Business Report:

Predictive Modelling.

Problem 1: Linear Regression.

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:

Variable	Description
Carat	Carat weight of the cubic zirconia.
Cut	Describe the cut quality of the cubic zirconia. Quality is increasing order Fair, Good, Very Good, Premium, Ideal.
Color	Colour of the cubic zirconia.With D being the best and J the worst.
Clarity	cubic zirconia Clarity refers to the absence of the Inclusions and Blemishes. (In order from Best to Worst, FL = flawless, I3= level 3 inclusions) FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3
Depth	The Height of a cubic zirconia, measured from the Culet to the table, divided by its average Girdle Diameter.
Table	The Width of the cubic zirconia's Table expressed as a Percentage of its Average Diameter.
Price	the Price of the cubic zirconia.
X	Length of the cubic zirconia in mm.
Y	Width of the cubic zirconia in mm.
Z	Height of the cubic zirconia in mm.

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 [1]:

```
# importing the necessary libraries.

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:
numpy.ufunc size changed, may indicate binary incompatibility. Expected 19
2 from C header, got 216 from PyObject
  return f(*args, **kwds)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:
numpy.ufunc size changed, may indicate binary incompatibility. Expected 19
2 from C header, got 216 from PyObject
  return f(*args, **kwds)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:
numpy.ufunc size changed, may indicate binary incompatibility. Expected 19
2 from C header, got 216 from PyObject
  return f(*args, **kwds)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:
numpy.ufunc size changed, may indicate binary incompatibility. Expected 19
2 from C header, got 216 from PyObject
  return f(*args, **kwds)
```

In [2]:

```
# Reading the data set.

df_1 = pd.read_csv (r'E:\Great Learning\Projects\Predictive Modelling\Data Sets\cubic_z
irconia.csv')
# checking head of the data

df_1.head(8)
```

Out[2]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779
5	6	1.02	Ideal	D	VS2	61.5	56.0	6.46	6.49	3.99	9502
6	7	1.01	Good	H	SI1	63.7	60.0	6.35	6.30	4.03	4836
7	8	0.50	Premium	E	SI1	61.5	62.0	5.09	5.06	3.12	1415

Exploratory Data Analysis (EDA):

In [3]:

```
df_1.shape
```

Out[3]:

```
(26967, 11)
```

In [4]:

```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 11 columns):
Unnamed: 0    26967 non-null int64
carat         26967 non-null float64
cut           26967 non-null object
color         26967 non-null object
clarity       26967 non-null object
depth         26270 non-null float64
table         26967 non-null float64
x             26967 non-null float64
y             26967 non-null float64
z             26967 non-null float64
price         26967 non-null int64
dtypes: float64(6), int64(2), object(3)
memory usage: 2.3+ MB
```

In [5]:

```
df_1.dtypes
```

Out[5]:

```
Unnamed: 0      int64
carat         float64
cut            object
color          object
clarity        object
depth         float64
table         float64
x             float64
y             float64
z             float64
price           int64
dtype: object
```

There are 3 Categories which are OBJECT and remaining are Integers/Float, for those Categories which are Object we will need to convert them into the Integers/Float.

In [6]:

```
df_1.describe (include='all').T
```

Out[6]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	
Unnamed: 0	26967	NaN	NaN	NaN	13484	7784.85	1	6742.5	13484	20225.5	26
carat	26967	NaN	NaN	NaN	0.798375	0.477745	0.2	0.4	0.7	1.05	
cut	26967	5	Ideal	10816	NaN	NaN	NaN	NaN	NaN	NaN	
color	26967	7	G	5661	NaN	NaN	NaN	NaN	NaN	NaN	
clarity	26967	8	SI1	6571	NaN	NaN	NaN	NaN	NaN	NaN	
depth	26270	NaN	NaN	NaN	61.7451	1.41286	50.8	61	61.8	62.5	
table	26967	NaN	NaN	NaN	57.4561	2.23207	49	56	57	59	
x	26967	NaN	NaN	NaN	5.72985	1.12852	0	4.71	5.69	6.55	1
y	26967	NaN	NaN	NaN	5.73357	1.16606	0	4.71	5.71	6.54	
z	26967	NaN	NaN	NaN	3.53806	0.720624	0	2.9	3.52	4.04	
price	26967	NaN	NaN	NaN	3939.52	4024.86	326	945	2375	5360	18

In [7]:

```
df_1.isnull().sum() # checking missing/null values.
```

Out[7]:

```
Unnamed: 0      0
carat           0
cut             0
color           0
clarity         0
depth         697
table           0
x               0
y               0
z               0
price           0
dtype: int64
```

check ***Almost 2 % of missing values in the entire data set which is only present in Column "depth", which we can treat by imputing its mean or dropping.

In [9]:

```
dups = df_1.duplicated() # checking for duplicates.
print ('Number of Duplicates in Data are %d' % (dups.sum()))
```

```
Number of Duplicates in Data are 0
```

In [14]:

```
# checking for unique values in categorical column.

for column in df_1.columns:
    if df_1 [column].dtype == 'object':
        print (column.upper(), ':', df_1 [column].nunique())
        print (df_1 [column].value_counts().sort_values())
        print('\n')
```

```
CUT : 5
Fair           781
Good          2441
Very Good     6030
Premium       6899
Ideal        10816
Name: cut, dtype: int64


COLOR : 7
J    1443
I    2771
D    3344
H    4102
F    4729
E    4917
G    5661
Name: color, dtype: int64


CLARITY : 8
I1       365
IF       894
VVS1    1839
VVS2    2531
VS1     4093
SI2     4575
VS2     6099
SI1     6571
Name: clarity, dtype: int64
```

Observations: We have 11 columns (including the Unnames) Carat, cut, Color, Clarity, Depth, Table, x,y,z, price.

Rows in the entire data is 26967.

There are Missing values (we need to treat them).

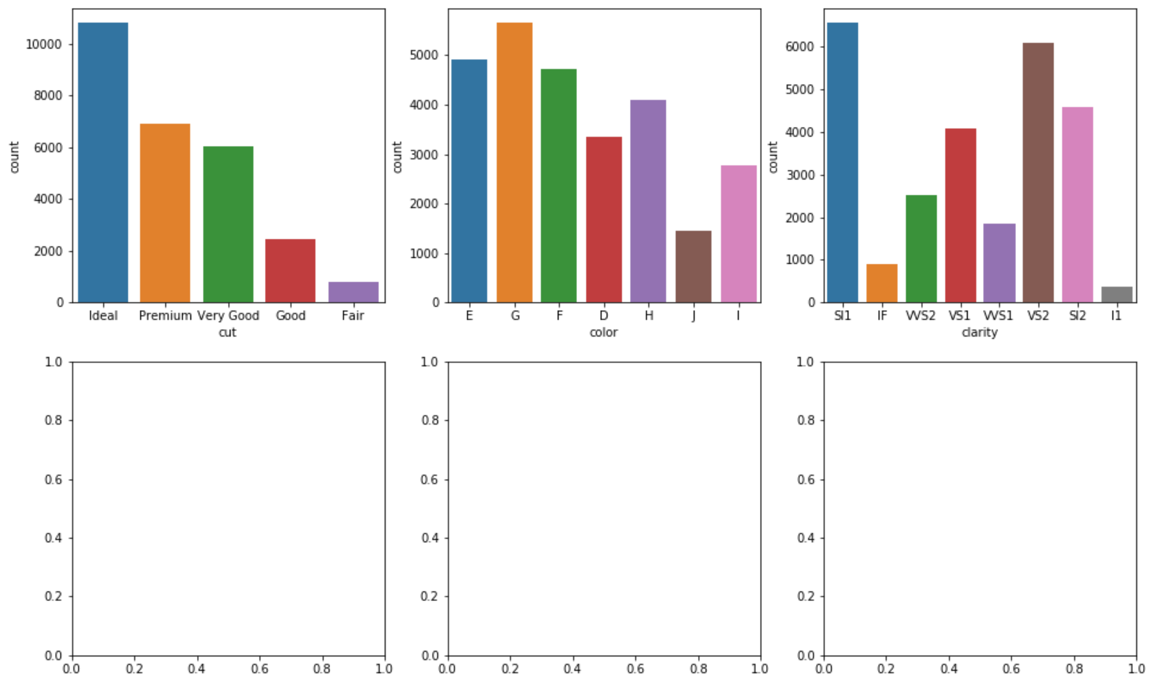
There are No Duplicates.

Also there are unique values in the Categorical columns (We will treat them further).

Univariate and Bi-variate Analysis.

In [36]:

```
# for object category.

fig,ax = plt.subplots(2,3,figsize=(16,10))
sns.countplot ('cut', data= df_1, ax=ax [0][0]);
sns.countplot ('color', data= df_1, ax=ax [0][1]);
sns.countplot ('clarity', data= df_1, ax=ax [0][2]);
```



Cut :

As being the the Best one is Ideal and least one is Fair, we can see that Quality of Cubic Zirconia is the Ideal one as its aroung more than 11,000 and Fair is around 400.

Very Good and Premium quality is somewhere equal (Need to understand why its not being marked as one by the company).

Color :

We can Rank as : D - 1 Has the Fifth maximum count

E - 2 Has the second maximum count (E and F can be marked as one as count is very much similar between them)

F - 3 Has the third maximum count

G - 4 Has the first maximum count

H - 5 Has the fourth maximum count

I - 6 Has the sixth maximum count

J - 7 Has the seventh maximum count.

Clarity :

We can Rank as : FL - 1 (not present as in practical world no mineral is flawless)

IF - 2 (Around 900)

VVS1 - 3 (Around 1800)

VVS2 - 4 (Around 2400)

VS1 - 5 (Around 4000)

VS2 - 6 (Around 6000)

SI1 - 7 (Around 6400) has the maximum clarity or purity.

SI2 - 8 (Around 4200)

I1 - 9 (Around 200)

I2 - 10 (Not present)

I3 - 11 (Not present)

In [39]:

```
# lets create a copy of our original data set so that we can try our multiple combinati
ons.

df2 = df_1.copy ()
df2.head()
```

Out[39]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

In [40]:

```
# dropping Unnamed: 0
df2 = df2.drop ('Unnamed: 0',axis=1)
df2.head()
```

Out[40]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

In [41]:

```
for feature in df2.columns:
    if df2 [feature].dtype=='object':
        df2 [feature]=pd.Categorical (df2[feature]).codes # coverting all dtypes in Int
egers.
```

In [42]:

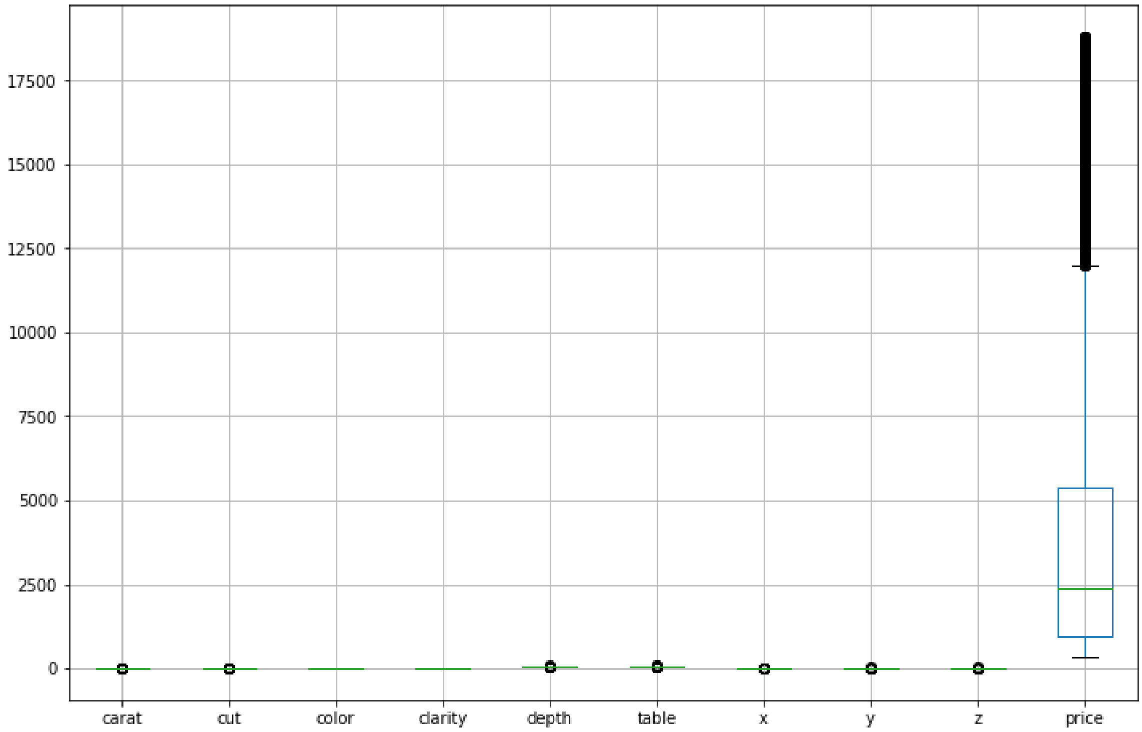
```
df2.dtypes
```

Out[42]:

```
carat       float64
cut            int8
color          int8
clarity        int8
depth       float64
table       float64
x           float64
y           float64
z           float64
price         int64
dtype: object
```

In [43]:

```
df2.boxplot (figsize=(12,8));
```



In [46]:

```
df2.describe()
```

Out[46]:

	carat	cut	color	clarity	depth	table
count	26967.000000	26967.000000	26967.000000	26967.000000	26270.000000	26967.000000
mean	0.798375	2.554604	2.606111	3.833537	61.745147	57.456080
std	0.477745	1.024243	1.705992	1.724904	1.412860	2.232068
min	0.200000	0.000000	0.000000	0.000000	50.800000	49.000000
25%	0.400000	2.000000	1.000000	2.000000	61.000000	56.000000
50%	0.700000	2.000000	3.000000	4.000000	61.800000	57.000000
75%	1.050000	3.000000	4.000000	5.000000	62.500000	59.000000
max	4.500000	4.000000	6.000000	7.000000	73.600000	79.000000

As we can see that almost all the columns have an outlier when we compared min,75% and max values. Also, we can assume that Maximum value is not far from the 75 % so its good when we treat the outlier by bringing all of them into Maximum Value.

In [47]:

```
df2.mean()
```

Out[47]:

```
carat          0.798375
cut            2.554604
color          2.606111
clarity        3.833537
depth         61.745147
table         57.456080
x              5.729854
y              5.733569
z              3.538057
price       3939.518115
dtype: float64
```

In [49]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 10 columns):
carat      26967 non-null float64
cut        26967 non-null int8
color      26967 non-null int8
clarity    26967 non-null int8
depth      26270 non-null float64
table      26967 non-null float64
x          26967 non-null float64
y          26967 non-null float64
z          26967 non-null float64
price      26967 non-null int64
dtypes: float64(6), int64(1), int8(3)
memory usage: 1.5 MB
```

In [48]:

```
df2.isnull().sum()
```

Out[48]:

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

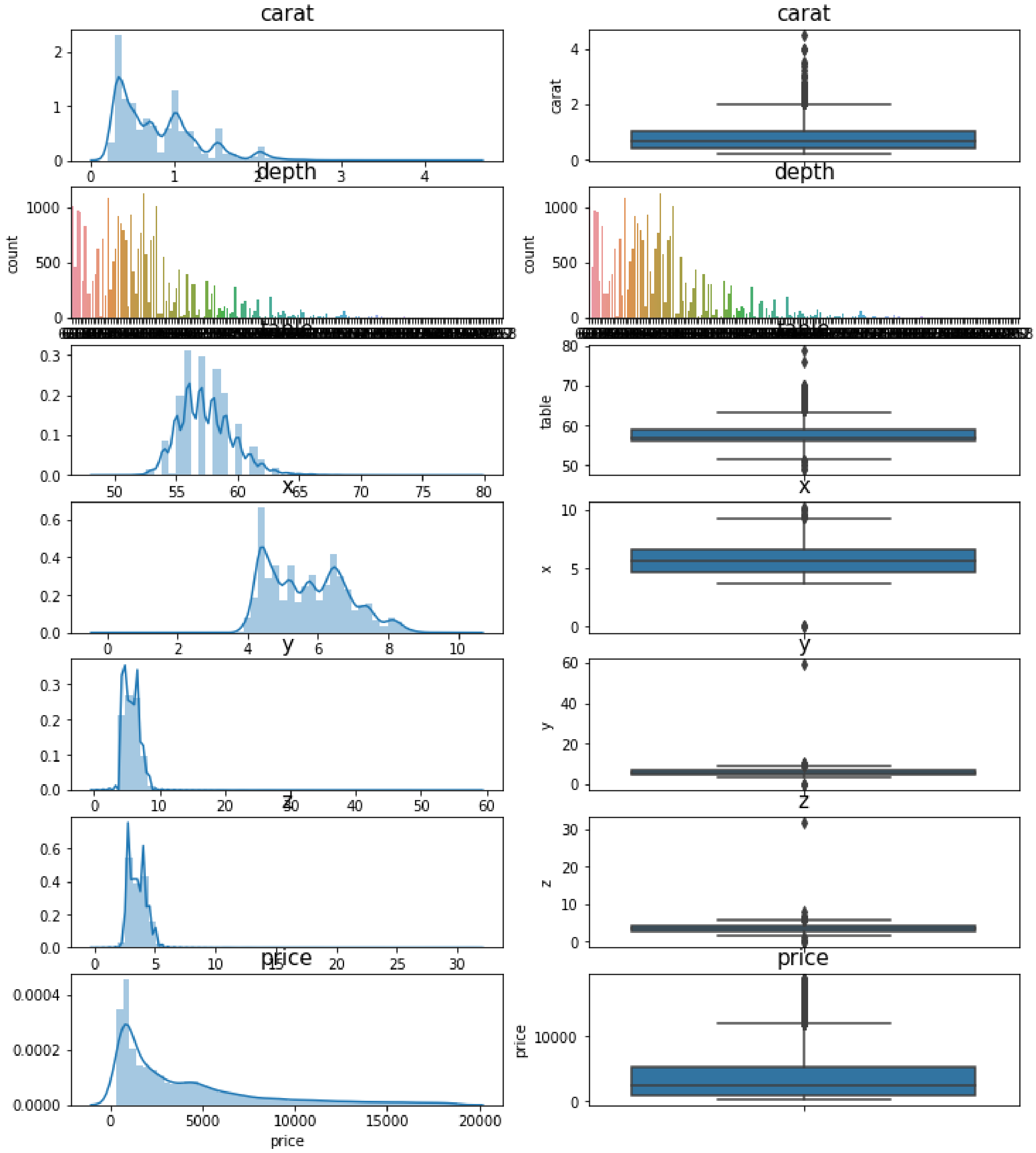
As we saw that there were 697 missing values for one attribute depth, which was 2.5 % of it which we were able to drop but we have imputed it by its mean.

In [60]:

```
# for integers category.
# carat, depth, table, x,y,z,price,Unnamed: 0

fig , axes = plt.subplots (nrows=7,ncols=2)
fig.set_size_inches (12,14)

r = sns.distplot (df2 ['carat'],ax=axes [0][0]);
r.set_title ('carat',fontsize=15)
r = sns.boxplot (df2 ['carat'],orient='v',ax=axes [0][1]);
r.set_title ('carat',fontsize=15)

r = sns.countplot (df2 ['depth'],ax=axes [1][0]);
r.set_title ('depth',fontsize=15)
r = sns.countplot (df2 ['depth'],orient='v',ax=axes [1][1]);
r.set_title ('depth',fontsize=15)

r = sns.distplot (df2 ['table'],ax=axes [2][0]);
r.set_title ('table',fontsize=15)
r = sns.boxplot (df2 ['table'],orient='v',ax=axes [2][1]);
r.set_title ('table',fontsize=15)

r = sns.distplot (df2 ['x'],ax=axes [3][0]);
r.set_title ('x',fontsize=15)
r = sns.boxplot (df2 ['x'],orient='v',ax=axes [3][1]);
r.set_title ('x',fontsize=15)

r = sns.distplot (df2 ['y'],ax=axes [4][0]);
r.set_title ('y',fontsize=15)
r = sns.boxplot (df2 ['y'],orient='v',ax=axes [4][1]);
r.set_title ('y',fontsize=15)

r = sns.distplot (df2 ['z'],ax=axes [5][0]);
r.set_title ('z',fontsize=15)
r = sns.boxplot (df2 ['z'],orient='v',ax=axes [5][1]);
r.set_title ('z',fontsize=15)

r = sns.distplot (df2 ['price'],ax=axes [6][0]);
r.set_title ('price',fontsize=15)
r = sns.boxplot (df2 ['price'],orient='v',ax=axes [6][1]);
r.set_title ('price',fontsize=15)
```

Out[60]:

```
Text(0.5, 1.0, 'price')
```


carat
carat
depth
depth
count
count
table
x
x
y
y
z
z
price
price
price

As we can see all the attributes have Outliers as we already checked in the describe function also none of is having the Normal Distribution except attribute (x), all others are highly Right skewed with long flat tail.

In [61]:

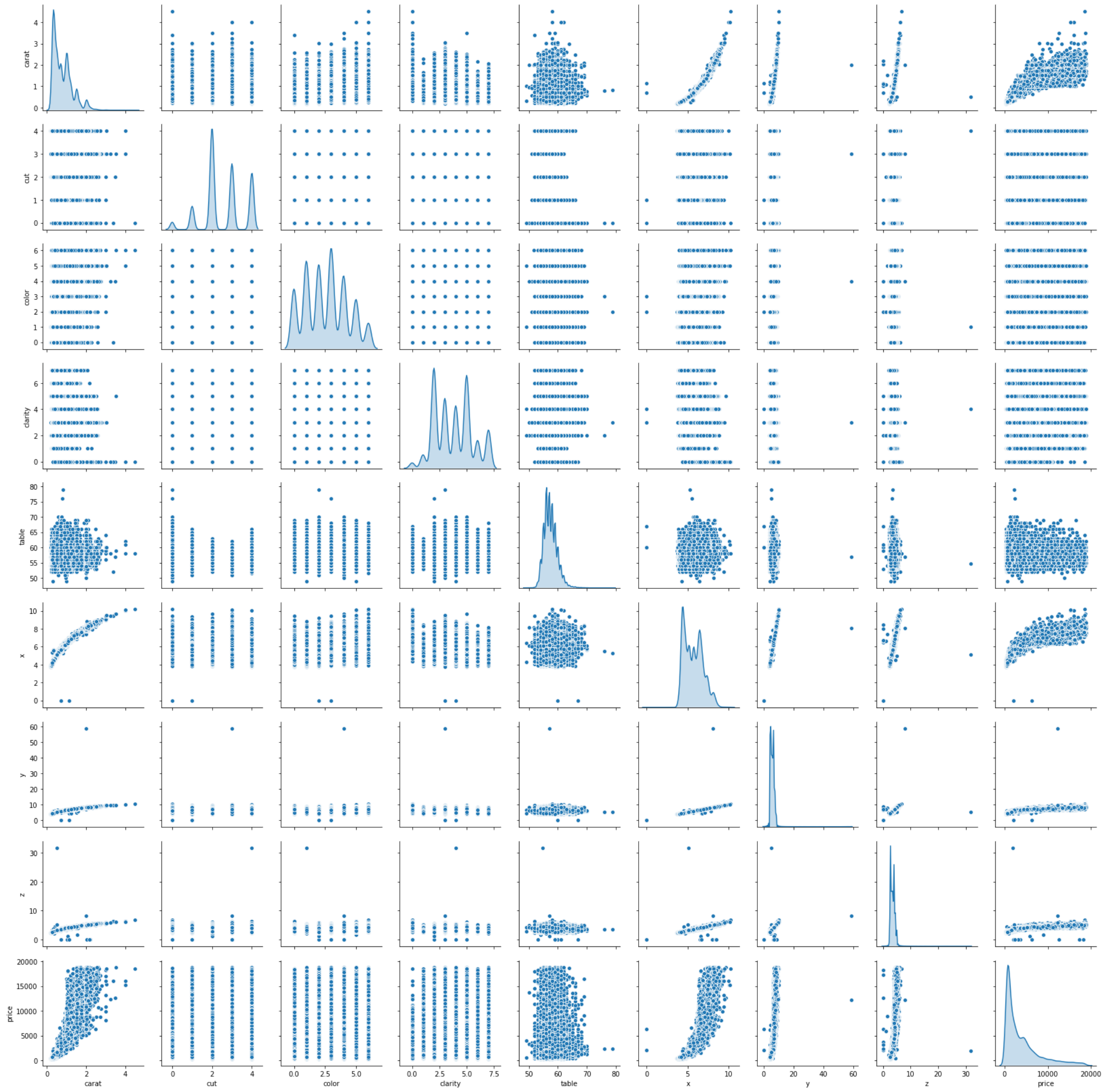
```
# Correlation:
df2.corr (method='pearson')
```

Out[61]:

	carat	cut	color	clarity	table	x	y	z
carat	1.000000	0.020146	0.293966	-0.211670	0.181685	0.976368	0.941071	0.940640
cut	0.020146	1.000000	-0.000679	0.017603	0.143989	0.024919	0.029999	0.006795
color	0.293966	-0.000679	1.000000	-0.023366	0.024418	0.274076	0.264290	0.267356
clarity	-0.211670	0.017603	-0.023366	1.000000	-0.079601	-0.224428	-0.213497	-0.219203
table	0.181685	0.143989	0.024418	-0.079601	1.000000	0.196206	0.182346	0.148944
x	0.976368	0.024919	0.274076	-0.224428	0.196206	1.000000	0.962715	0.956606
y	0.941071	0.029999	0.264290	-0.213497	0.182346	0.962715	1.000000	0.928923
z	0.940640	0.006795	0.267356	-0.219203	0.148944	0.956606	0.928923	1.000000
price	0.922416	0.039287	0.173213	-0.069447	0.126942	0.886247	0.856243	0.850536

In [62]:

```
# Pairplot:
sns.pairplot (df2, diag_kind='kde');
```



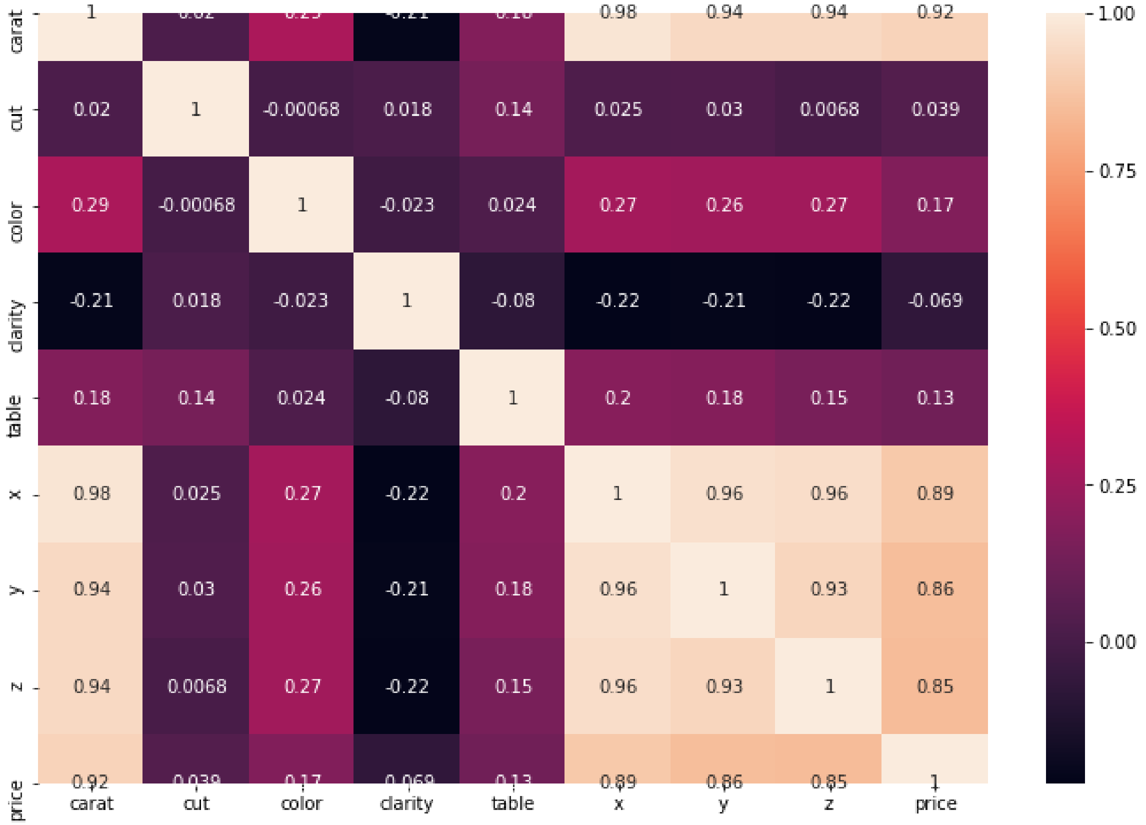
As we can see that (x, y and z) are highly correlated with the Carat and hence its Price is also High, also in the real-world price is always dependent on the size of the Precious minerals.

Cut and Color shows a slightly positive correlation.

Price is highly correlated with almost all the variables except (Clarity which have 6 %), also Clarity is totally depends upon the the origin of the minerals and the other factors play an important role for deciding the pricing, we will be able to check this all Correlation once we perform our Hypothesis

In [63]:

```
# HeatMap:

# Heatmap.
plt.figure(figsize=(12,8))
sns.heatmap (df2.corr(), annot=True),
plt.show()
```



In [ ]:

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?

Answer:

1 : We have already imputed the Null Values for the column [depth] since there were only 2.5 % of Null values for this single column, we could drop them, but we imputed it by its Mean value, Reason is when we checked the Mean and Maximum values for this specific column, we found that values were saturated around its mean and due to the close values when compared to its median we preferred Mean.

carat	0
depth	0
table	0
x	0
y	0
z	0

Also, after imputing null values we were able to plot the univariate analysis for the above question

There are 3 columns where values are equal to zero which are (x (length), y(width) and z(height)) as we know in real world this parameters are invalid as nothing can be counted in 0 mm, hence we don't have any useful information with it , but since this (0) values are only in 3 rows out of 26,967 so we don't need to change or drop them, it will not affect any calculations further in our model building.

Hence, we will not Change or Drop them.

Scaling is not Necessary in this case as we are building the model (Linear Regression) which is completely based on the equation and we need to calculate every attributes original value which we get via the BEST fit line, but if we are more concern about the co-efficient like (y = a1x1 + a2x2 + c) where a1 and a2 are co-efficient then we can perform scaling, but that will also not have any changes in the model's results.

Hence in this Case we will not perform Scaling.

In [ ]:

```

```

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 [112]:

```
#We have called up the Data Frame and store in a new data frame df8.
df8 = pd.read_csv (r'E:\Great Learning\Projects\Predictive Modelling\Data Sets\cubic_zi
rconia.csv')
df8.head()
```

Out[112]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

In [115]:

```
df8 = df8.drop ('Unnamed: 0',axis=1)
df8.head()
```

Out[115]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

In [116]:

```
df8.shape
```

Out[116]:

```
(26967, 10)
```

Encoding the Data which have String Values.

In [117]:

```
df8 = pd.get_dummies (df8, columns= ['cut','color','clarity'],drop_first=True)
df8.head()
```

Out[117]:

	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	cut_Premium	...	color_
0	0.30	62.1	58.0	4.27	4.29	2.66	499	0	1	0	...	
1	0.33	60.8	58.0	4.42	4.46	2.70	984	0	0	1	...	
2	0.90	62.2	60.0	6.04	6.12	3.78	6289	0	0	0	...	
3	0.42	61.6	56.0	4.82	4.80	2.96	1082	0	1	0	...	
4	0.31	60.4	59.0	4.35	4.43	2.65	779	0	1	0	...	

5 rows × 24 columns

In [118]:

```
df8.shape
```

Out[118]:

```
(26967, 24)
```

In [119]:

```
df8.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 24 columns):
carat            26967 non-null float64
depth            26270 non-null float64
table            26967 non-null float64
x                26967 non-null float64
y                26967 non-null float64
z                26967 non-null float64
price            26967 non-null int64
cut_Good         26967 non-null uint8
cut_Ideal        26967 non-null uint8
cut_Premium      26967 non-null uint8
cut_Very Good    26967 non-null uint8
color_E          26967 non-null uint8
color_F          26967 non-null uint8
color_G          26967 non-null uint8
color_H          26967 non-null uint8
color_I          26967 non-null uint8
color_J          26967 non-null uint8
clarity_IF       26967 non-null uint8
clarity_SI1      26967 non-null uint8
clarity_SI2      26967 non-null uint8
clarity_VS1      26967 non-null uint8
clarity_VS2      26967 non-null uint8
clarity_VVS1     26967 non-null uint8
clarity_VVS2     26967 non-null uint8
dtypes: float64(6), int64(1), uint8(17)
memory usage: 1.9 MB
```

In [120]:

```
df8.isnull().sum()
```

Out[120]:

```
carat              0
depth            697
table              0
x                  0
y                  0
z                  0
price              0
cut_Good           0
cut_Ideal          0
cut_Premium        0
cut_Very Good      0
color_E            0
color_F            0
color_G            0
color_H            0
color_I            0
color_J            0
clarity_IF         0
clarity_SI1        0
clarity_SI2        0
clarity_VS1        0
clarity_VS2        0
clarity_VVS1       0
clarity_VVS2       0
dtype: int64
```

In [121]:

```
df8.dtypes
```

Out[121]:

```
carat              float64
depth              float64
table              float64
x                  float64
y                  float64
z                  float64
price                int64
cut_Good             uint8
cut_Ideal            uint8
cut_Premium          uint8
cut_Very Good        uint8
color_E              uint8
color_F              uint8
color_G              uint8
color_H              uint8
color_I              uint8
color_J              uint8
clarity_IF           uint8
clarity_SI1          uint8
clarity_SI2          uint8
clarity_VS1          uint8
clarity_VS2          uint8
clarity_VVS1         uint8
clarity_VVS2         uint8
dtype: object
```

In [122]:

```
# Imputing Missing value.

for column in df8.columns:
    if df8 [column].dtype != 'object':
        mean = df8 [column].mean ()
        df8 [column] = df8 [column].fillna (mean)

df8.isnull().sum() # So we have treated the missing values.
```

Out[122]:

```
carat            0
depth            0
table            0
x                0
y                0
z                0
price            0
cut_Good         0
cut_Ideal        0
cut_Premium      0
cut_Very Good    0
color_E          0
color_F          0
color_G          0
color_H          0
color_I          0
color_J          0
clarity_IF       0
clarity_SI1      0
clarity_SI2      0
clarity_VS1      0
clarity_VS2      0
clarity_VVS1     0
clarity_VVS2     0
dtype: int64
```

In [123]:

```
df8.dtypes
```

Out[123]:

```
carat              float64
depth              float64
table              float64
x                  float64
y                  float64
z                  float64
price                int64
cut_Good             uint8
cut_Ideal            uint8
cut_Premium          uint8
cut_Very Good        uint8
color_E              uint8
color_F              uint8
color_G              uint8
color_H              uint8
color_I              uint8
color_J              uint8
clarity_IF           uint8
clarity_SI1          uint8
clarity_SI2          uint8
clarity_VS1          uint8
clarity_VS2          uint8
clarity_VVS1         uint8
clarity_VVS2         uint8
dtype: object
```

In [124]:

```
df8.describe ()
```

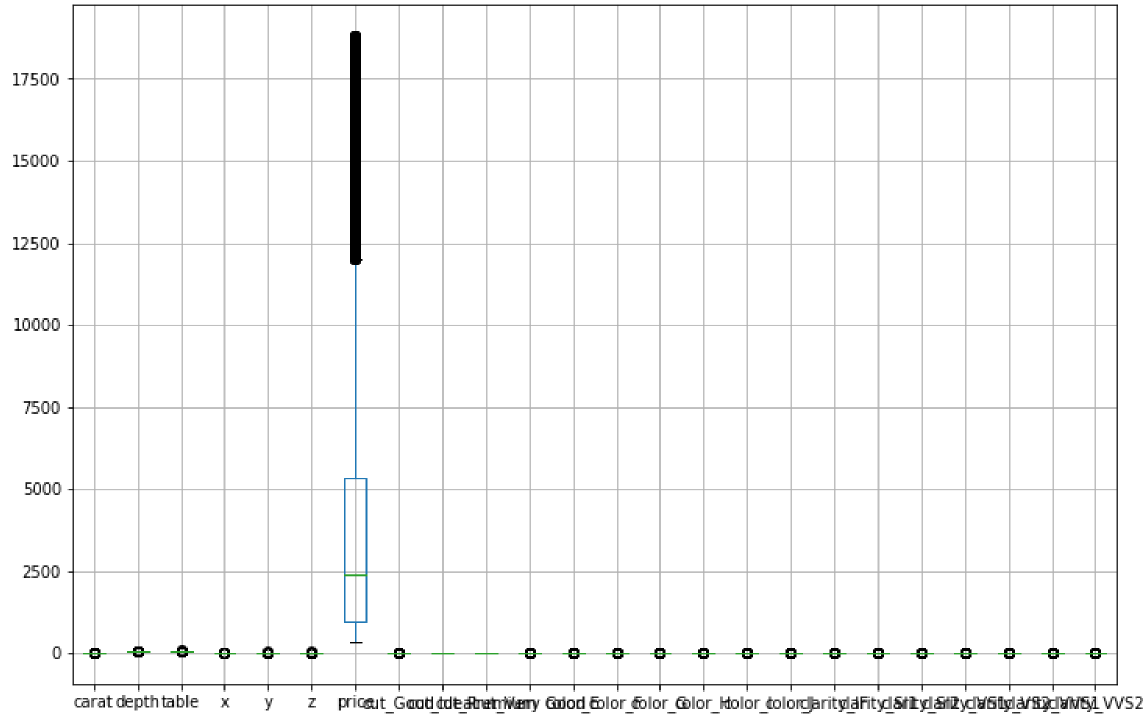
Out[124]:

	carat	depth	table	x	y	z
count	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000
mean	0.798375	61.745147	57.456080	5.729854	5.733569	3.538057
std	0.477745	1.394481	2.232068	1.128516	1.166058	0.720624
min	0.200000	50.800000	49.000000	0.000000	0.000000	0.000000
25%	0.400000	61.100000	56.000000	4.710000	4.710000	2.900000
50%	0.700000	61.800000	57.000000	5.690000	5.710000	3.520000
75%	1.050000	62.500000	59.000000	6.550000	6.540000	4.040000
max	4.500000	73.600000	79.000000	10.230000	58.900000	31.800000

8 rows × 24 columns

In [125]:

```
# Checking Outliers.
df8.boxplot (figsize=(12,8));
```



In [126]:

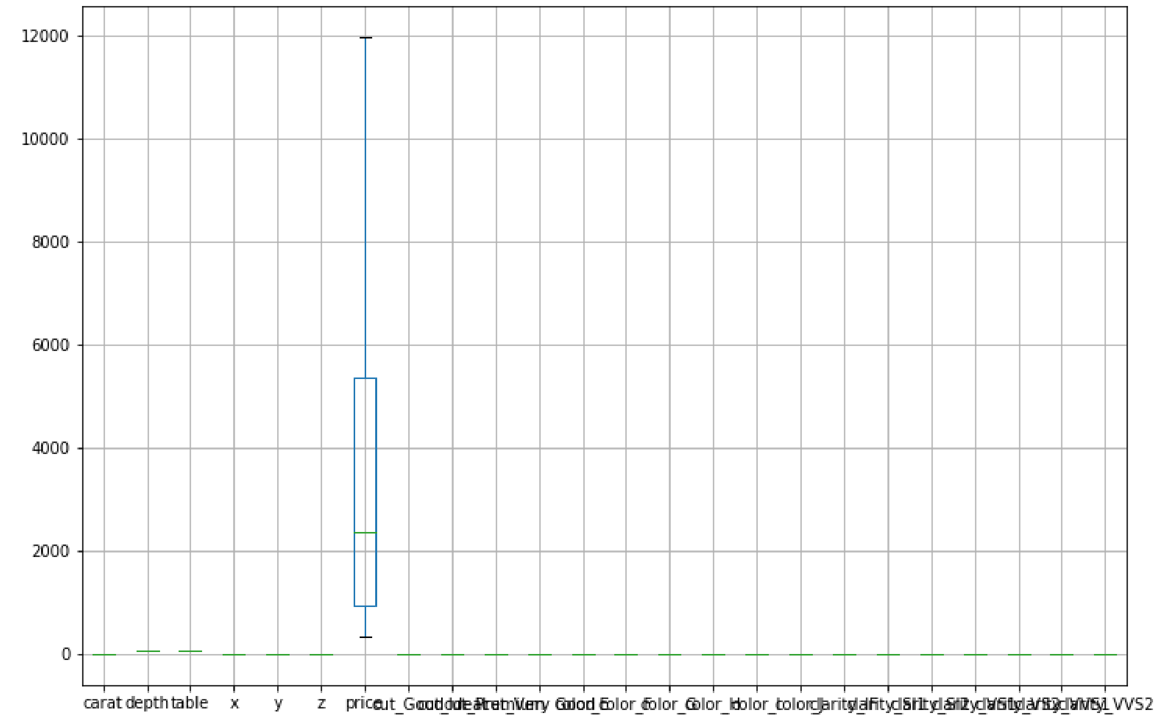
```
# We will treating the Outliers.

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 [129]:

```
for column in df8.columns:
    lr, ur = remove_outlier (df8 [column])
    df8 [column] = np.where (df8 [column] > ur, ur, df8[column])
    df8 [column] = np.where (df8 [column] < lr, lr, df8 [column])
```

In [130]:

```
df8.boxplot (figsize=(12,8)); # We have treated the Outliers.
```



In [131]:

```
dups = df8.duplicated ()
print ('Number of Duplicates in Data are %d' % (dups.sum()))
```

```
Number of Duplicates in Data are 57
```

In [132]:

```
# Removing Duplicates:
print ('Before',df8.shape)

df8.drop_duplicates(inplace=True)

print ('After',df8.shape)
```

```
Before (26967, 24)
After (26910, 24)
```

In [133]:

```
dups = df8.duplicated ()
print ('Number of Duplicates in Data are %d' % (dups.sum()))
```

```
Number of Duplicates in Data are 0
```

We have Treated :

Missing Values

Duplicates

Outliers

All Data Type is in Integer form.

Now we are ready to build our Linear Regression Model.

In [134]:

```
#  Data Split: Split the data into train and test (70:30). Apply Linear regression.

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [135]:

```
X = df8.drop ('price',axis=1)

y = df8.pop ('price')
```

In [136]:

```
X.shape
```

Out[136]:

```
(26910, 23)
```

In [137]:

```
y.shape
```

Out[137]:

```
(26910,)
```

In [138]:

```
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.30, random_sta
te = 8)
```

In [139]:

```
X_train.shape
```

Out[139]:

```
(18837, 23)
```

In [140]:

```
X_test.shape
```

Out[140]:

```
(8073, 23)
```

In [141]:

```
y_train.shape
```

Out[141]:

```
(18837,)
```

In [142]:

```
y_test.shape
```

Out[142]:

```
(8073,)
```

In [143]:

```
# Applying Linear Regression.

regression = LinearRegression ()
regression.fit (X_train, y_train)
```

Out[143]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

In [146]:

```
# Performance Metrics: Check the performance of Predictions on Train and Test sets usin
g Rsquare, RMSE.

# Rsquare (Maximum variance captured by our model,maximum socre means Good model)

regression.score (X_train, y_train)
```

Out[146]:

```
0.8829967286401554
```

In [147]:

```
regression.score (X_test, y_test)
```

Out[147]:

```
0.889262728951151
```

In [148]:

```
# Using RMSE.

from sklearn import metrics

#RMSE on Training data
predicted_train=regression.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[148]:

```
1185.7611450031725
```

In [150]:

```
#RMSE on Testing data
predicted_test = regression.fit (X_train, y_train).predict (X_test)
np.sqrt (metrics.mean_squared_error (y_test,predicted_test))
```

Out[150]:

```
1154.7048661736212
```

Training			Test	
Rsquare	0.88299		Rsquare	0.8892
RMSE	1185.76		RMSE	1154.7

Our model predicts good value for both the categories and we can go head with this model. But lets try to verify with Rsquare and RMSE by using stats model and also we will get the Inssights for the Business.

In [174]:

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

data_train.head()
data_train.columns
```

Out[174]:

```
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 [156]:

```
data_train.head()
```

Out[156]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...
19033	0.52	59.6	62.0	5.18	5.23	3.10	0.0	0.0	0.0	0.0	...
19694	0.32	61.6	57.0	4.40	4.43	2.72	0.0	1.0	0.0	0.0	...
4962	0.41	61.9	60.0	4.76	4.70	2.93	0.0	0.0	1.0	0.0	...
13309	0.25	61.7	56.0	4.06	4.07	2.51	0.0	1.0	0.0	0.0	...
5839	0.30	62.9	57.0	4.22	4.27	2.67	0.0	0.0	0.0	0.0	...

5 rows × 24 columns

In [161]:

```
import statsmodels.formula.api as smf
lm1 = smf.ols(formula= 'price ~ carat+depth+table+x+y+z+cut_Good+cut_Ideal+cut_Premium+
color_E+color_F+color_G+color_H+color_I+color_J+clarity_IF+clarity_SI1+clarity_SI2+clar
ity_VS1+clarity_VS2+clarity_VVS1+clarity_VVS2', data = data_train).fit()
lm1.params
```

Out[161]:

```
Intercept       8.367421e+03
carat           8.935702e+03
depth          -8.365336e+01
table          -4.229982e+01
x              -2.729444e+03
y               2.286626e+03
z              -4.947861e+02
cut_Good        1.389222e-12
cut_Ideal       2.927712e+02
cut_Premium     1.636165e+02
color_E         0.000000e+00
color_F         0.000000e+00
color_G         0.000000e+00
color_H         0.000000e+00
color_I         0.000000e+00
color_J         0.000000e+00
clarity_IF      0.000000e+00
clarity_SI1     0.000000e+00
clarity_SI2     0.000000e+00
clarity_VS1     0.000000e+00
clarity_VS2     0.000000e+00
clarity_VVS1    0.000000e+00
clarity_VVS2    0.000000e+00
dtype: float64
```

In [162]:

```
print(lm1.summary())  #Inferential statistics
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1755: RuntimeWarning: divide by zero encountered in double_scalars
  return np.sqrt(eigvals[0]/eigvals[-1])
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\model.py:1294:
RuntimeWarning: invalid value encountered in true_divide
  return self.params / self.bse
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructu
re.py:901: RuntimeWarning: invalid value encountered in greater
  return (a < x) & (x < b)
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructu
re.py:901: RuntimeWarning: invalid value encountered in less
  return (a < x) & (x < b)
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructu
re.py:1892: RuntimeWarning: invalid value encountered in less_equal
  cond2 = cond0 & (x <= _a)
```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                  price   R-squared:
0.883
Model:                            OLS   Adj. R-squared:
0.883
Method:                 Least Squares   F-statistic:                 1.776
e+04
Date:                Wed, 07 Apr 2021   Prob (F-statistic):
0.00
Time:                        17:32:35   Log-Likelihood:            -1.6006
e+05
No. Observations:               18837   AIC:                         3.201
e+05
Df Residuals:                   18828   BIC:                         3.202
e+05
Df Model:                           8
Covariance Type:            nonrobust
==========================================================================
======
                  coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
------
Intercept     8367.4208    914.109      9.154      0.000    6575.684    1.
02e+04
carat         8935.7015    105.294     84.865      0.000    8729.317    91
42.086
depth          -83.6534     12.293     -6.805      0.000    -107.749     -
59.557
table          -42.2998      5.252     -8.054      0.000     -52.594     -
32.005
x            -2729.4436    156.974    -17.388      0.000   -3037.126   -24
21.761
y             2286.6259    156.197     14.639      0.000    1980.466    25
92.786
z             -494.7861    138.397     -3.575      0.000    -766.057    -2
23.515
cut_Good      1.389e-12      1e-13     13.832      0.000    1.19e-12    1.
59e-12
cut_Ideal      292.7712     24.255     12.070      0.000     245.229     3
40.314
cut_Premium    163.6165     24.814      6.594      0.000     114.980     2
12.253
color_E               0          0        nan        nan           0
0
color_F               0          0        nan        nan           0
0
color_G               0          0        nan        nan           0
0
color_H               0          0        nan        nan           0
0
color_I               0          0        nan        nan           0
0
color_J               0          0        nan        nan           0
0
clarity_IF            0          0        nan        nan           0
0
clarity_SI1           0          0        nan        nan           0
0
```

```
clarity_SI2           0          0        nan        nan          0
0
clarity_VS1           0          0        nan        nan          0
0
clarity_VS2           0          0        nan        nan          0
0
clarity_VVS1          0          0        nan        nan          0
0
clarity_VVS2          0          0        nan        nan          0
0
==========================================================================
====
Omnibus:                     5066.228   Durbin-Watson:
1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           2825
8.745
Skew:                           1.180   Prob(JB):
0.00
Kurtosis:                       8.517   Cond. No.
inf
==========================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The smallest eigenvalue is      0. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

As we can see our P values are less than 0.05 hence we can say that there is no evidence to say that our Null Hypothesis is True.

Which means there is a strong relation between all those attributes for predicting the price.

And also our Data is proven that its from the recent population as there is no prove of Flux.

In [163]:

```
# Let us check the sum of squared errors by predicting value of y for test cases and
# subtracting from the actual y for the test cases

mse = np.mean((regression.predict(X_test)-y_test)**2)
```

In [164]:

```
import math

math.sqrt(mse)
```

Out[164]:

```
1154.7048661736192
```

In [172]:

```
#Root Mean Squared Error - RMSE
np.sqrt(mse)
```

Out[172]:

```
1154.7048661736192
```

In [175]:

```
# Prediction on Test data
y_pred = lm1.predict(data_test)
```

In [176]:

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

```
(8367.42) * Intercept + (8935.7) * carat + (-83.65) * depth + (-42.3) * ta
ble + (-2729.44) * x + (2286.63) * y + (-494.79) * z + (0.0) * cut_Good +
(292.77) * cut_Ideal + (163.62) * cut_Premium + (0.0) * color_E + (0.0) *
color_F + (0.0) * color_G + (0.0) * color_H + (0.0) * color_I + (0.0) * co
lor_J + (0.0) * clarity_IF + (0.0) * clarity_SI1 + (0.0) * clarity_SI2 +
(0.0) * clarity_VS1 + (0.0) * clarity_VS2 + (0.0) * clarity_VVS1 + (0.0) *
clarity_VVS2 +
```

In [177]:

```
# Checking on Train Data.
mse = np.mean((regression.predict(X_train)-y_train)**2)
```

In [178]:

```
import math

math.sqrt(mse)
```

Out[178]:

```
1185.7611450031725
```

In [179]:

```
regression.score (X_train,y_train)
```

Out[179]:

```
0.8829967286401554
```

In [180]:

```
regression.score (X_test,y_test)
```

Out[180]:

```
0.889262728951151
```

In [181]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = [variance_inflation_factor(X.values, ix) for ix in range(X.shape[1])]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1638: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - self.ssr/self.uncentered_tss
```

In [182]:

```
i=0
for column in X.columns:
    if i < 11:
        print (column ,"--->",  vif[i])
        i = i+1
```

```
carat ---> 110.79019942751951
depth ---> 1003.0877207182991
table ---> 865.4434452380447
x ---> 11223.925534891658
y ---> 10186.508885265544
z ---> 1907.5233844769832
cut_Good ---> nan
cut_Ideal ---> 2.705069384278806
cut_Premium ---> 2.0385346015058374
cut_Very Good ---> nan
color_E ---> nan
```

In [ ]:

```

```

As we can see in the below image the values for Training and Testing Data for predictions are almost same for Sklearn and by using Statsmodel, also there is not a huge difference between Training and Testing Data so there will be no overfit or underfit. By removing Attributes Color and Clarity (Based on our Hypothesis, or we can ask the Domain expert) and based on this we can put this model into the Production.

Sklearn				
Training			Test	
Rsquare	0.88		Rsquare	0.89
RMSE	1186		RMSE	1155

Statsmodel				
Training			Test	
Rsquare	0.88299		Rsquare	0.89
RMSE	1185.76		RMSE	1155

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

As per our model predictions, we know that our R^2 and RMSE values are same in both the scenarios, which means our model performed well on the given Data and there is no overfitting or underfitting.

After analysing all the details for every attribute, we came across that maximum columns are very much useful for predicting the Price of Cubic.

As we can see that (x, y and z) are highly correlated with the Carat and hence its Price is also High, also in the real-world price is always depend on the size of the Precious minerals.Cut and Color shows a slightly positive correlation.

Price is highly correlated with almost all the variables except (Clarity which have 6 %), also Clarity is totally dependent upon the origin of the minerals and also the other factors play an important role for deciding the pricing, we will be able to check this all Correlation once we perform our Hypothesis Testing.

We were able to find out the most important features which are and will always be valuable for predicting the price of Cubic, and, we have done the Hypothesis to check how much they are valid in this data set.

Important Features for Predicting "price" for the Cubic are:

carat
depth
table
x
y
z
cut_Good
cut_Ideal
cut_Premium

Also, the Variance Inflation Factor values are :

carat ---> 110.79019942751951
depth ---> 1003.0877207182991
table ---> 865.4434452380447
x ---> 11223.925534891658
y ---> 10186.508885265544
z ---> 1907.5233844769832
cut_Good ---> nan
cut_Ideal ---> 2.705069384278806
cut_Premium ---> 2.0385346015058374
cut_Very Good ---> nan
color_E ---> nan

As we can see that the values are pretty much higher for every important feature, highest value are the shape which include (x, y and z).

Recommendations:

By using the above insights, we can remove the unwanted/not important features and can only use those specific features which are mentioned as important and can try building the Model and check the scores. (decision is on Decision makers of the company).

Also, we can ask the Domain expert to understand what the impact is if we remove the Color and Clarity (I believe Clarity is much more specific towards the Market but in our scenario, we have received the NAN value which further i think because of encoding), hence we can ask the expert or can build the model without these features and check the scores.

Also, pricing is different in every country, we can ask the Decision makers if they want a specific price for specific country or state so that we can add that data in our model building and give them a specific predicting model.

We need to more focus on Depth, Carat, Size (x, y and z) and Table, by engaging with this and adding continuous data we will be able to provide more accuracy to our prediction.

5 Best Attributes.

carat
depth
table
x
y

In [ ]:

```

```

In [ ]:

```

```

Problem 2: Logistic Regression and LDA.

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.

Variable Name	Description
Holiday_Package	Opted for Holiday Package yes/no?
Salary	Employee salary
age	Age in years
edu	Years of formal education
no_young_children	The number of young children (younger than 7 years)
no_older_children	Number of older children
foreign	foreigner Yes/No

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.

In [185]:

```
# Loading the DataSet.

hdf = pd.read_csv (r'E:\Great Learning\Projects\Predictive Modelling\Data Sets\Holiday_
Package.csv')

# Reading the DataSet.

hdf.head()
```

Out[185]:

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	fo
0	1	no	48412	30	8	1	1	
1	2	yes	37207	45	8	0	1	
2	3	no	58022	46	9	0	0	
3	4	no	66503	31	11	2	0	
4	5	no	66734	44	12	0	2	

In [186]:

```
# Checking Shape of the Data.

hdf.shape # 872 Rows and 8 Columns.
```

Out[186]:

```
(872, 8)
```

In [188]:

```
# Checking the DataTypes.

hdf.dtypes # As Target Variable is in Object, hence we know that we will be using the C
lassification Model (Logistic Regression).
```

Out[188]:

```
Unnamed: 0            int64
Holliday_Package     object
Salary                int64
age                   int64
educ                  int64
no_young_children     int64
no_older_children     int64
foreign              object
dtype: object
```

In [189]:

```
hdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 872 entries, 0 to 871
Data columns (total 8 columns):
Unnamed: 0           872 non-null int64
Holliday_Package     872 non-null object
Salary               872 non-null int64
age                  872 non-null int64
educ                 872 non-null int64
no_young_children    872 non-null int64
no_older_children    872 non-null int64
foreign              872 non-null object
dtypes: int64(6), object(2)
memory usage: 54.6+ KB
```

In [192]:

```
hdf.describe(include='all')
```

Out[192]:

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_chil(
count	872.000000	872	872.000000	872.000000	872.000000	872.00(
unique	NaN	2	NaN	NaN	NaN	
top	NaN	no	NaN	NaN	NaN	
freq	NaN	471	NaN	NaN	NaN	
mean	436.500000	NaN	47729.172018	39.955275	9.307339	0.31'
std	251.869014	NaN	23418.668531	10.551675	3.036259	0.61:
min	1.000000	NaN	1322.000000	20.000000	1.000000	0.00(
25%	218.750000	NaN	35324.000000	32.000000	8.000000	0.00(
50%	436.500000	NaN	41903.500000	39.000000	9.000000	0.00(
75%	654.250000	NaN	53469.500000	48.000000	12.000000	0.00(
max	872.000000	NaN	236961.000000	62.000000	21.000000	3.00(

In [193]:

```
hdf.isnull().sum() # No Missing values.
```

Out[193]:

```
Unnamed: 0           0
Holliday_Package     0
Salary               0
age                  0
educ                 0
no_young_children    0
no_older_children    0
foreign              0
dtype: int64
```

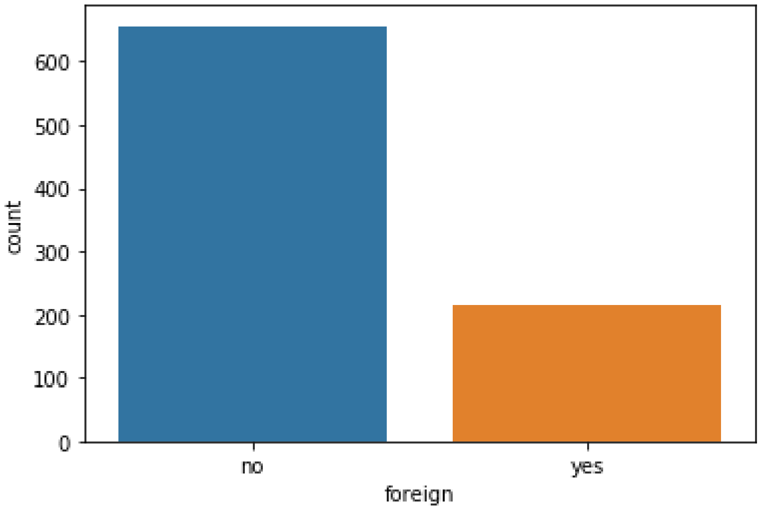
In [195]:

```
sns.countplot (hdf ['foreign']);
print ('/n')

hdf.foreign.value_counts() # We can Perform One hot encoding for this column so that da
ta type will convert in Integer.
```

```
/n
```

Out[195]:

```
no     656
yes    216
Name: foreign, dtype: int64
```



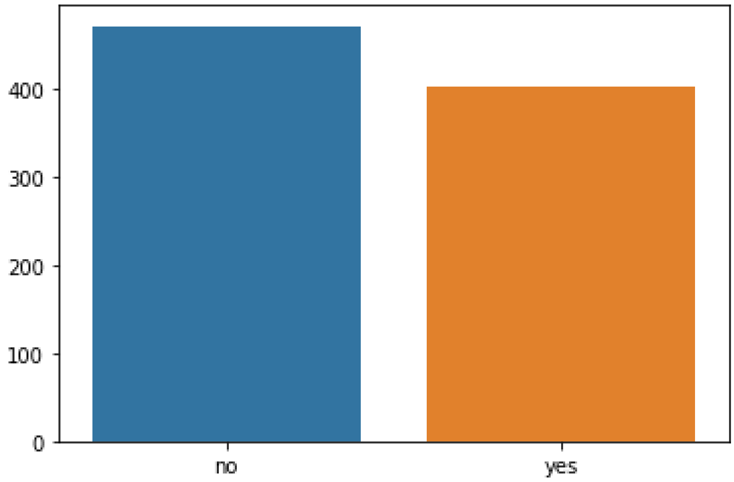
In [196]:

```
# Lets Check the Duplicates.

dups = hdf.duplicated ()
print ('Number of Duplicates in Data Set are %d' % (dups.sum()))
```

```
Number of Duplicates in Data Set are 0
```

In [199]:

```
# Lets check the Counts for Target Variable (Holliday_Package).

sns.barplot (hdf.Holliday_Package.value_counts().index, hdf.Holliday_Package.value_coun
ts().values);

plt.show()
print (hdf.Holliday_Package.value_counts(normalize=True))
```




```
no      0.540138
yes     0.459862
Name: Holliday_Package, dtype: float64
```

Inferences:

As we have seen that There are no Null Values (Missing Values) in our Data Set.

Also, there are no Duplicates.

After Describing the Data, we have found that the central tendency values are much more different for every column and when we compared it to its Standard Deviation, we see that there is a Huge difference between those.

Also, the Minimum and Maximum values have the Huge variations between every attribute which proves that there are Outliers, and for Classification (Logistic Regression) Outlier’s treatment is Mandatory which we will perform.

For Conversion of One OBJECT variable to the Integer which is (foreign), as we have checked that the response is in yes and no with values 216 and 656 respectively, which will add 2 columns.

TAREGT variable:

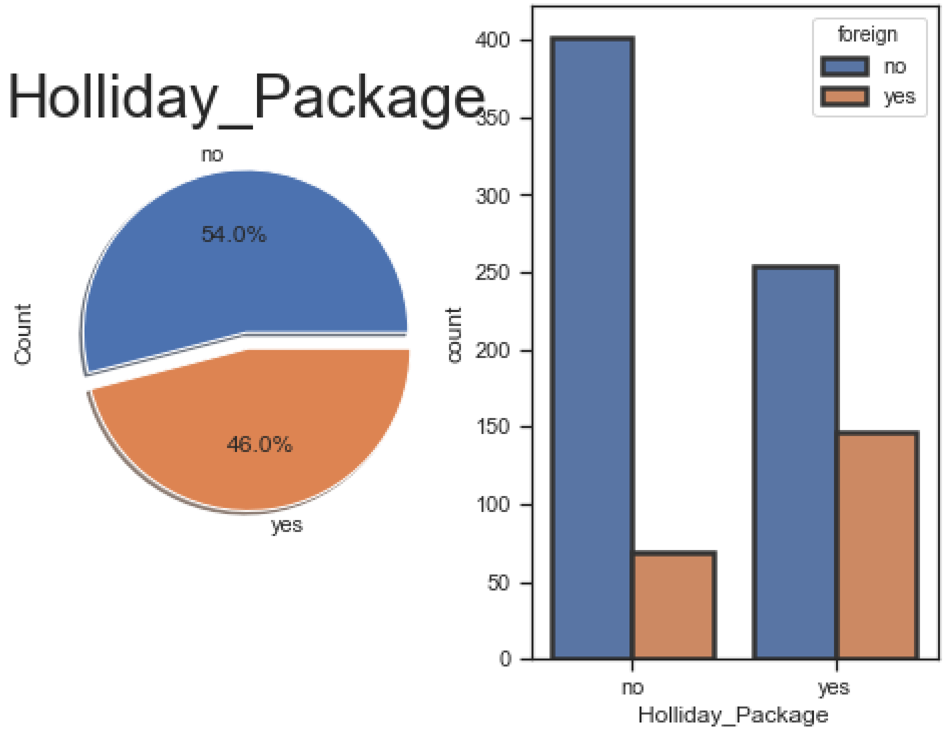
Holliday_Package: It is having yes and no output and as we know that the no is having 54 % and yes is having 45 %, which we can say that the Data is Balanced and can be able to give good Model for the Prediction.

Exploratory Data Analysis (EDA):

Univariate and Bi-Variate Analysis:

In [203]:

```
f,ax=plt.subplots(1,2,figsize=(8,6))
hdf['Holliday_Package'].value_counts().plot.pie(ax=ax[0],explode=[0,0.1],shadow=True,au
topct='%1.1f%%')
ax[0].set_title('Holliday_Package',fontsize=30)
ax[0].set_ylabel('Count')
sns.set(font="Verdana")
sns.set_style("ticks")
sns.countplot('Holliday_Package',hue='foreign',linewidth=2.5,edgecolor=".2",data=hdf,ax
=ax[1])
plt.ioff()
```

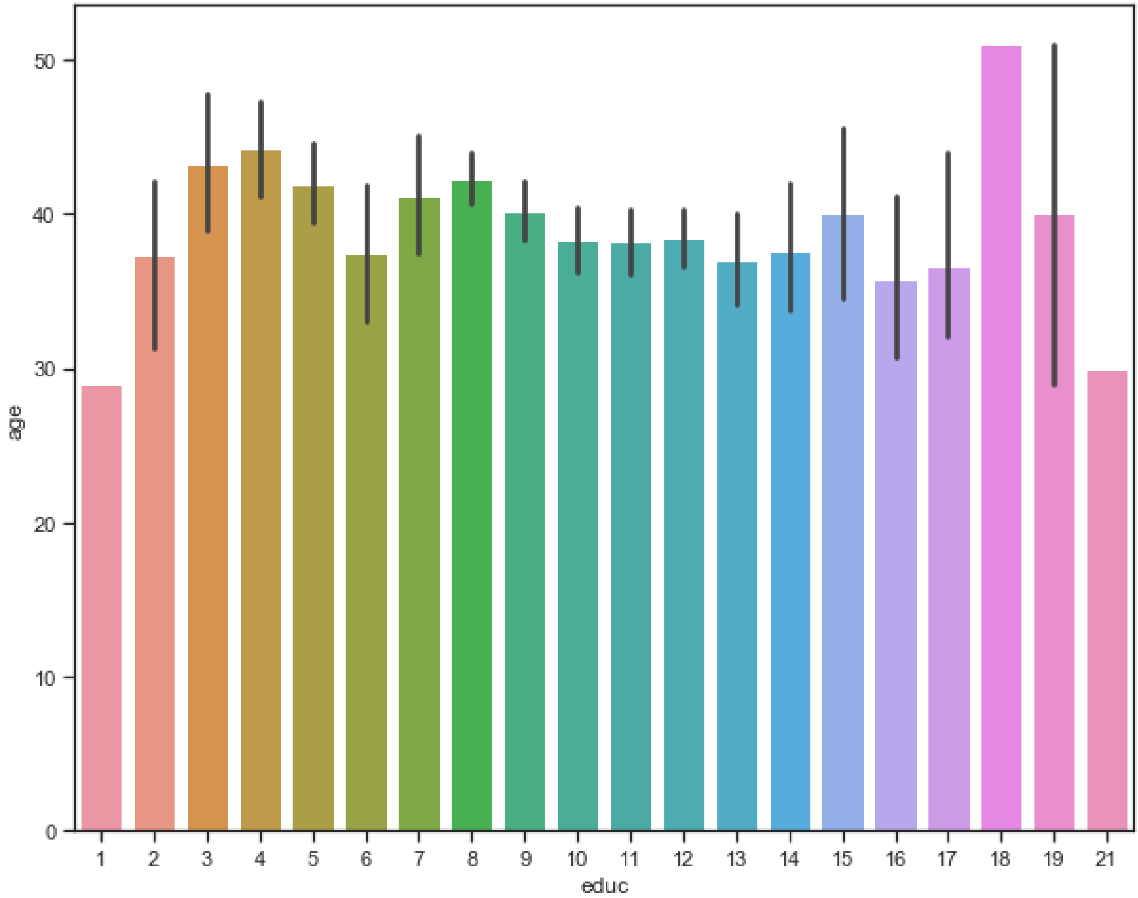


Above plot gives us an idea that if the Employee is a Foreigner then the chances of getting the Holliday_Package is More as compared to the not a Foreigner.

Important Note : Foreigner Employee has more impact on Holliday_Package.

In [208]:

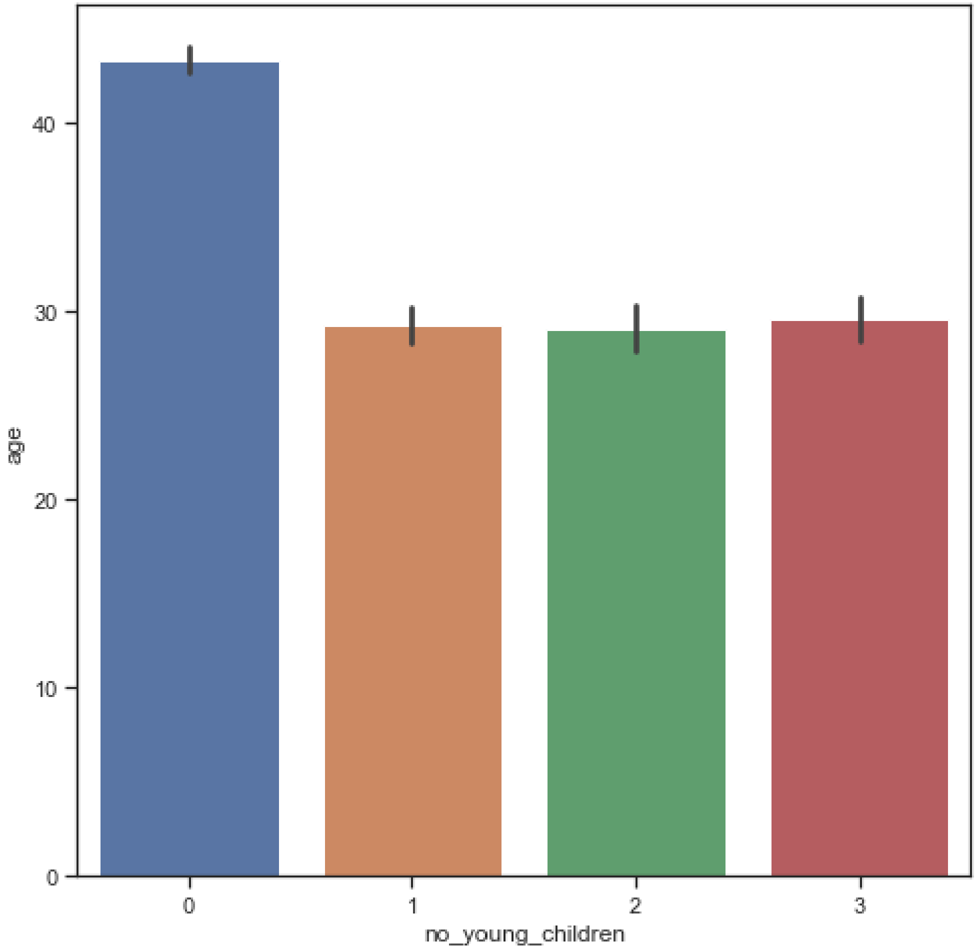
```
plt.figure (figsize=(10,8))
sns.barplot (x='educ', y='age', data=hdf);
```



Above plot shows that Years of formal education is maximum of 18 years (12+4+2) and minimum is 1 (which is possible for those employees who must be a security or in pantry) and 19 years of Education for those people who have the on an average of 40 years.

In [210]:

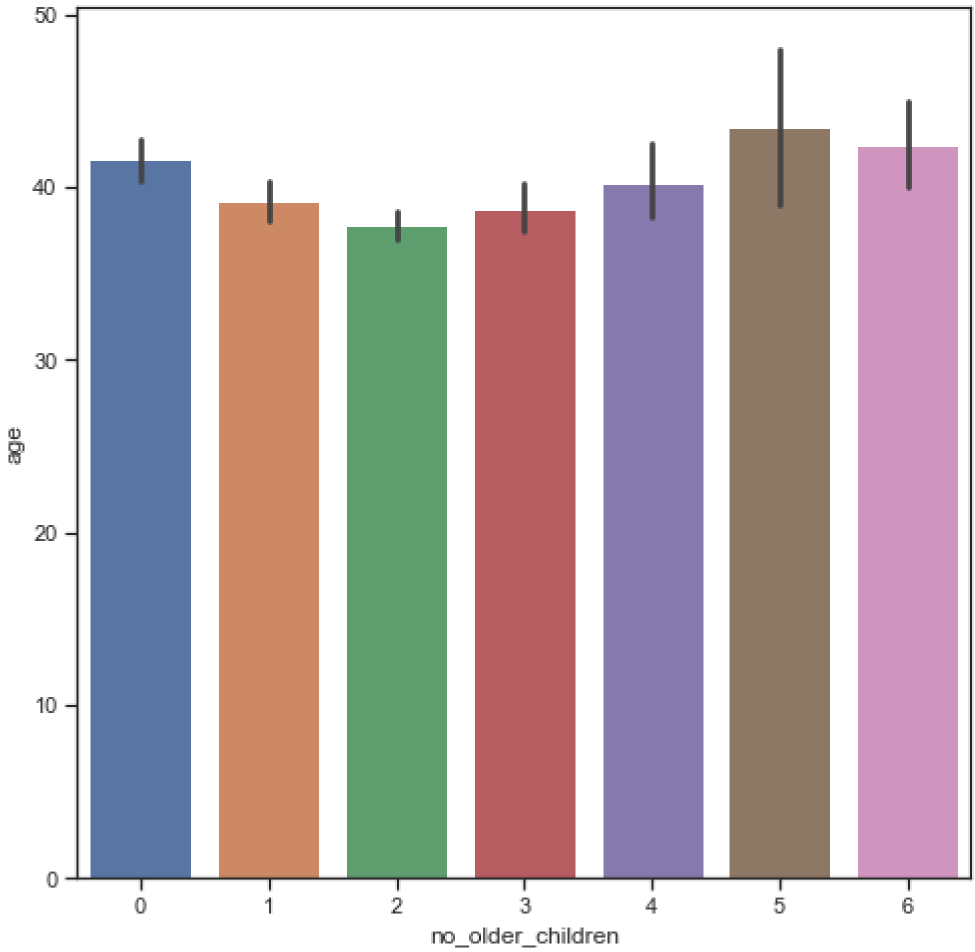
```
# lets check age and childrens.
plt.figure (figsize=(8,8))
sns.barplot (x='no_young_children',y='age', data=hdf),
plt.show()
```



Above plot shows that as Age increases no of young children (younger than 7 years) decreases and Below 30 Years almost everyone is having the Young children which proves that the Dependency on the employee is in early stage.

In [211]:

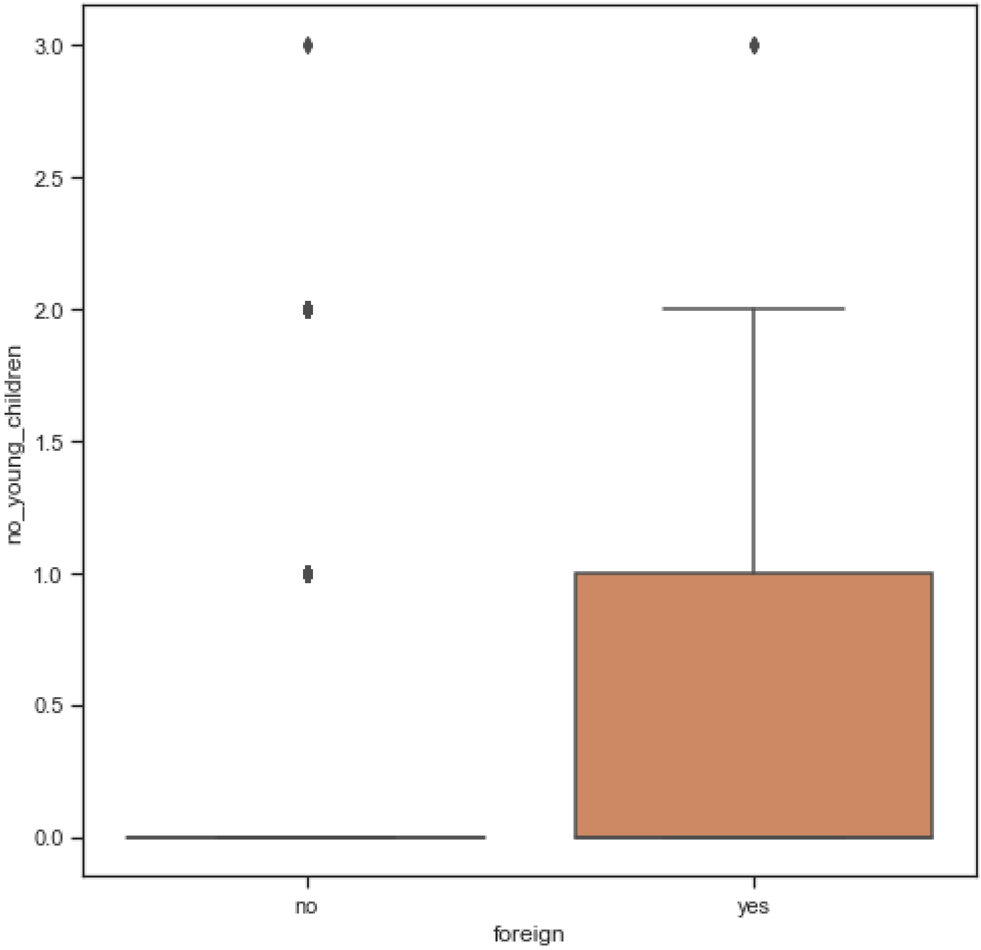
```
# For older children.

plt.figure (figsize=(8,8))
sns.barplot (x='no_older_children',y='age', data=hdf),
plt.show()
```



Here also as age increases no of older children's also increases.

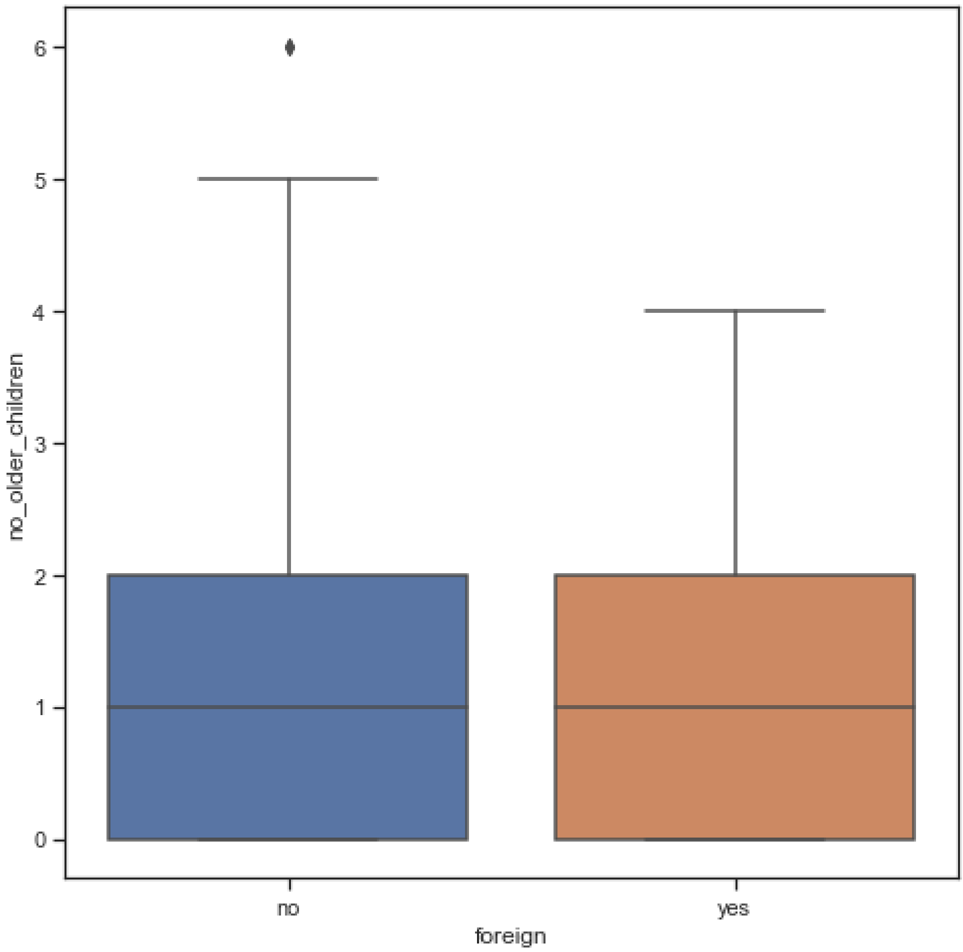
In [214]:

```
plt.figure (figsize=(8,8))
sns.boxplot (hdf ['foreign'], hdf ['no_young_children']);
plt.show ()
```



In [215]:

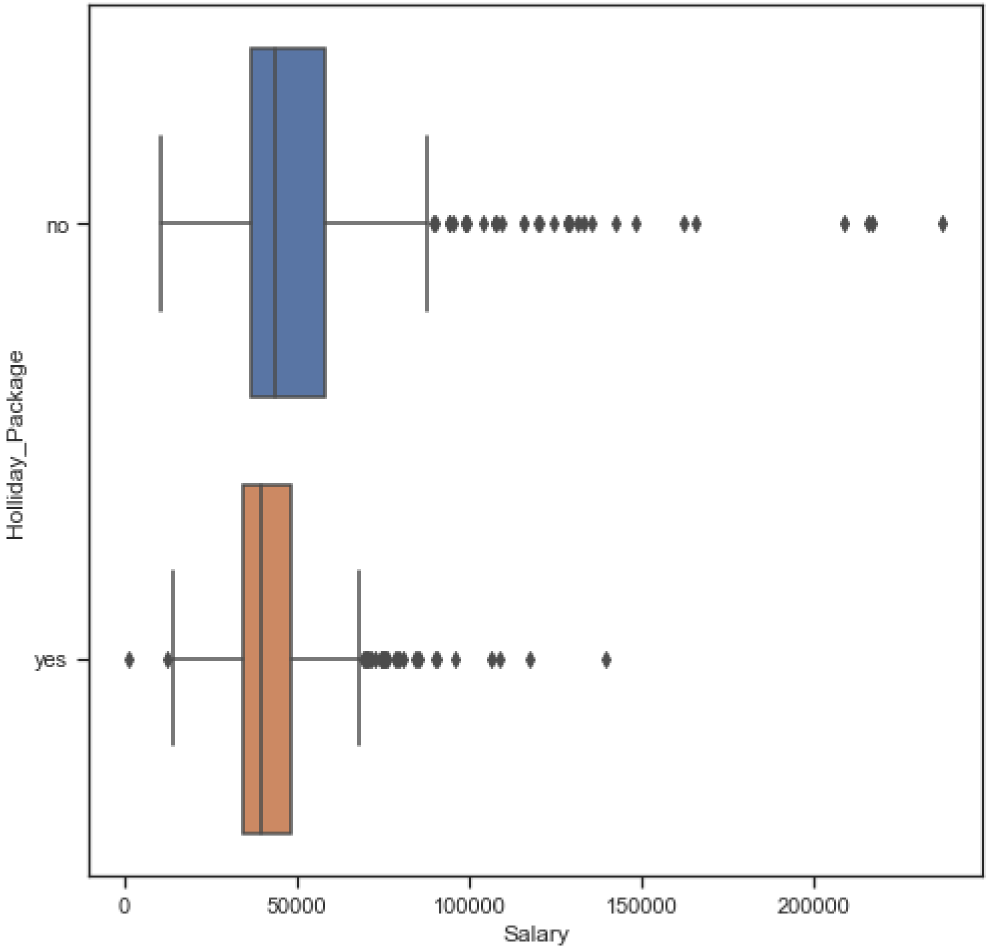
```
plt.figure (figsize=(8,8))
sns.boxplot (hdf ['foreign'], hdf ['no_older_children']);
plt.show ()
```



It shows that Foreigners have a smaller number of older children as compared to the Non-Foreigners.

In [216]:

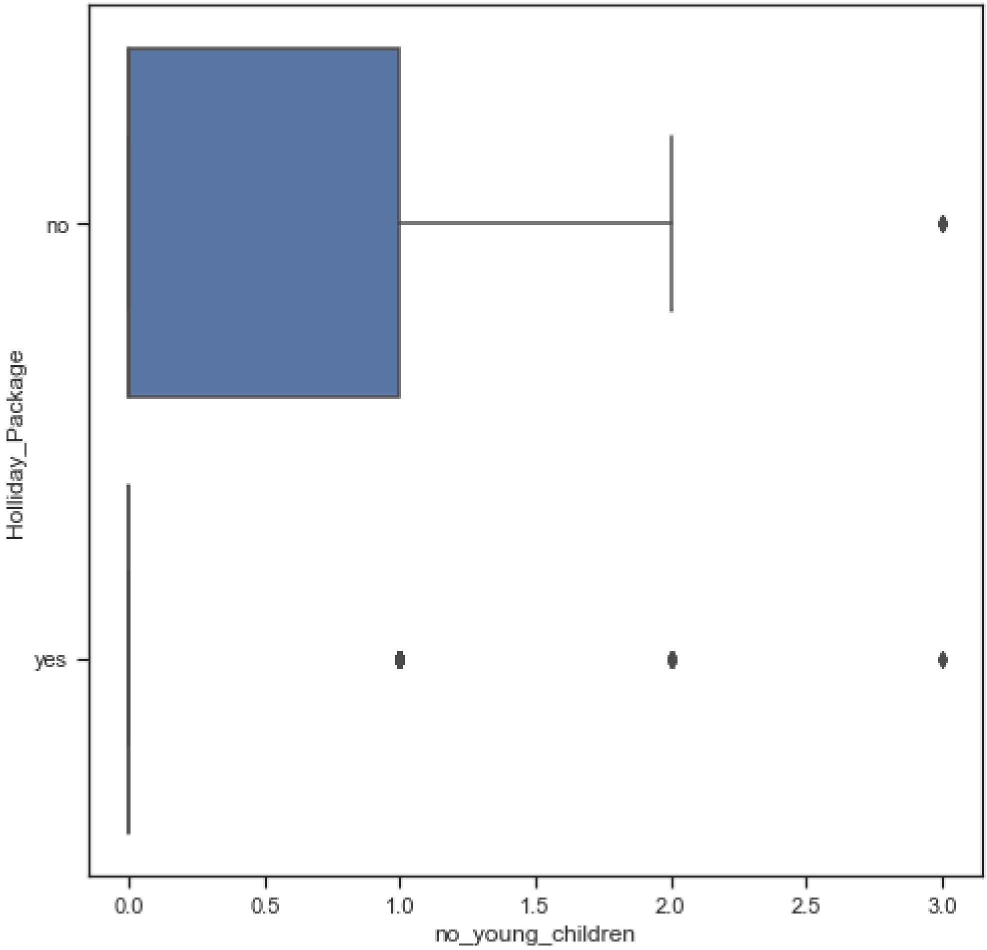
```
plt.figure (figsize=(8,8))
sns.boxplot (hdf ['Salary'], hdf ['Holliday_Package']);
plt.show ()
```



Employee opting for Holliday_Package seems to have Less Salary as Compared to the Employee having Much salary.

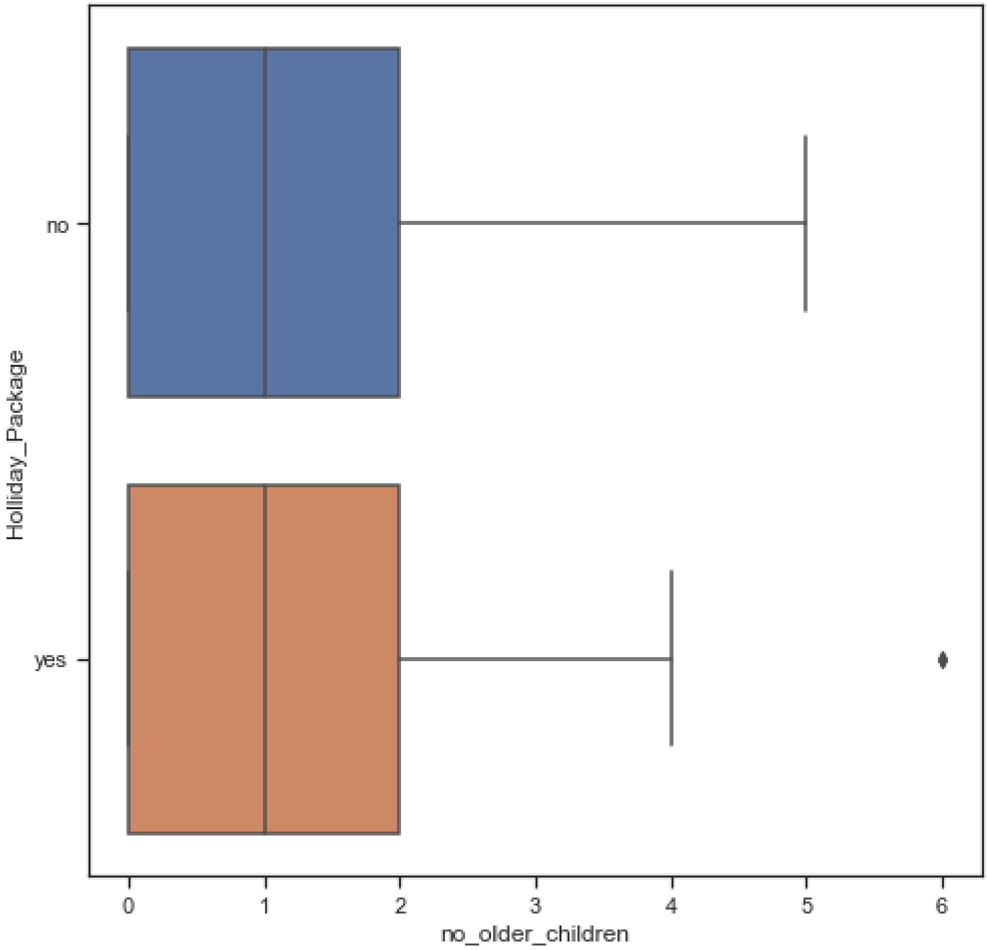
In [217]:

```
plt.figure (figsize=(8,8))
sns.boxplot (hdf ['no_young_children'], hdf ['Holliday_Package']);
plt.show ()
```



In [218]:

```
plt.figure (figsize=(8,8))
sns.boxplot (hdf ['no_older_children'], hdf ['Holliday_Package']);
plt.show ()
```



Employee having more than 2 children which are old (older than 7 years) are going for Holiday package also maximum is 6 children and median is 1 so this gives us an idea that Older children parents are opting the Holliday Package, but also same amount of ratio is for not obtaining the Holliday Package.

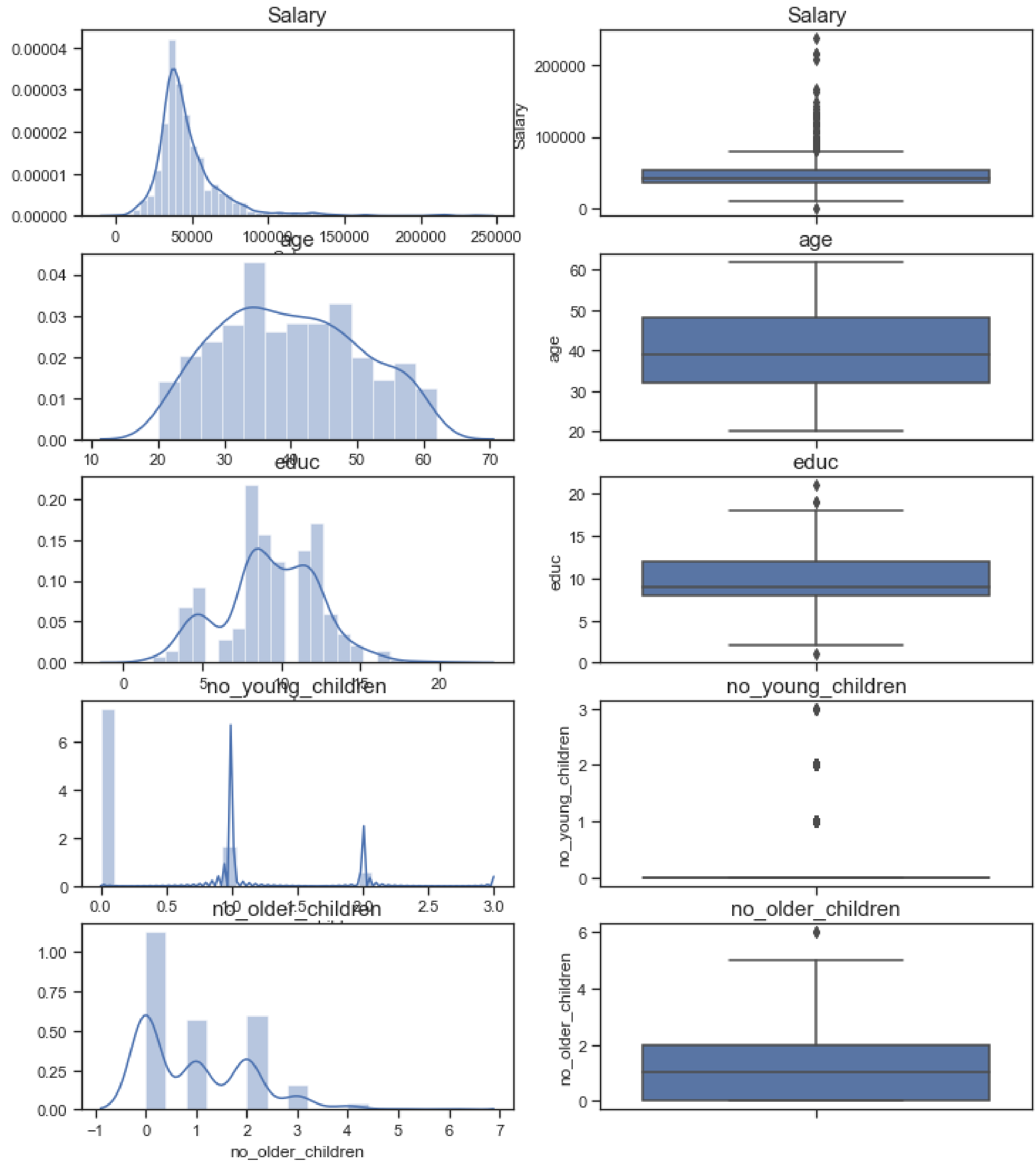
So, its 50-50 %, company can try for all those employees who are having no of older children with more focusing on children age group.

In [219]:

```
# BiVariate Analysis.

fig , axes = plt.subplots (nrows=5,ncols=2)
fig.set_size_inches (12,14)

r = sns.distplot (hdf ['Salary'],ax=axes [0][0]);
r.set_title ('Salary',fontsize=15)
r = sns.boxplot (hdf ['Salary'],orient='v',ax=axes [0][1]);
r.set_title ('Salary',fontsize=15)

r = sns.distplot (hdf ['age'],ax=axes [1][0]);
r.set_title ('age',fontsize=15)
r = sns.boxplot (hdf ['age'],orient='v',ax=axes [1][1]);
r.set_title ('age',fontsize=15)

r = sns.distplot (hdf ['educ'],ax=axes [2][0]);
r.set_title ('educ',fontsize=15)
r = sns.boxplot (hdf ['educ'],orient='v',ax=axes [2][1]);
r.set_title ('educ',fontsize=15)

r = sns.distplot (hdf ['no_young_children'],ax=axes [3][0]);
r.set_title ('no_young_children',fontsize=15)
r = sns.boxplot (hdf ['no_young_children'],orient='v',ax=axes [3][1]);
r.set_title ('no_young_children',fontsize=15)

r = sns.distplot (hdf ['no_older_children'],ax=axes [4][0]);
r.set_title ('no_older_children',fontsize=15)
r = sns.boxplot (hdf ['no_older_children'],orient='v',ax=axes [4][1]);
r.set_title ('no_older_children',fontsize=15)
```

Out[219]:

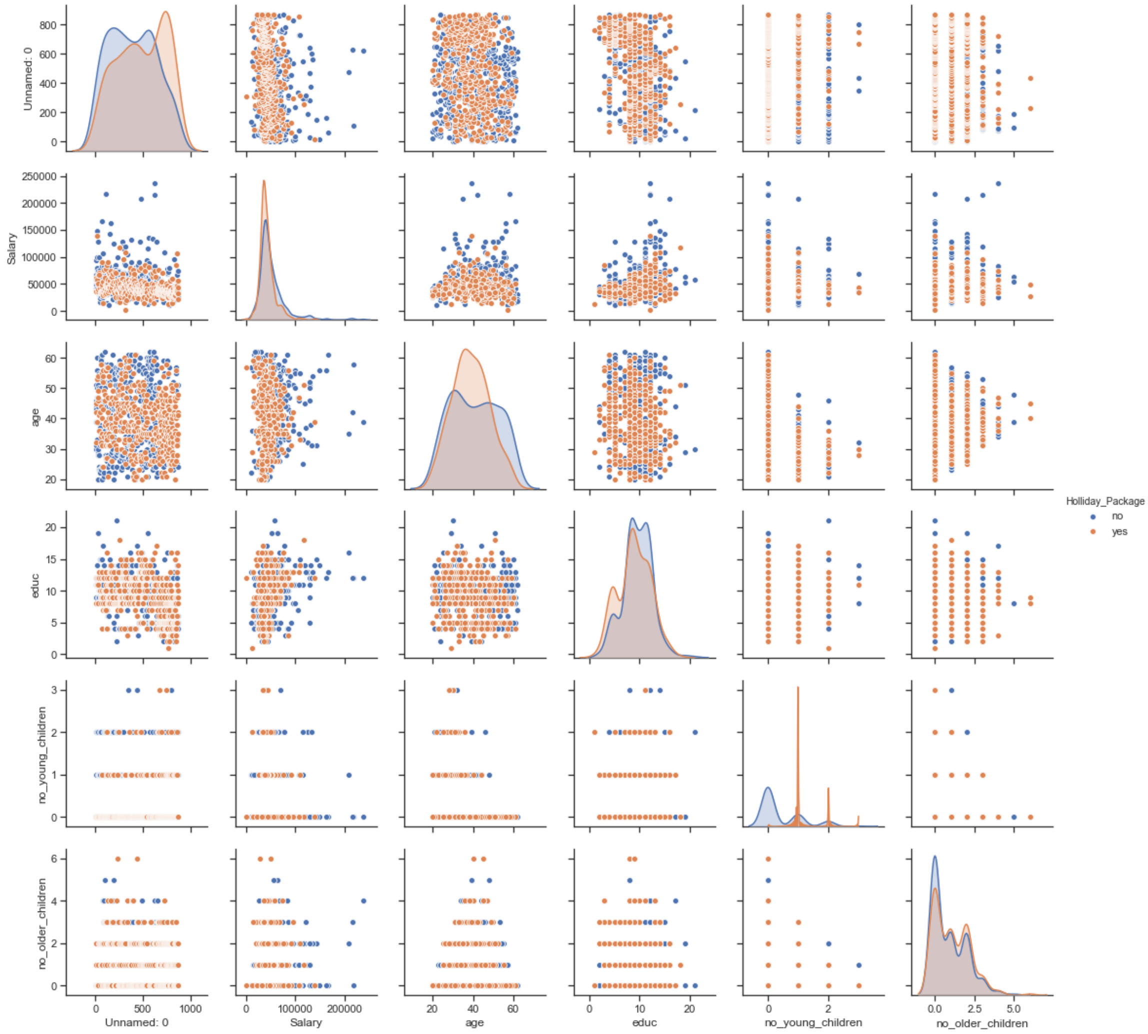
```
Text(0.5, 1.0, 'no_older_children')
```


Salary
Salary
age
age
educ
educ
no_young_children
no_young_children
no_older_children
no_older_children
Salary
age
educ
no_young_children
no_older_children
no_older_children

Salary and Education is Slightly Normally Distributed as per the Plot and remaining are not normally distributed.

In [221]:

```
# Pairplot : We will be more targetting on our Taget Variable which is Holliday_Packag
e.

sns.pairplot (hdf, diag_kind='kde', hue='Holliday_Package');
```



In [222]:

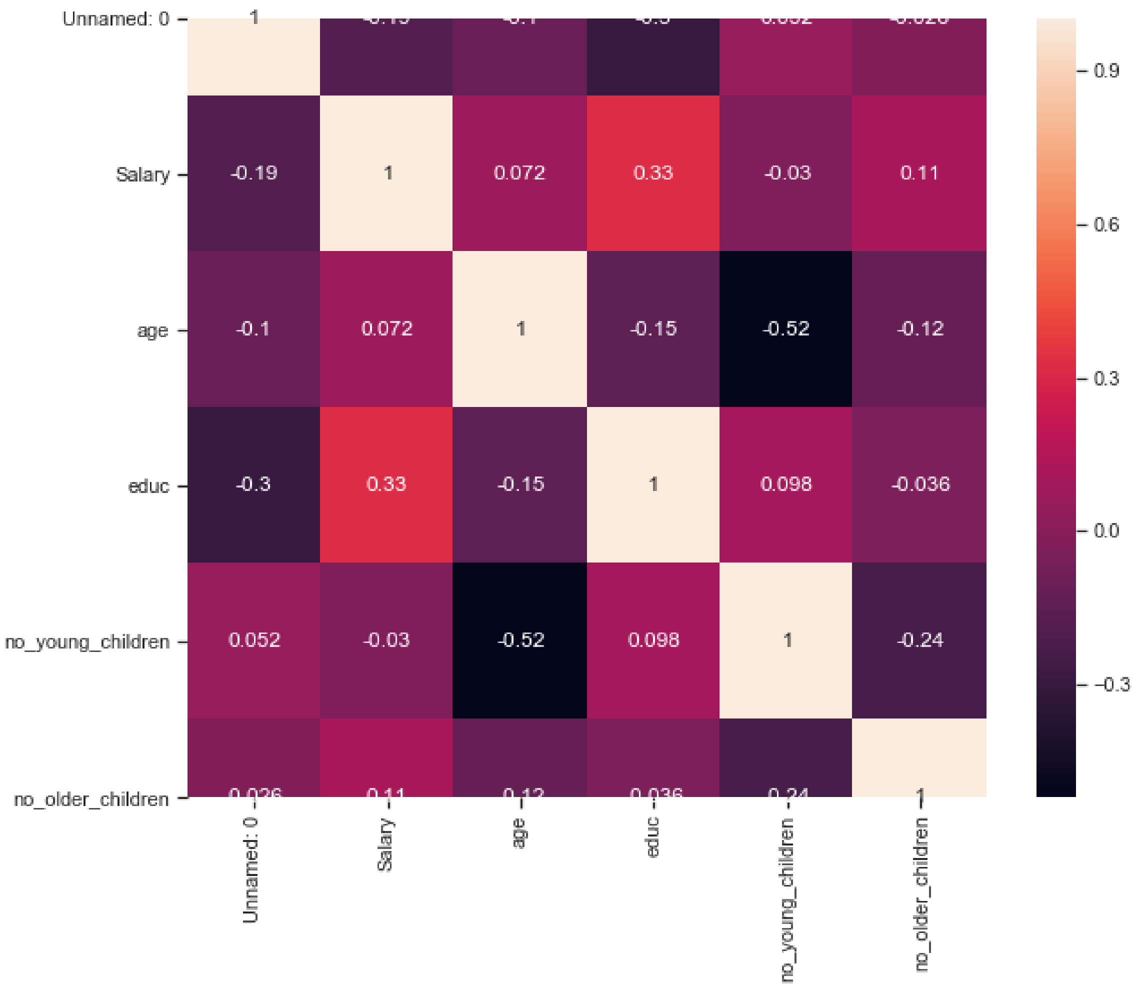
```
hdf.corr()
```

Out[222]:

	Unnamed: 0	Salary	age	educ	no_young_children	no_older_cl
Unnamed: 0	1.000000	-0.193249	-0.103782	-0.296015	0.052146	-0.(
Salary	-0.193249	1.000000	0.071709	0.326540	-0.029664	0.
age	-0.103782	0.071709	1.000000	-0.149294	-0.519093	-0.
educ	-0.296015	0.326540	-0.149294	1.000000	0.098350	-0.(
no_young_children	0.052146	-0.029664	-0.519093	0.098350	1.000000	-0.:
no_older_children	-0.025852	0.113772	-0.116205	-0.036321	-0.238428	1.(

In [223]:

```
# Heat Map.

plt.figure (figsize=(10,8))
sns.heatmap (hdf.corr(), annot=True);
plt.show()
```



Inferences:

As we can see in the pair plot, there are not many attributes/variables which are separating or classifying the Target Variable. Salary, Age and No of Older children are few attributes who are classifying or distinguishing the Target variable as we have seen this when we have done the Exploratory Data Analysis. Heat Map also showing that Education is have somehow correlation which is 0.33 %. By looking at this type of relation between attributes which are not classifying the Target variable, our model can be accurate or cannot be accurate or Good model, as we have only 2 classifiers out of 5. We will be building the model with this feature and will check the Score and Performance.

In [ ]:

```

```

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).

For Model building, we need to make sure all data types in integers and there should not be outliers.

In [246]:

```
# lets pull up the data one more time and will save it on new data frame name df_h.

df_h = pd.read_csv (r'E:\Great Learning\Projects\Predictive Modelling\Data Sets\Holiday_Package.csv')
df_h.head()
```

Out[246]:

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	fo
0	1	no	48412	30	8	1	1	
1	2	yes	37207	45	8	0	1	
2	3	no	58022	46	9	0	0	
3	4	no	66503	31	11	2	0	
4	5	no	66734	44	12	0	2	

In [247]:

```
# Removing Unwanted column Unnamed: 0

df_h = df_h.drop ('Unnamed: 0',axis=1)
df_h.head()
```

Out[247]:

	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
0	no	48412	30	8	1	1	no
1	yes	37207	45	8	0	1	no
2	no	58022	46	9	0	0	no
3	no	66503	31	11	2	0	no
4	no	66734	44	12	0	2	no

In [248]:

```
df_h.dtypes
```

Out[248]:

```
Holliday_Package     object
Salary                int64
age                   int64
educ                  int64
no_young_children     int64
no_older_children     int64
foreign              object
dtype: object
```

In [249]:

```
df_h.foreign.value_counts()
```

Out[249]:

```
no     656
yes    216
Name: foreign, dtype: int64
```

In [250]:

```
# Lets do Encoding by converting all the Object into an Integer type.

for feature in df_h.columns:
    if df_h [feature].dtype=='object':
        df_h [feature]=pd.Categorical (df_h[feature]).codes # coverting all dtypes in I
ntegers.
```

In [296]:

```
df_h.dtypes
```

Out[296]:

```
Salary                float64
age                   float64
educ                  float64
no_young_children     float64
no_older_children     float64
foreign               float64
dtype: object
```

In [252]:

```
df_h.head()
```

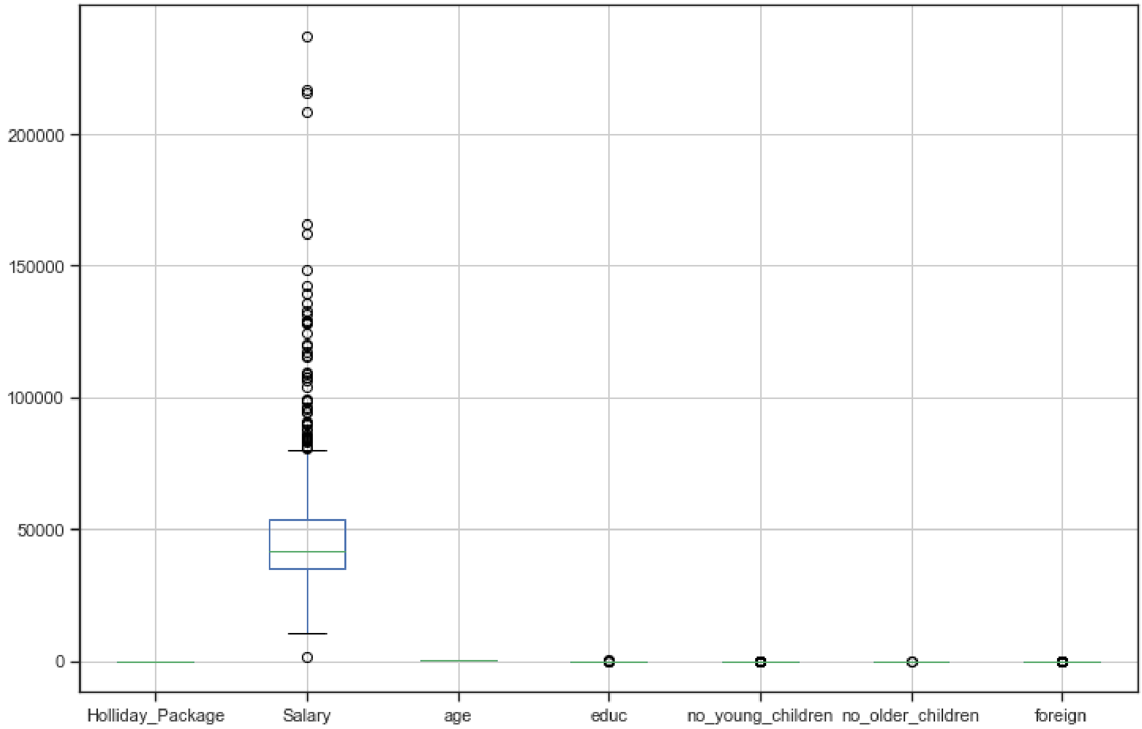
Out[252]:

	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
0	0	48412	30	8	1	1	0
1	1	37207	45	8	0	1	0
2	0	58022	46	9	0	0	0
3	0	66503	31	11	2	0	0
4	0	66734	44	12	0	2	0

Holliday_Package: 0 means NOT OPTED, 1 means OPTED, Foreign: 0 means Not Foreigner, 1 means Foreigner.

In [253]:

```
# Checking Outliers.

df_h.boxplot (figsize=(12,8));
```



In [254]:

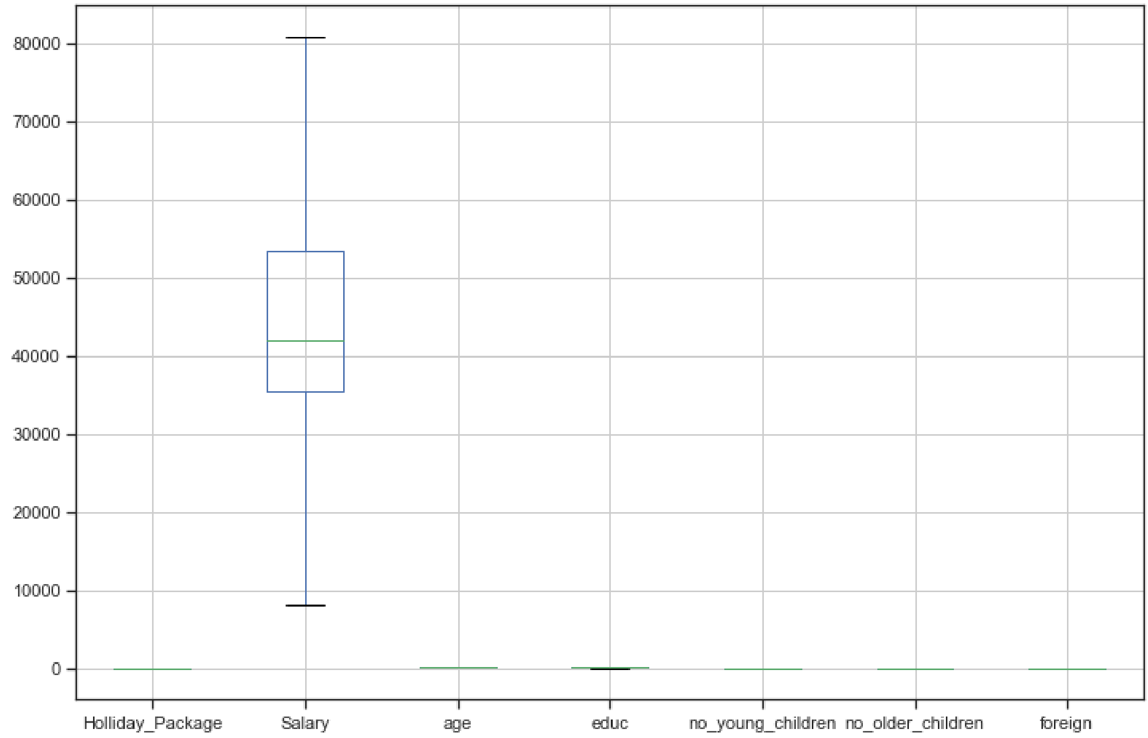
```
# Removing Outliers.

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 [255]:

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

In [256]:

```
df_h.boxplot (figsize=(12,8)); # Treated the Outliers.
```



Data Split: Split the data into train and test (70:30). Apply Logistic Regression and LDA (linear discriminant analysis).

In [258]:

```
X.shape
```

Out[258]:

(872, 6)

In [259]:

```
y.shape
```

Out[259]:

(872,)

In [261]:

```
# Splitting the Data into train and test.

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.30, random_state
=8)
```

In [262]:

```
X_train.shape
```

Out[262]:

```
(610, 6)
```

In [263]:

```
y_test.shape
```

Out[263]:

```
(262,)
```

In [264]:

```
# Applying Logistic Regression.

from sklearn.linear_model import LogisticRegression
```

In [265]:

```
log_regression = LogisticRegression ()
```

In [266]:

```
log_regression.fit (X_train, y_train) # This is our Best S curve.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. S
pecify a solver to silence this warning.
  FutureWarning)
```

Out[266]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [267]:

```
# Applying Linear Discriminate Analysis.

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [268]:

```
lda_model = LinearDiscriminantAnalysis()
```

In [269]:

```
lda_model.fit (X_train, y_train) # This is our Best Model.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.p
y:388: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
```

Out[269]:

```
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False, tol=0.000
1)
```

So we have treated the Outliers, Encoded the Data and Applied Logistic Regression and Linear Discriminate Analysis.

In [ ]:

```

```

In [ ]:

```

```

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.

In [270]:

```
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc
_curve,accuracy_score
```

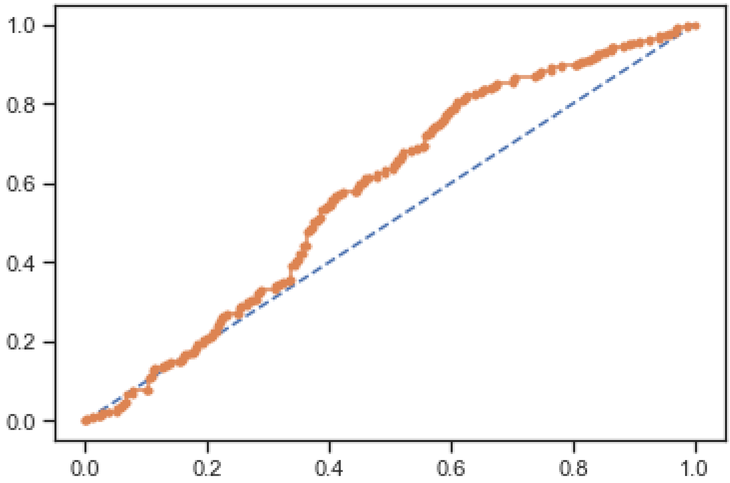
In [271]:

```
# For Logistic Regression.

y_train_pred = log_regression.predict (X_train)
y_test_pred = log_regression.predict (X_test)
```

In [273]:

```
# For Training Data.

models_names={log_regression:'Logistic Regression'}


print('Classification report for {} model is'.format(models_names[log_regression]),'\n'
,classification_report(y_train, y_train_pred))

print ('\n')

print('Accuracy for {} model is'.format(models_names[log_regression]),'\n',accuracy_sco
re(y_train, y_train_pred))

print('ROC AUC score for {} model is'.format(models_names[log_regression]),'\n',roc_auc
_score (y_train, y_train_pred))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[log_regression]),'\n',conf
usion_matrix(y_train,y_train_pred))


# predict probabilities
probs = log_regression.predict_proba (X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

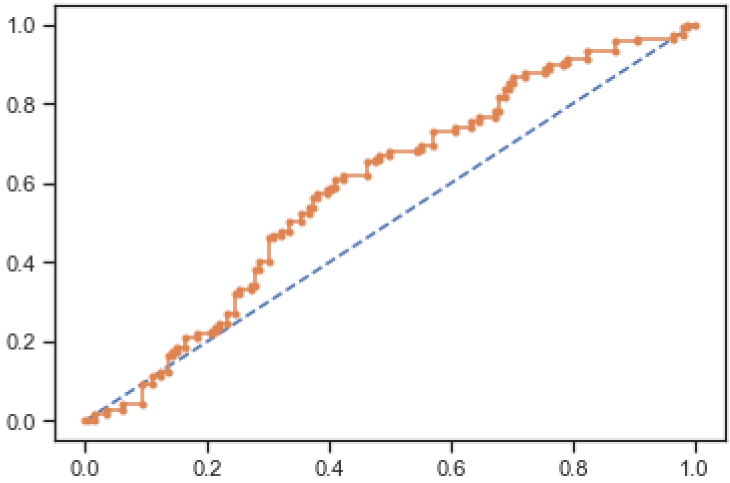
```
Classification report for Logistic Regression model is
               precision    recall  f1-score   support

         0.0       0.53      0.90      0.66       324
         1.0       0.43      0.09      0.15       286

    accuracy                           0.52       610
   macro avg       0.48      0.49      0.41       610
weighted avg       0.48      0.52      0.42       610



Accuracy for Logistic Regression model is
 0.5180327868852459
ROC AUC score for Logistic Regression model is
 0.49298540965207627


Confusion Matrix for Logistic Regression model is
 [[290  34]
 [260  26]]
AUC: 0.580
```



In [275]:

```
# For Testing Data.

models_names={log_regression:'Logistic Regression'}


print('Classification report for {} model is'.format(models_names[log_regression]),'\n'
,classification_report(y_test, y_test_pred))

print ('\n')

print('Accuracy for {} model is'.format(models_names[log_regression]),'\n',accuracy_sco
re(y_test, y_test_pred))

print('ROC AUC score for {} model is'.format(models_names[log_regression]),'\n',roc_auc
_score (y_test, y_test_pred))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[log_regression]),'\n',conf
usion_matrix(y_test, y_test_pred))


# predict probabilities
probs = log_regression.predict_proba (X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

```
Classification report for Logistic Regression model is
               precision    recall  f1-score   support

         0.0       0.56      0.89      0.69       147
         1.0       0.43      0.10      0.17       115

    accuracy                           0.55       262
   macro avg       0.49      0.50      0.43       262
weighted avg       0.50      0.55      0.46       262



Accuracy for Logistic Regression model is
 0.5458015267175572
ROC AUC score for Logistic Regression model is
 0.4977521443359953


Confusion Matrix for Logistic Regression model is
 [[131  16]
 [103  12]]
AUC: 0.591
```



As we can see that the model performs poor as Accuracy and Recall score (for 1) is very poor.

Lets try to do fine tuning to our model and check whether the model performance improves or not.

In [276]:

```
grid = {'penalty' : ['l2','none'],
       'solver' : ['sag','lbfgs'],
       'tol': [0.0001,0.00001]}
```

In [277]:

```
fine_model = LogisticRegression(max_iter=10000, n_jobs=3, random_state=8, tol=0.0001, s
olver='liblinear')
```

In [278]:

```
from sklearn.model_selection import GridSearchCV
```

In [279]:

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

In [280]:

```
grid.fit (X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
```

```
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Out[280]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=F
alse,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=No
ne,
                                          max_iter=10000, multi_class='war
n',
                                          n_jobs=3, penalty='l2',
                                          random_state=8, solver='liblinea
r',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=1,
             param_grid={'penalty': ['l2', 'none'], 'solver': ['sag', 'lbf
gs'],
                         'tol': [0.0001, 1e-05]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring='f1', verbose=0)
```

In [281]:

```
grid.best_params_
```

Out[281]:

```
{'penalty': 'l2', 'solver': 'lbfgs', 'tol': 0.0001}
```

In [282]:

```
grid.best_estimator_ # After fine tuning this is our Best Sigmoid curve (S curve).
```

Out[282]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=3, penalty='l2', random_stat
e=8,
                   solver='lbfgs', tol=0.0001, verbose=0, warm_start=Fals
e)
```

In [283]:

```
best_model = grid.best_estimator_
```

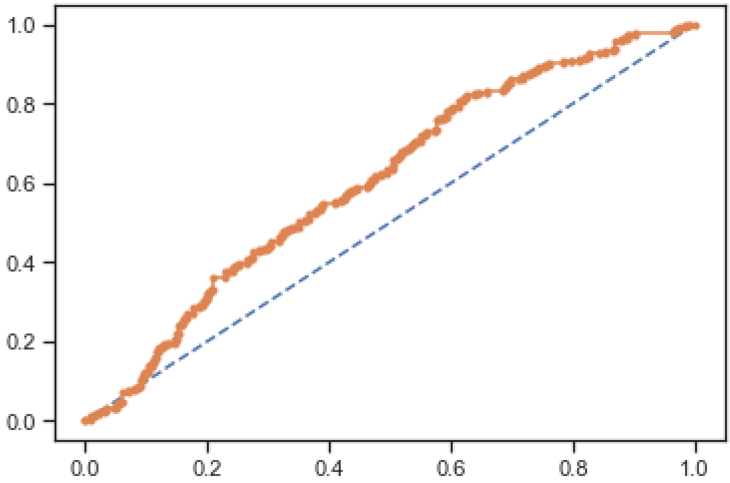
In [284]:

```
# Prediction on Training set.

y_train_predict = best_model.predict (X_train)
y_test_predict = best_model.predict (X_test)
```

In [285]:

```
# Lets try to calculate scores.

# For Training Data.

models_names={best_model:'Logistic Regression'}


print('Classification report for {} model is'.format(models_names[best_model]),'\n',cla
ssification_report(y_train, y_train_pred))

print ('\n')

print('Accuracy for {} model is'.format(models_names[best_model]),'\n',accuracy_score(y
_train, y_train_pred))

print('ROC AUC score for {} model is'.format(models_names[best_model]),'\n',roc_auc_sco
re (y_train, y_train_pred))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[best_model]),'\n',confusio
n_matrix(y_train,y_train_pred))


# predict probabilities
probs = best_model.predict_proba (X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

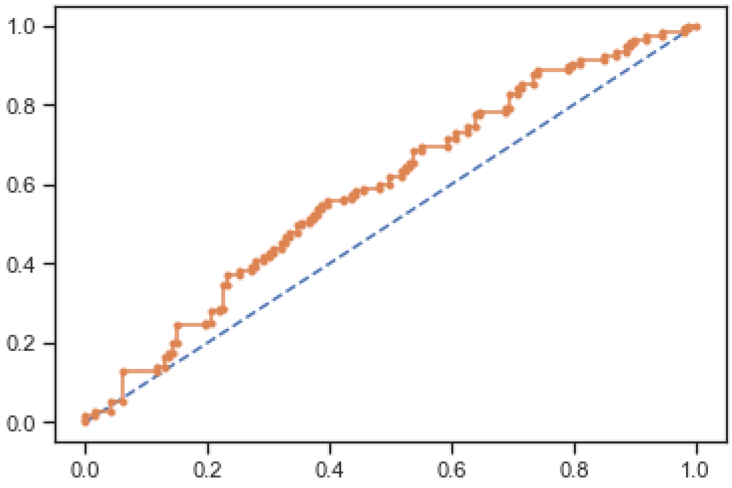
```
Classification report for Logistic Regression model is
               precision    recall  f1-score   support

         0.0       0.53      0.90      0.66       324
         1.0       0.43      0.09      0.15       286

    accuracy                           0.52       610
   macro avg       0.48      0.49      0.41       610
weighted avg       0.48      0.52      0.42       610


Accuracy for Logistic Regression model is
 0.5180327868852459
ROC AUC score for Logistic Regression model is
 0.49298540965207627


Confusion Matrix for Logistic Regression model is
 [[290  34]
 [260  26]]
AUC: 0.609
```



In [287]:

```
# For Testing Data.

models_names={best_model:'Logistic Regression'}


print('Classification report for {} model is'.format(models_names[best_model]),'\n',cla
ssification_report(y_test, y_test_predict))

print ('\n')

print('Accuracy for {} model is'.format(models_names[best_model]),'\n',accuracy_score(y
_test, y_test_predict))

print('ROC AUC score for {} model is'.format(models_names[best_model]),'\n',roc_auc_sco
re (y_test, y_test_predict))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[best_model]),'\n',confusio
n_matrix(y_test, y_test_predict))


# predict probabilities
probs = best_model.predict_proba (X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

Classification report for Logistic Regression model is
               precision    recall  f1-score   support

         0.0       0.56      1.00      0.72       147
         1.0       0.00      0.00      0.00       115

    accuracy                           0.56       262
   macro avg       0.28      0.50      0.36       262
weighted avg       0.31      0.56      0.40       262



Accuracy for Logistic Regression model is
 0.5610687022900763
ROC AUC score for Logistic Regression model is
 0.5


Confusion Matrix for Logistic Regression model is
 [[147   0]
 [115   0]]
AUC: 0.591
```



In [290]:

```
# lets check the probabilities:

ytest_predict_prob_logreg = best_model.predict_proba(X_test)
ytest_predict_prob_logreg.mean()
```

Out[290]:

```
0.5
```

In [291]:

```
pd.DataFrame (ytest_predict_prob_logreg).head()
```

Out[291]:

	0	1
0	0.600151	0.399849
1	0.598449	0.401551
2	0.584966	0.415034
3	0.562176	0.437824
4	0.535245	0.464755

In [ ]:

```

```

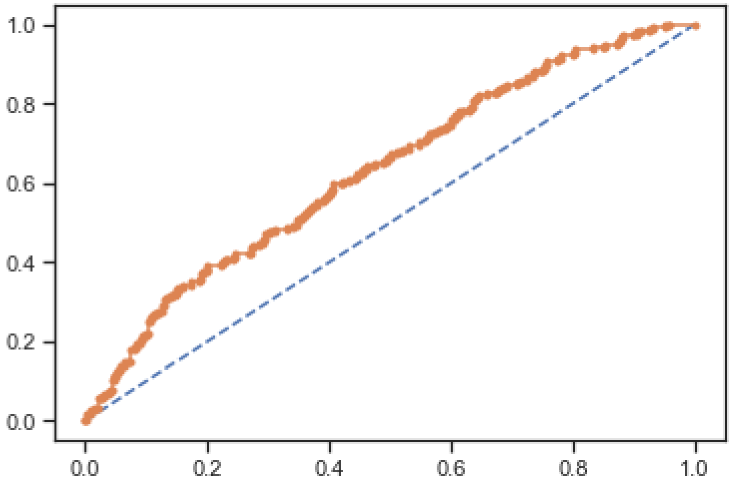
In [292]:

```
# For Linear Discriminate Analysis (LDA)

y_train_pred_lda = lda_model.predict (X_train)
y_test_pred_lda = lda_model.predict (X_test)
```

In [294]:

```
# For Training Data.

models_names={lda_model:'Linear Discriminate Analysis'}


print('Classification report for {} model is'.format(models_names[lda_model]),'\n',clas
sification_report(y_train, y_train_pred_lda))

print ('\n')

print('Accuracy for {} model is'.format(models_names[lda_model]),'\n',accuracy_score(y_
train, y_train_pred_lda))

print('ROC AUC score for {} model is'.format(models_names[lda_model]),'\n',roc_auc_scor
e (y_train, y_train_pred_lda))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[lda_model]),'\n',confusion
_matrix(y_train,y_train_pred_lda))


# predict probabilities
probs = lda_model.predict_proba (X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

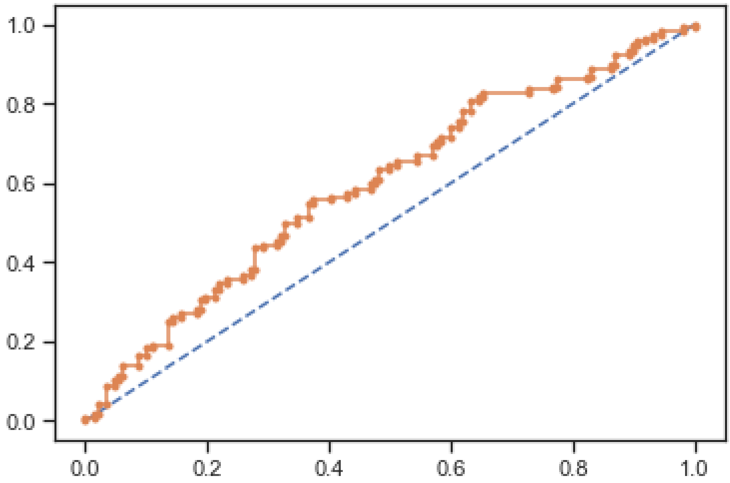
```
Classification report for Linear Discriminate Analysis model is
               precision    recall  f1-score   support

         0.0       0.61      0.63      0.62       324
         1.0       0.56      0.53      0.55       286

    accuracy                           0.59       610
   macro avg       0.58      0.58      0.58       610
weighted avg       0.58      0.59      0.58       610



Accuracy for Linear Discriminate Analysis model is
 0.5852459016393443
ROC AUC score for Linear Discriminate Analysis model is
 0.5822973322973324


Confusion Matrix for Linear Discriminate Analysis model is
 [[204 120]
 [133 153]]
AUC: 0.633
```



In [295]:

```
# For Testing Data.

models_names={lda_model:'Linear Discriminate Analysis'}


print('Classification report for {} model is'.format(models_names[lda_model]),'\n',clas
sification_report(y_test, y_test_pred_lda))

print ('\n')

print('Accuracy for {} model is'.format(models_names[lda_model]),'\n',accuracy_score(y_
test, y_test_pred_lda))

print('ROC AUC score for {} model is'.format(models_names[lda_model]),'\n',roc_auc_scor
e (y_test, y_test_pred_lda))

print('\n')

print('Confusion Matrix for {} model is'.format(models_names[lda_model]),'\n',confusion
_matrix(y_test, y_test_pred_lda))


# predict probabilities
probs = lda_model.predict_proba (X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

```
Classification report for Linear Discriminate Analysis model is
               precision    recall  f1-score   support

         0.0       0.64      0.63      0.63       147
         1.0       0.53      0.54      0.54       115

    accuracy                           0.59       262
   macro avg       0.59      0.59      0.59       262
weighted avg       0.59      0.59      0.59       262



Accuracy for Linear Discriminate Analysis model is
 0.5916030534351145
ROC AUC score for Linear Discriminate Analysis model is
 0.5858917480035493


Confusion Matrix for Linear Discriminate Analysis model is
 [[93 54]
 [53 62]]
AUC: 0.598
```



In [325]:

```
# lets check the probabilities:

ytest_predict_prob_lda = lda_model.predict_proba(X_test)
ytest_predict_prob_lda.mean()
```

Out[325]:

```
0.5
```

In [326]:

```
pd.DataFrame (ytest_predict_prob_lda).head()
```

Out[326]:

	0	1
0	0.710318	0.289682
1	0.735975	0.264025
2	0.687159	0.312841
3	0.538032	0.461968
4	0.419467	0.580533

Final Model: Compare Both the models and write inference which model is best/optimized.

Performance Matrix	Models				Models	
	Logistic Regression		After Fine Tuning		Linear Discriminate Analysis	
	Training	Testing	Training	Testing	Training	Testing
Accuracy	0.51	0.54	0.51	0.56	0.58	0.59
AUC	0.58	0.59	0.6	0.59	0.63	0.59
ROC AUC score	0.49	0.49	0.49	0.5	0.58	0.58
Precision						
0	0.53	0.56	0.53	0.56	0.61	0.64
1	0.43	0.43	0.43	0	0.56	0.53
Recall						
0	0.9	0.89	0.9	1	0.63	0.63
1	0.09	0.17	0.09	0	0.53	0.54
f1 score						
0	0.66	0.69	0.66	0.72	0.62	0.63
1	0.15	0.17	0.15	0	0.55	0.54
Confusion Matrix	[290 34 260 26]	[131 16 103 12]	[290 34 260 26]	[147 0 115 0	[204 120 133 153]	[93 54 53 62]

Comparing Probabilities:

	Probabilites			Probabilites	
	Training Set			Testing Set	
	0	1		0	1
0	0.600151	0.399849	0	0.710318	0.289682
1	0.598449	0.401551	1	0.735975	0.264025
2	0.584966	0.415034	2	0.687159	0.312841
3	0.562176	0.437824	3	0.538032	0.461968
4	0.535245	0.464755	4	0.419467	0.580533

Inferences:

After comparing both the models we can see that Linear Discriminate Analysis performs good in all the parameters. Even though Score like Accuracy, Recall, Precision is not up to the mark, but if we just want to compare both the models then Linear Discriminate Analysis is our Choice and also Data is small hence LDA performs good.

In [ ]:

```

```

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

Company wants to know how many employees will choose for Holliday_Package as per the Business Problem, having this objective we have built two classify models.

Insights:

While doing Exploratory Data Analysis (EDA) we come to know that Employee having more than 2 children which are old (older than 7 years) are going for Holiday package also maximum is 6 children and median is 1 so this gives us an idea that Older children parents are opting the Holliday Package but also same amount of ratio is for not obtaining the Holliday Package.

So its 50-50 %, company can try for all those employees who are having no of older children with more focusing on children age group.

Employee opting for Holliday_Package seems to have Less Salary as Compared to the Employee having Much salary.

Also, Age increases no of young children (younger than 7 years) decreases and Below 30 Years almost everyone is having the Young children which proves that the Dependency on the employee is in early stage.

Foreigners have a smaller number of older children as compared to the Non-Foreigners.

We also know Years of formal education is maximum of 18 years (12+4+2) and minimum is 1 (which is possible for those employees who must be a security or in pantry) and 19 years of Education for those people who have the on an average of 40 years.

We know that if the Employee is a Foreigner then the chances of getting the Holliday_Package is More as compared to the not a Foreigner.

The important factors based on which the company will focus on employees to sell their packages.

Employees having Older Children
Employees having Less Salary going for Holiday Package
Foreigner employee is having higher chance for Holiday Package

Recommendations:

After building both the models we predicted the business objective, but our Model's performance is not appropriate so that we cannot put our model into the Production. Linear Discriminate Analysis (LDA) gave better results when compared to the Logistic Regression For LDA, we consider only (1's) for Precision and Recall we are getting on average of 55 % means Model is only Predicting 55% while 45 % it is giving the incorrect predictions, and this is where we know that our Model is not Perfect.

Selecting Precision or Recall is totally dependent on Domain Expert.

But if we as a Business Analyst wants to choose then we will be selecting the Recall and in this case Recall (Employee wants Holiday Package but our model is predicting Opposite and its around 45 %, which is in real world is inappropriate to use this type of Model. Also, Accuracy, AUC, ROC_AUC score is almost same for Training and Testing Data, which is also very less, and Strength of the Model is very weak.

What Can we Do:

When we checked the pair plot for the Data we saw that there are only 2 Features which are Classifying or Separating the Target/Dependent variable and remaining 4 features are not classifying, hence based on only 2 features our model will perform poor and it will be underfit or overfit like we had experienced after building 2 models.

Hence, we want to add more Features to the Data which will be relevant by the company itself.

We can run a choice of Travel/Holiday place in organisation where we will know the Maximum preferred Travel location.

So, by adding good specific features with the help of Stake Holders/Domain Expert Our model could give Better Results.