 0.52517014, 0.20610503, 0.5404816 , 0.44478123,
       0.48247476, 0.06261534, 0.37742779, 0.33694909, 0.18903175,
       0.28128231, 0.39951162, 0.178948  , 0.49905107, 0.71146773,
       0.45264288, 0.81757548, 0.41487134, 0.26031229, 0.45089437,
       0.529018  , 0.79004835, 0.41169303, 0.3181271 , 0.43483395,
       0.31641471, 0.72935994, 0.45260201, 0.69136321, 0.6633681 ,
       0.19395577, 0.50393907, 0.27772169, 0.56116769, 0.23488687,
       0.77079976, 0.59195866, 0.43362903, 0.26277843, 0.4580244 ,
```

```
0.25032838, 0.13844701, 0.34568608, 0.49698094, 0.18566966,
0.35109544, 0.49294473, 0.47641486, 0.73795386, 0.42161149,
0.06229571, 0.1581428 , 0.35594836, 0.18764244, 0.45298584,
0.41005638, 0.73974108, 0.14018691, 0.40822672, 0.51822275,
0.78178486, 0.32592049, 0.60015823, 0.34085242, 0.45072088,
0.27961609, 0.66436781, 0.78159235, 0.34073854, 0.58912261,
0.35122329, 0.36765515, 0.2978601 , 0.32792071, 0.21413565,
0.76658471, 0.19984202, 0.32843256, 0.79500546, 0.20882333,
0.34371142, 0.30553688, 0.78075944, 0.41715537, 0.7498246 ,
0.47796229, 0.28727224, 0.76949624, 0.71253017, 0.31815105,
0.35291993, 0.06077497, 0.55544254, 0.88503212, 0.70564725,
0.60487399, 0.74261114, 0.66839085, 0.13547501, 0.51459181,
0.43718177, 0.45730564, 0.43374086, 0.7164607 , 0.70925743,
0.54295621, 0.37142744, 0.68668209, 0.29574937, 0.29265307,
0.68042303, 0.2062512 , 0.43504539, 0.32809752, 0.41441835,
0.59907657, 0.16758297, 0.45930385, 0.25896523, 0.42612595,
0.55151653, 0.38942437, 0.16883373, 0.85037391, 0.67092639,
0.2732217 , 0.83344785, 0.36160728, 0.17153013, 0.82911233,
0.22804333, 0.3445843 , 0.34686932, 0.40436238, 0.3159984 ,
0.51340754, 0.71483811, 0.14779098, 0.5649874 , 0.1244422 ,
0.24627663, 0.23394929, 0.77865277, 0.04018553, 0.55330876,
0.26319744, 0.05045702, 0.78165496, 0.73895935, 0.68959338,
0.72893529, 0.38533768, 0.39265905, 0.52118353, 0.82775083,
0.48604987, 0.22758067, 0.36803216, 0.28102472, 0.252465  ,
0.44859804, 0.11323111, 0.66903538, 0.458799  , 0.49943112,
0.25577021, 0.32429741, 0.35336564, 0.23416983, 0.73354456,
0.34560799, 0.13883419, 0.63741426, 0.2469326 , 0.36837838,
0.76037877, 0.17733507, 0.85323086, 0.87678412, 0.68240595,
0.37708493, 0.33849918, 0.24016445, 0.40329575, 0.6763881 ,
0.73002024, 0.8031362 , 0.45760962, 0.80904281, 0.62974268,
0.32862734, 0.61148053, 0.37814081, 0.32664153, 0.72999864,
0.42233413, 0.39623996, 0.82461924, 0.68372526, 0.46182588,
0.43092682, 0.2630122 , 0.72058804, 0.54901725, 0.51649024,
0.75278144, 0.23580589, 0.53541603, 0.46750567, 0.38717556,
0.6476411 , 0.58123497, 0.30093   , 0.33515296, 0.37541846,
0.23718207, 0.71370908, 0.38295121, 0.53011156, 0.5422955 ,
0.87277544, 0.56902952, 0.58809912, 0.36916778, 0.14604874,
0.51499491, 0.30587331, 0.72363803, 0.41349651, 0.52088866,
0.17353901, 0.39613308, 0.25196479, 0.58475031, 0.34827362,
0.72551323, 0.30119211, 0.6164726 , 0.29419961, 0.2832862 ,
0.16740438, 0.32428109, 0.31185739, 0.41931201, 0.59870159,
0.31656484, 0.549763  , 0.42439067, 0.72573306, 0.81893183,
0.17316527, 0.69808572, 0.28305666, 0.37258725, 0.31219309,
0.77305294, 0.56576443, 0.234862  , 0.3664639 , 0.31918383,
0.21350342, 0.29916519, 0.77873648, 0.86391346, 0.32544912,
0.15384096, 0.70544338, 0.67625912, 0.23091696, 0.86579177,
0.57043938, 0.59357384, 0.79466617, 0.31747845, 0.64463952,
0.48329349, 0.5099068 , 0.27973536, 0.80364772, 0.39533764,
0.68130403, 0.17733507, 0.5052279 , 0.3444913 , 0.50045267,
0.4492691 , 0.40699094, 0.49811997, 0.25602291, 0.18761371,
0.51329831, 0.3443562 , 0.2555839 , 0.62433929, 0.18923398,
0.19388512, 0.40921973, 0.27644011, 0.34251839, 0.54038819,
0.28123359, 0.22273066, 0.45304983, 0.30297484, 0.52462392,
0.56874132, 0.56930422, 0.68684871, 0.32402954, 0.377056  ,
0.40126654, 0.66838248, 0.23268841, 0.15452033, 0.41576428,
0.13536216, 0.12989277, 0.52621923, 0.1023751 , 0.76762203,
0.29566514, 0.6392037 , 0.72654237, 0.83150954, 0.48240629,
```

```
0.77314358, 0.33739508, 0.23269807, 0.61970885, 0.2994335 ,
0.44630473, 0.37404461, 0.11829941, 0.11142323, 0.5464569 ,
0.35865226, 0.30544221, 0.50798801, 0.58734033, 0.4602424 ,
0.12033415, 0.68150639, 0.24414107, 0.26735009, 0.5078248 ,
0.32653896, 0.33220615, 0.62948944, 0.45798082, 0.38751535,
0.28106622, 0.4521367 , 0.46722676, 0.15585761, 0.34146489,
0.37295646, 0.2749017 , 0.86486816, 0.50710463, 0.78215575,
0.38324091, 0.43486903, 0.53636009, 0.74207132, 0.48480572])
```

In [130]:
```
# AUC and ROC for the training data

# calculate AUC
auc = metrics.roc_auc_score(Y_train,pred_prob_train[:,1])
print('AUC for the Training Data: %.3f' % auc)

#  calculate roc curve
fpr, tpr, thresholds = metrics.roc_curve(Y_train,pred_prob_train[:,1])
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.',label = 'Training Data')


# AUC and ROC for the test data

# calculate AUC
auc = metrics.roc_auc_score(Y_test,pred_prob_test[:,1])
print('AUC for the Test Data: %.3f' % auc)

#  calculate roc curve
fpr, tpr, thresholds = metrics.roc_curve(Y_test,pred_prob_test[:,1])
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.',label='Test Data')
# show the plot
plt.legend(loc='best')
plt.show()
```

```
AUC for the Training Data: 0.731
AUC for the Test Data: 0.713
```



Conclusion

The model accuracy on the training data is 66% and for the testing data is about 65%. This model is affected by a class imbalance problem. We will build the model again with different cut off values to get the better accuracy.

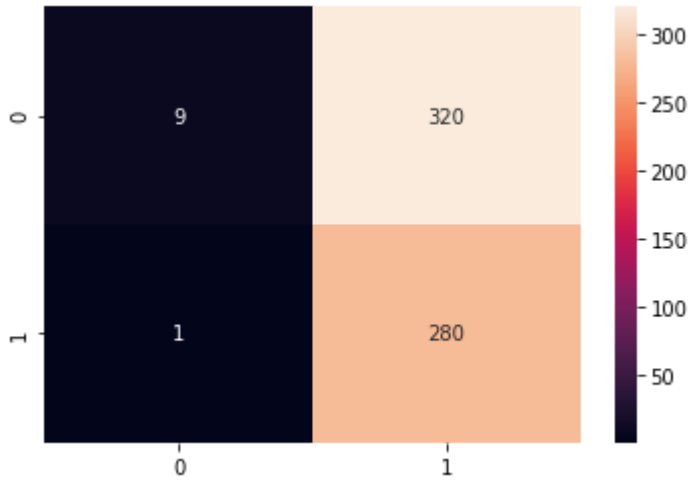
We will build the model again with different cut off values to get the better accuracy.

In [131]:
```
#defining the cut-off value of our choice

for j in np.arange(0.1,1,0.1):
    custom_prob = j
    custom_cutoff_data=[]
    for i in range(0,len(Y_train)): #defining a loop for the length of the tes
t data
        if np.array(pred_prob_train[:,1])[i] > custom_prob: #issuing a conditi
on for our probability values to be #greater than the custom cutoff value

            a=1#if the probability values are greater than the custom cutoff t
hen the value should be 1
        else:
            a=0#if the probability values are less than the custom cutoff then
the value should be 0
        custom_cutoff_data.append(a)#adding either 1 or 0 based on the conditi
on to the end of the list defined by us
    print(round(j,3),'\n')
    print('Accuracy Score',round(metrics.accuracy_score(Y_train,custom_cutoff_
data),4))
    print('F1 Score',round(metrics.f1_score(Y_train,custom_cutoff_data),4),'\n
')
    plt.figure(figsize=(6,4))
    print('Confusion Matrix')
    sns.heatmap(metrics.confusion_matrix(Y_train,custom_cutoff_data),annot=Tru
e,fmt='.4g'),'\n\n'
    plt.show();
```

0.1

Accuracy Score 0.4738
F1 Score 0.6356

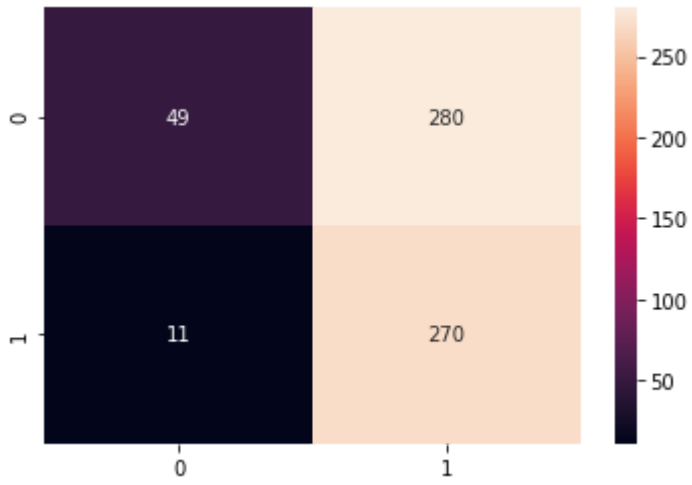
Confusion Matrix



0.2

Accuracy Score 0.523
F1 Score 0.6498

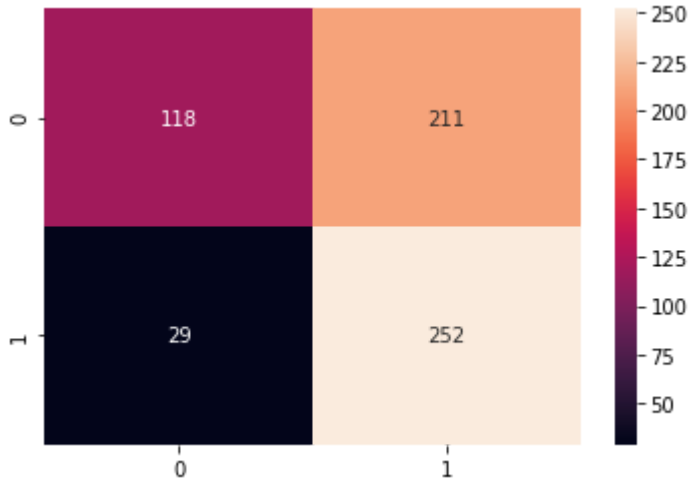
Confusion Matrix



0.3

Accuracy Score 0.6066
F1 Score 0.6774

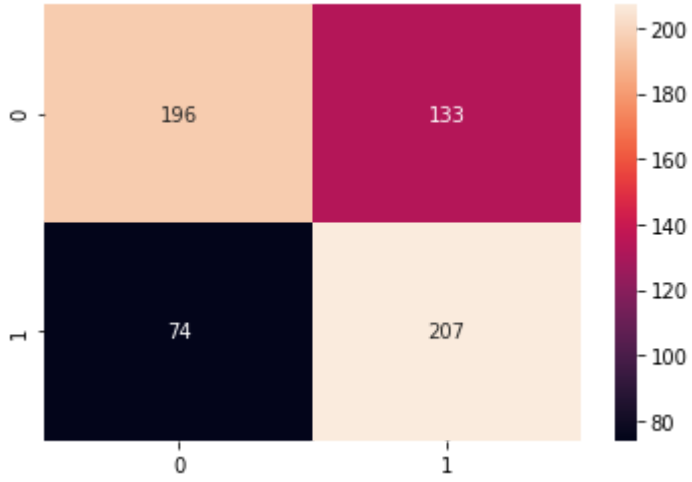
Confusion Matrix



0.4

Accuracy Score 0.6607
F1 Score 0.6667

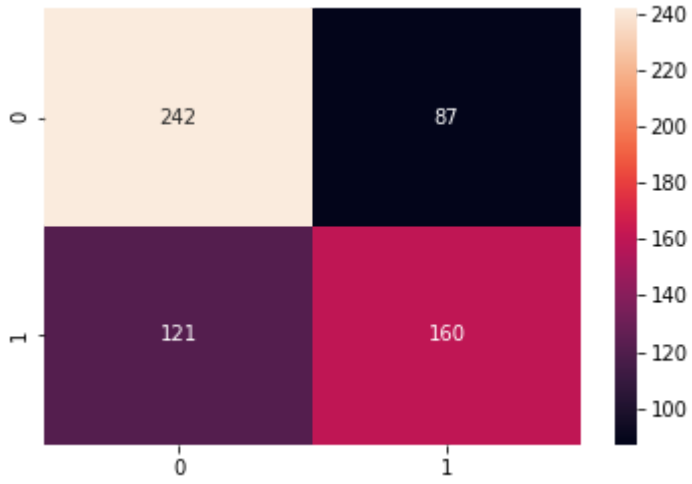
Confusion Matrix



0.5

Accuracy Score 0.659
F1 Score 0.6061

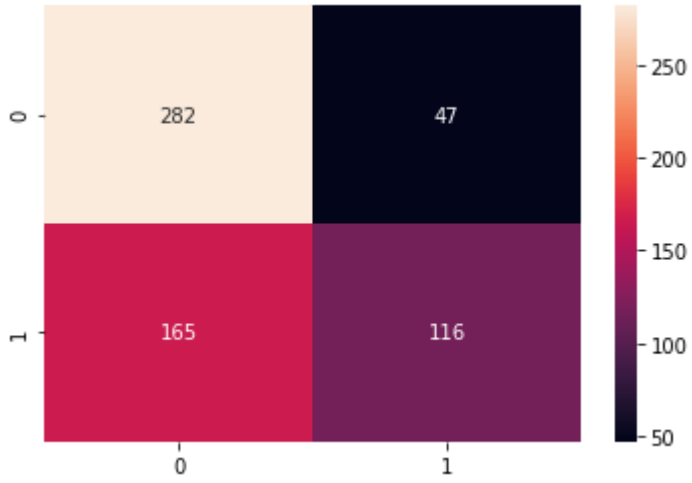
Confusion Matrix



0.6

Accuracy Score 0.6525
F1 Score 0.5225

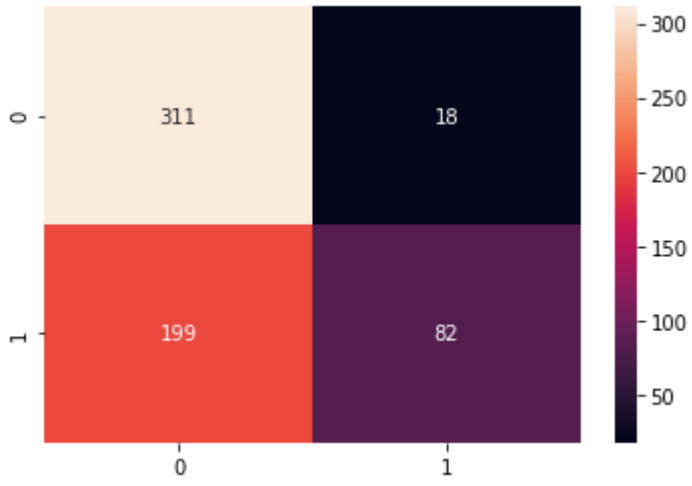
Confusion Matrix



0.7

Accuracy Score 0.6443
F1 Score 0.4304

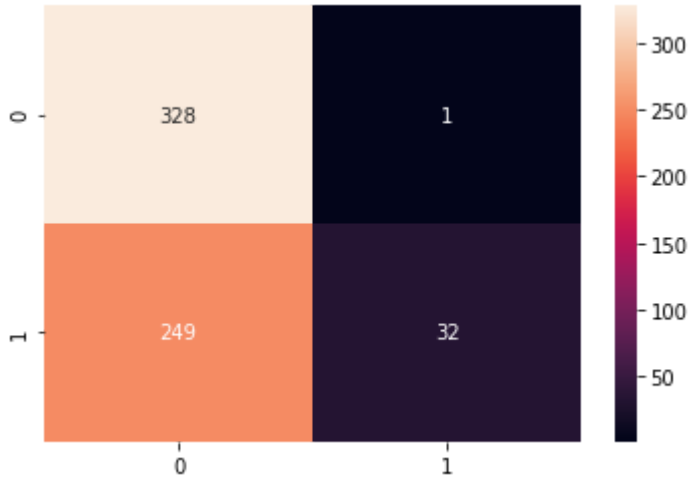
Confusion Matrix



0.8

Accuracy Score 0.5902
F1 Score 0.2038

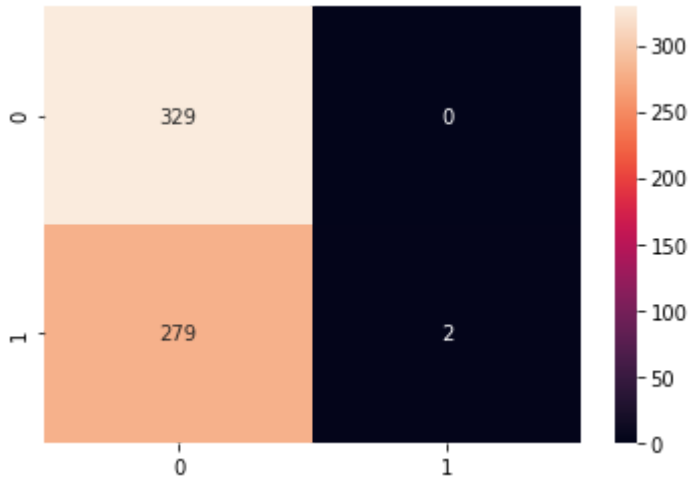
Confusion Matrix



0.9

Accuracy Score 0.5426
F1 Score 0.0141

Confusion Matrix

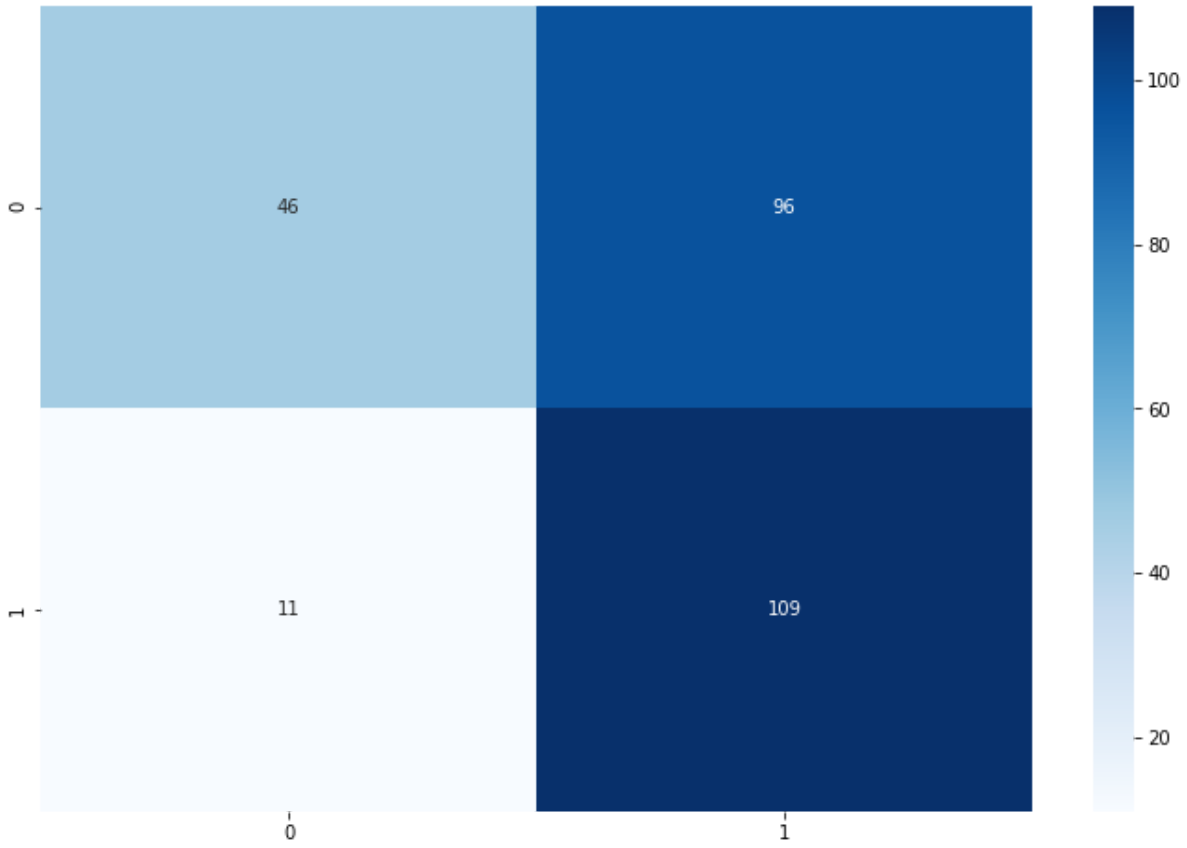


Conclusion:

We can see that 0.3 and 0.4 gives better accuracy than the rest of the custom cut-off values. But 0.3 cut-off gives us the best 'f1-score' of around 0.67. Here, we will take the cut-off as 0.3 to get the optimum 'f1' score.

In [132]:
```
### Let us evaluate the predictions of the test data using these cut-off value
s:

#Predicting the classes on the test data

data_pred_custom_cutoff=[]
for i in range(0,len(pred_prob_test[:,1])):
    if np.array(pred_prob_test[:,1])[i]>0.3:
        a=1
    else:
        a=0
    data_pred_custom_cutoff.append(a)
```

In [133]:
```
sns.heatmap((metrics.confusion_matrix(Y_test,data_pred_custom_cutoff)),annot=True,fmt='.5g',cmap='Blues');
```



In [134]:
```
print('Classification Report of the default cut-off test data:\n\n',metrics.classification_report(Y_test,pred_class_test),'\n\n\n')
print('Classification Report of the custom cut-off test data:\n\n',metrics.classification_report(Y_test,data_pred_custom_cutoff),'\n')
```

```
Classification Report of the default cut-off test data:

              precision    recall  f1-score   support

           0       0.65      0.76      0.70       142
           1       0.65      0.52      0.57       120

    accuracy                           0.65       262
   macro avg       0.65      0.64      0.64       262
weighted avg       0.65      0.65      0.64       262



Classification Report of the custom cut-off test data:

              precision    recall  f1-score   support

           0       0.81      0.32      0.46       142
           1       0.53      0.91      0.67       120

    accuracy                           0.59       262
   macro avg       0.67      0.62      0.57       262
weighted avg       0.68      0.59      0.56       262
```

As we can see accuracy is almost the same, but f1 score is better in LDA model.

2.4 Inference: Basis on these predictions, what are the insights and recommendations.

Insights and Observations:

1. As per the statistics we get from exploratory data analysis, we can see that aged people are not interested in holiday package and young people are opting for more holiday trips.
2. We can see from the holiday vs salary plot that the employess with the salary less than 150000 have always opted for the holiday package in comparison with the employees with salary greater than 150000.
3. Employees with 8 to 12 years of formal education with salary between 250000 to 600000 took more holiday package among all others.
4. Age , Salary and years of education of the employees are the crucial features in the role of prediction.
5. The Logistic regression model performs better in terms of Accuracy than that of LDA model, but LDA model gives best f1 score around 0.66 and that is much better than that of logistic regression model, a good F1 score means that we have low false positives and low false negatives.

Recommendation:

Employees having young children less than 3 are more to opt for holiday packages but employees having older children are not going for holidays, so we can provide them family trip tour packages with the combination of places including hill stations and spiritual places, so that the parents and their children both can take the package into consideration and can plan their trip.

Employees with salary above than 150000 seems to be more busy with their work or with their business, so I recommend that we should give them a short holiday trip pacakges or weekend trip packages which includes some nearby places so that they can think about planning a small holiday trip with their loved ones or with their friends.

Obviously they might be older in the age, so we can add spiritual or holy places like some famous temples in their state, and also we can provide some special concessions to attract them.

We can also understand from the insights we got from the dataset that the employees which falls under the category of age group 25 years to 45 years are more curious and adventurous and also capable to manage their financial conditions better than the employees of the age group more than 50 years, so my recommendations to attract such employees more that we can offer them holiday trip package which includes some attractive places like beach, waterfalls, tracking. We can also add some water sport activities or resorts with night staying facilties.

We can also provide them the pre-booking offers for packages which include some special events that occurs every year

__ THE END __