PROBLEM – 1

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.

Solution:- let's start with importing important libraries and loading the data into Jupyter notebook

```
#importing nupy and pandas
import numpy as np
import pandas as pd
```

```
#loding data
stone  = pd.read_csv(r'C:\Users\abhay\Downloads\cubic_zirconia.csv')
```

- Checking head of dataset to check if data loaded correctly and also to check parameters of the data

```
#checking head of data
stone.head()
```

	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

- Checking info of the data

```
#checking info
stone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   26967 non-null  int64
 1   carat        26967 non-null  float64
 2   cut          26967 non-null  object
 3   color        26967 non-null  object
 4   clarity      26967 non-null  object
 5   depth        26270 non-null  float64
 6   table        26967 non-null  float64
 7   x            26967 non-null  float64
 8   y            26967 non-null  float64
 9   z            26967 non-null  float64
 10  price        26967 non-null  int64
dtypes: float64(6), int64(2), object(3)
memory usage: 2.3+ MB
```

- Checking shape of data

```
#checking shape
stone.shape
```

```
(26967, 11)
```

- Describing data to check various statistical aspects of data

```
#describing data
stone.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	26967.0	13484.000000	7784.846691	1.0	6742.50	13484.00	20225.50	26967.00
carat	26967.0	0.798375	0.477745	0.2	0.40	0.70	1.05	4.50
depth	26270.0	61.745147	1.412860	50.8	61.00	61.80	62.50	73.60
table	26967.0	57.456080	2.232068	49.0	56.00	57.00	59.00	79.00
x	26967.0	5.729854	1.128516	0.0	4.71	5.69	6.55	10.23
y	26967.0	5.733569	1.166058	0.0	4.71	5.71	6.54	58.90
z	26967.0	3.538057	0.720624	0.0	2.90	3.52	4.04	31.80
price	26967.0	3939.518115	4024.864666	326.0	945.00	2375.00	5360.00	18818.00

- Checking for null values

```
#checking null values
stone.isnull().sum()
```

```
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
```

- There are 697 null values in “depth” variable of the data

- Checking for duplicates

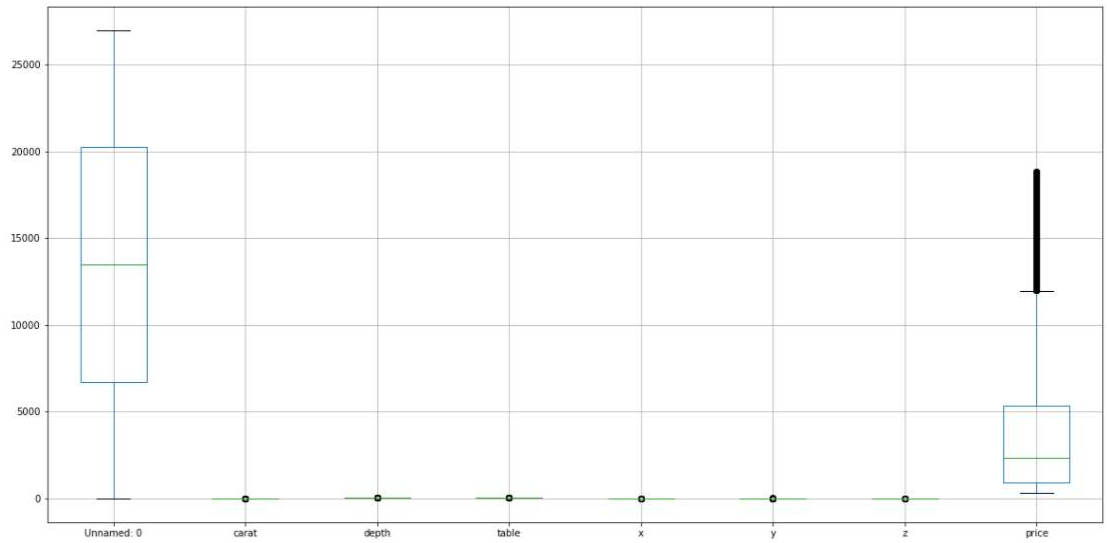
```
#checking duplicates
stone.duplicated().sum()
```

```
0
```

- Plotting box plot to check for outliers in data

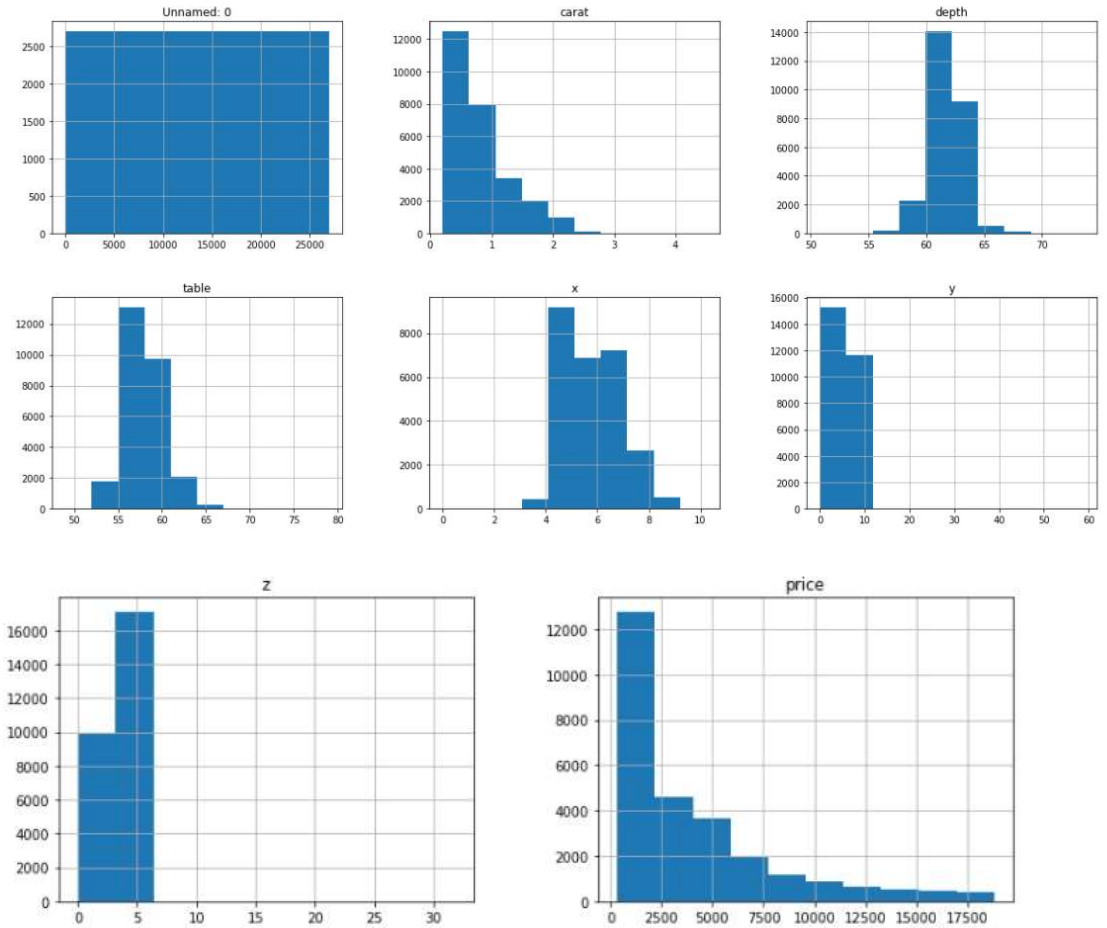
```
#checking for outliers using box plot
stone.boxplot(figsize=(20,10))
```

```
<AxesSubplot:>
```



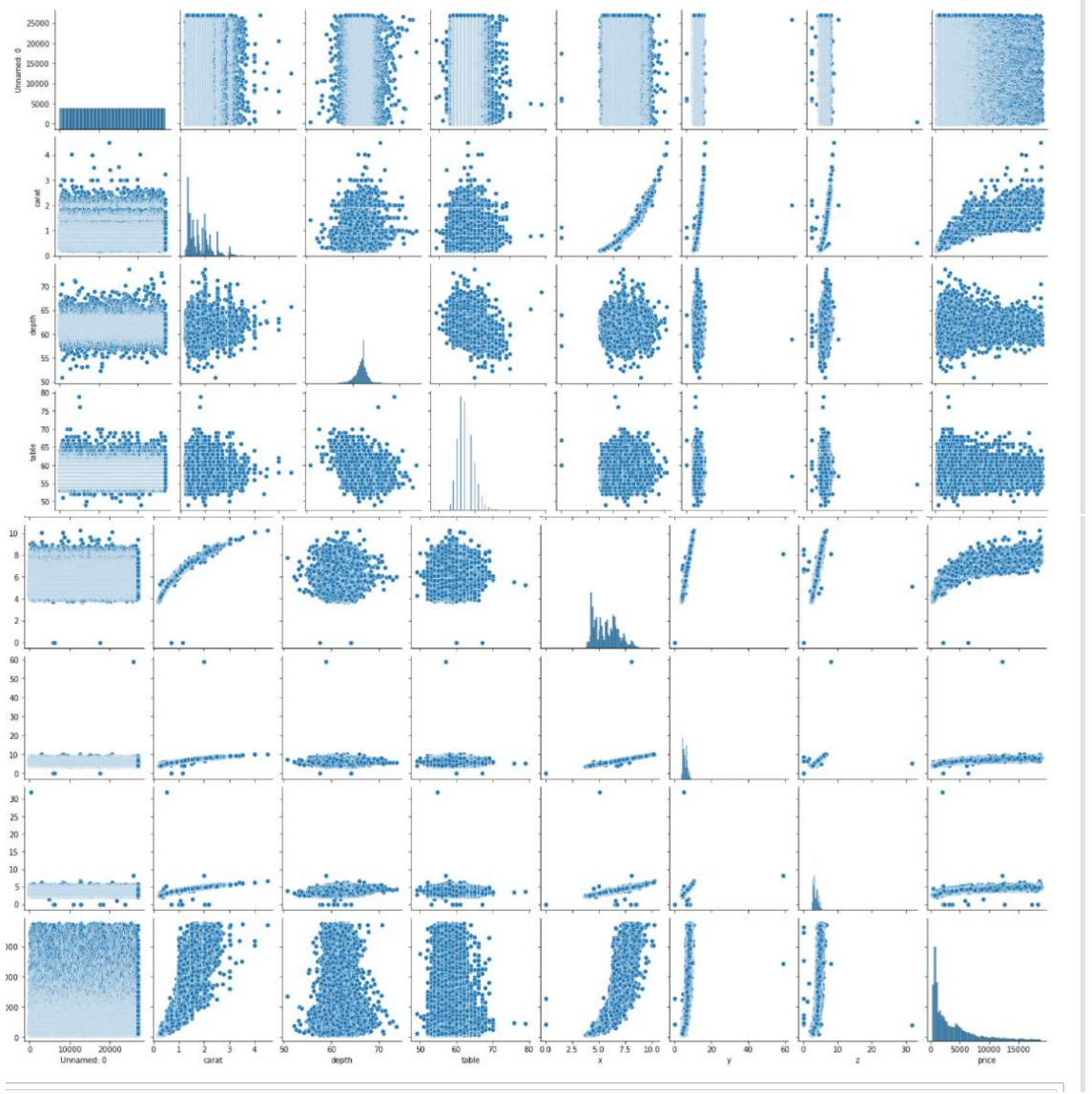
- Checking on skewness of data

```
#checking for skewness
stone.hist(figsize=(20,15));
```



- Plotting pair plot of data

```
#the pair plot:-
sns.pairplot(data = stone)
plt.show()
```



- Checking covariance among variables of the data

```
#checking covariance
stone.cov()
```

	Unnamed: 0	carat	depth	table	x	y	z	price
Unnamed: 0	6.060384e+07	12.978981	-17.458374	66.324405	40.641354	62.130324	9.429493	8.304578e+04
carat	1.297898e+01	0.228241	0.023844	0.193742	0.526403	0.524250	0.323839	1.773678e+03
depth	-1.745837e+01	0.023844	1.996174	-0.939265	-0.029813	-0.040760	0.100411	-1.459691e+01
table	6.632440e+01	0.193742	-0.939265	4.982127	0.494228	0.474596	0.239574	1.140420e+03
x	4.064135e+01	0.526403	-0.029813	0.494228	1.273549	1.266851	0.777946	4.025446e+03
y	6.213032e+01	0.524250	-0.040760	0.474596	1.266851	1.359690	0.780563	4.018538e+03
z	9.429493e+00	0.323839	0.100411	0.239574	0.777946	0.780563	0.519298	2.466906e+03
price	8.304578e+04	1773.677848	-14.596911	1140.419986	4025.446081	4018.537829	2466.905683	1.619954e+07

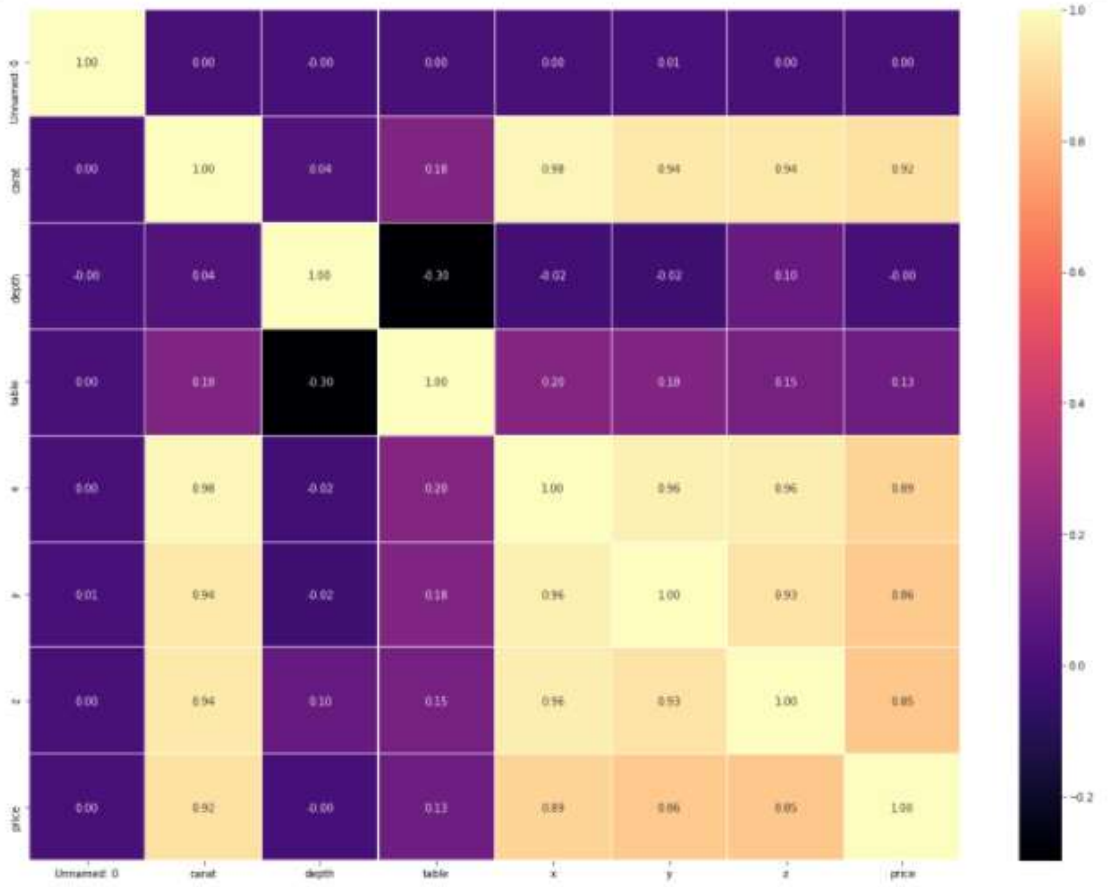
- Checking correlation of data

```
#checking correlation
stone.corr()
```

	Unnamed: 0	carat	depth	table	x	y	z	price
Unnamed: 0	1.000000	0.003490	-0.001588	0.003817	0.004626	0.006844	0.001681	0.002650
carat	0.003490	1.000000	0.035364	0.181685	0.976368	0.941071	0.940640	0.922416
depth	-0.001588	0.035364	1.000000	-0.298011	-0.018715	-0.024735	0.101624	-0.002569
table	0.003817	0.181685	-0.298011	1.000000	0.196206	0.182346	0.148944	0.126942
x	0.004626	0.976368	-0.018715	0.196206	1.000000	0.962715	0.956606	0.886247
y	0.006844	0.941071	-0.024735	0.182346	0.962715	1.000000	0.928923	0.856243
z	0.001681	0.940640	0.101624	0.148944	0.956606	0.928923	1.000000	0.850536
price	0.002650	0.922416	-0.002569	0.126942	0.886247	0.856243	0.850536	1.000000

- Plotting heat map for correlation

```
#heat map
fig,ax = plt.subplots(figsize=(20, 15))
sns.heatmap(stone.corr(), ax=ax, annot=True, linewidths=0.05, fmt= '.2f',cmap="magma")
plt.show()
```



- Checking standard deviation of all the variables

```
#checking standard deviations of each variable
print(stone.std())
```

```
Unnamed: 0    7784.846691
carat            0.477745
depth            1.412860
table            2.232068
x                1.128516
y                1.166058
z                0.720624
price         4024.864666
dtype: float64
```

- Dropping sl no column (1st Column) as it won't play any important part in further analysis and it also does not relate with any other variables in the data and checking the info of data after dropping the column.

```
# deleting the unnamed column as its a seriam number column and wont play any part in our further analysis
stone = stone.drop('Unnamed: 0', axis=1)
```

```
# checking the info of the data set again if unnamed column is deleted or not
stone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
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: 2.1+ MB
```

INFERENCE AFTER UNIVARIATE AND BIVARIATE ANALYSIS: -

- ✓ The data set has 10 variables, 9 independent and 1 dependent variable and below are the details of the variable's: -

Variable Name	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.

- ✓ Out of these 10 variables 7 are of integer (int) data types and 3 are of string/object data types.
- ✓ There are 26967 observations. (number of rows).
- ✓ There are 697 null values in "depth" variable and rest all variable doesn't have any null values.
- ✓ Data has NIL duplicate observations.
- ✓ Almost all variable has outliers derived from box plot.
- ✓ Among all the variables only variable "depth" is normally distributed rest all other variables are right or left skewed.
- ✓ Form pair plot and correlation matrix we can conclude that the variable "caret","x","y" and "z" are highly corelated and plays an important role in predicting the "price" which is the dependent variable.
- ✓ Variable's "X", "y" and "z" have "0" (Zero) values in it.

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?

Solution:-

- Checking for null values:-

```
#checking for null values
stone.isnull().sum()
```

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

- There are 697 null values in variable “Depth” and rest all other variables doesn’t have null values.
- Imputing null values with median.

```
# treating null values and replacing them with mean value of the variable
stone.depth=stone.depth.fillna(stone.depth.median())
```

- Checking for null values post imputation

```
#checking for null values after imputing
stone.isnull().sum()
```

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

- Now there no null values in the data
- Checking for “0” (zero) values in variables and found that the variable’s “x”, “y” and “z” have “0” values in them. And these “0” values need to be treated as these variable’s denotes length, width and height of cubic zirconia in mm which practically cannot be zero so these values need treatment and we can impute these values by median.

```
# checking for value "0" in variable "carat"
stone[stone['carat'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# checking for value "0" in variable "price"
stone[stone['price'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# checking for value "0" in variable "depth"
stone[stone['depth'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# checking for value "0" in variable "table"
stone[stone['table'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# checking for value "0" in variable "x"
stone[stone['x'] == 0]
```

	carat	cut	color	clarity	depth	table	x	y	z	price
5821	0.71	Good	F	SI2	64.1	60.0	0.0	0.0	0.0	2130
6215	0.71	Good	F	SI2	64.1	60.0	0.0	0.0	0.0	2130
17506	1.14	Fair	G	VS1	57.5	67.0	0.0	0.0	0.0	6381

```
# checking for value "0" in variable "y"
stone[stone['y'] == 0]
```

	carat	cut	color	clarity	depth	table	x	y	z	price
5821	0.71	Good	F	SI2	64.1	60.0	0.0	0.0	0.0	2130
6215	0.71	Good	F	SI2	64.1	60.0	0.0	0.0	0.0	2130
17506	1.14	Fair	G	VS1	57.5	67.0	0.0	0.0	0.0	6381

```
# checking for value "0" in variable "z"
stone[stone['z'] == 0]
```

	carat	cut	color	clarity	depth	table	x	y	z	price
5821	0.71	Good	F	SI2	64.1	60.0	0.00	0.00	0.0	2130
6034	2.02	Premium	H	VS2	62.7	53.0	8.02	7.95	0.0	18207
6215	0.71	Good	F	SI2	64.1	60.0	0.00	0.00	0.0	2130
10827	2.20	Premium	H	SI1	61.2	59.0	8.42	8.37	0.0	17265
12498	2.18	Premium	H	SI2	59.4	61.0	8.49	8.45	0.0	12631
12689	1.10	Premium	G	SI2	63.0	59.0	6.50	6.47	0.0	3696
17506	1.14	Fair	G	VS1	57.5	67.0	0.00	0.00	0.0	6381
18194	1.01	Premium	H	I1	58.1	59.0	6.66	6.60	0.0	3167
23758	1.12	Premium	G	I1	60.4	59.0	6.71	6.67	0.0	2383

- Imputing the “0” values with median

```
# replacing "0" values with median of the variable
stone.x = stone.x.replace(0,stone.x.median())
```

```
# checking for "0" values in variable "x" post imputation
stone[stone['x'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# replacing "0" values with median of the variable
stone.y = stone.y.replace(0,stone.y.median())
```

```
# checking for "0" values in variable "y" post imputation
stone[stone['y'] == 0]
```

carat	cut	color	clarity	depth	table	x	y	z	price

```
# replacing "0" values with median of the variable
stone.z = stone.z.replace(0,stone.z.median())
```

```
# checking for "0" values in variable "z" post imputation
stone[stone['z'] == 0]
```

- Lets describe the data post treating “0” values:-

```
# describing data after treating "0" values
stone.describe()
```

	carat	depth	table	x	y	z	price
count	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000
mean	0.798375	61.746564	57.456080	5.730487	5.734204	3.539232	3939.518115
std	0.477745	1.394509	2.232068	1.126897	1.164488	0.717718	4024.864666
min	0.200000	50.800000	49.000000	3.730000	3.710000	1.070000	326.000000
25%	0.400000	61.100000	56.000000	4.710000	4.720000	2.900000	945.000000
50%	0.700000	61.800000	57.000000	5.690000	5.710000	3.520000	2375.000000
75%	1.050000	62.500000	59.000000	6.550000	6.540000	4.040000	5360.000000
max	4.500000	73.600000	79.000000	10.230000	58.900000	31.800000	18818.000000

- We can see that the min values of “x”, “y” and “z” is changed from “0” and clearly indicates that the “0” values have been treated from the data.

- Scaling is necessary in this case as the magnitude of variable's "caret" and "price" is different from magnitudes of variable's "x", "y" and "z" and they differ in std deviation, min and max values as derived from describing the data. However, scaling data in a linear regression will only affect the co-efficient values and will bring intercept close to "0" (zero) and the accuracy of the model remains unaffected. (In this business report I have shown the model built with both the data one with scaling and other without scaling) and we can notice that the accuracy remains unaffected.

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.

Solution:-

- Getting unique values and their respective counts of object variable types that is "cut", "color" and "clarity" :-

```
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
```

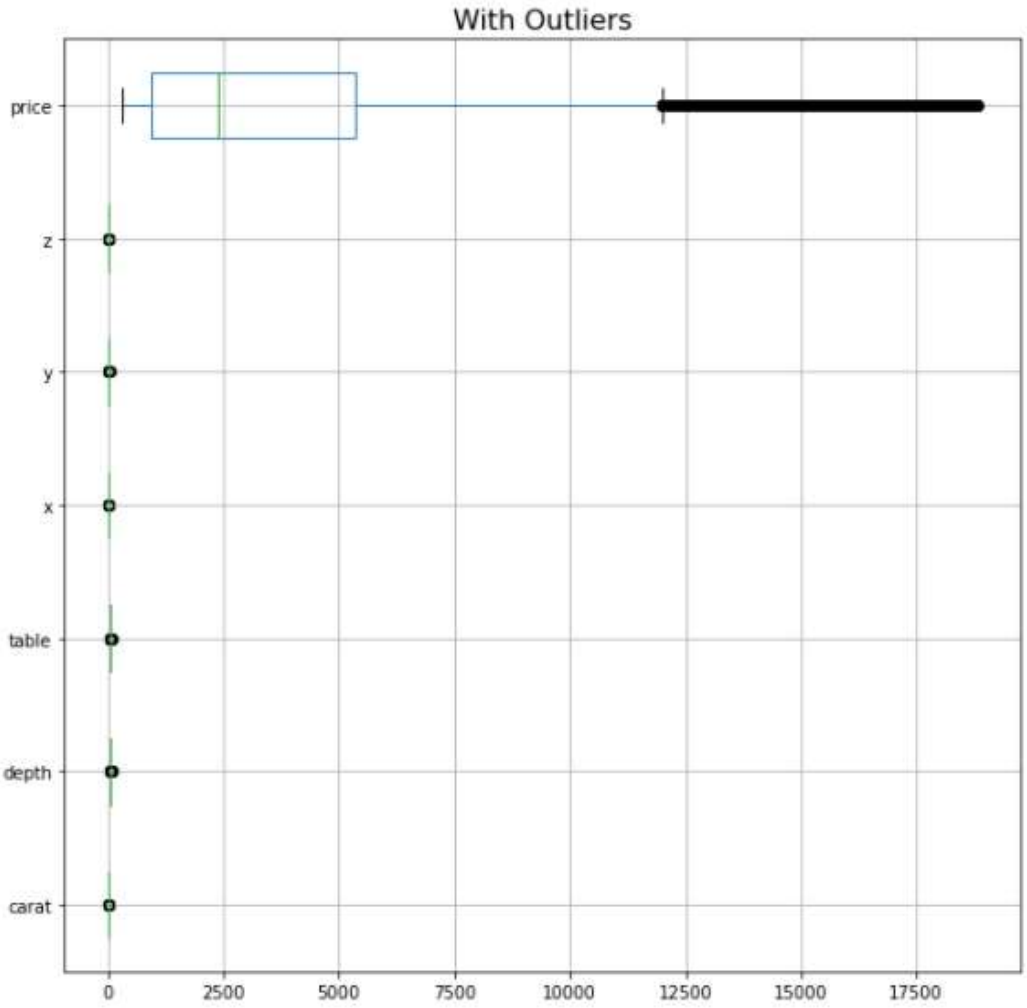
- Encoding object variables to get dummy observations

```
# encoding string data type variable to get dummies
stone = pd.get_dummies(stone, columns=['cut','color','clarity'])
```

- Describing data post encoding

	count	mean	std	min	25%	50%	75%	max
carat	26967.0	0.798375	0.477745	0.20	0.40	0.70	1.05	4.50
depth	26967.0	61.746564	1.394509	50.80	61.10	61.80	62.50	73.60
table	26967.0	57.456080	2.232068	49.00	56.00	57.00	59.00	79.00
x	26967.0	5.730487	1.126897	3.73	4.71	5.69	6.55	10.23
y	26967.0	5.734204	1.164488	3.71	4.72	5.71	6.54	58.90
z	26967.0	3.539232	0.717718	1.07	2.90	3.52	4.04	31.80
price	26967.0	3939.518115	4024.864666	326.00	945.00	2375.00	5360.00	18818.00
cut_Fair	26967.0	0.028961	0.167701	0.00	0.00	0.00	0.00	1.00
cut_Good	26967.0	0.090518	0.286928	0.00	0.00	0.00	0.00	1.00
cut_Ideal	26967.0	0.401083	0.490127	0.00	0.00	0.00	1.00	1.00
cut_Premium	26967.0	0.255831	0.436335	0.00	0.00	0.00	1.00	1.00
cut_Very Good	26967.0	0.223607	0.416669	0.00	0.00	0.00	0.00	1.00
color_D	26967.0	0.124003	0.329592	0.00	0.00	0.00	0.00	1.00
color_E	26967.0	0.182334	0.386127	0.00	0.00	0.00	0.00	1.00
color_F	26967.0	0.175362	0.380284	0.00	0.00	0.00	0.00	1.00
color_G	26967.0	0.209923	0.407261	0.00	0.00	0.00	0.00	1.00
color_H	26967.0	0.152112	0.359136	0.00	0.00	0.00	0.00	1.00
color_I	26967.0	0.102755	0.303645	0.00	0.00	0.00	0.00	1.00
color_J	26967.0	0.053510	0.225052	0.00	0.00	0.00	0.00	1.00
clarity_I1	26967.0	0.013535	0.115552	0.00	0.00	0.00	0.00	1.00
clarity_IF	26967.0	0.033152	0.179036	0.00	0.00	0.00	0.00	1.00
clarity_SI1	26967.0	0.243668	0.429303	0.00	0.00	0.00	0.00	1.00
clarity_SI2	26967.0	0.169652	0.375334	0.00	0.00	0.00	0.00	1.00
clarity_VS1	26967.0	0.151778	0.358812	0.00	0.00	0.00	0.00	1.00
clarity_VS2	26967.0	0.226165	0.418355	0.00	0.00	0.00	0.00	1.00
clarity_VVS1	26967.0	0.068194	0.252084	0.00	0.00	0.00	0.00	1.00
clarity_VVS2	26967.0	0.093855	0.291633	0.00	0.00	0.00	0.00	1.00

- Checking for outliers:-



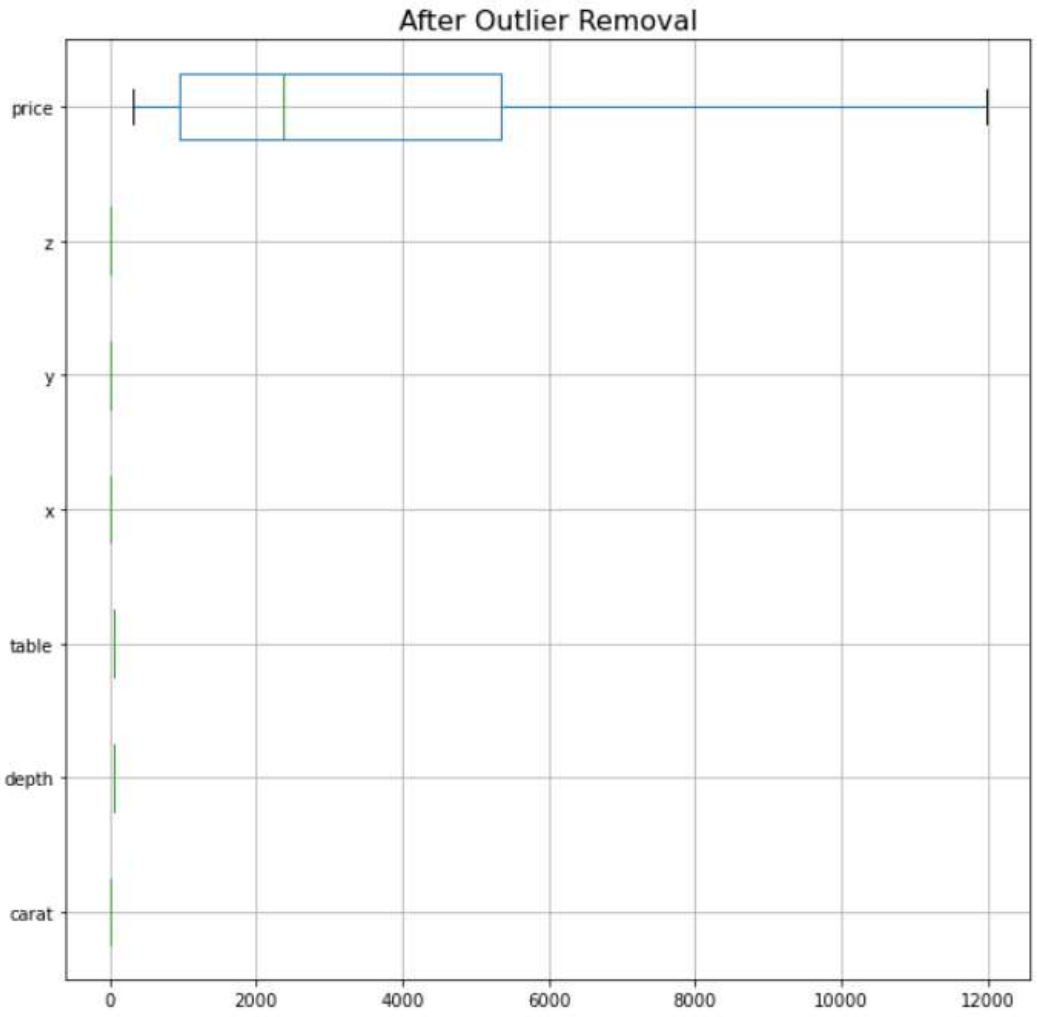
- Almost all variables have outliers.
- Treating outliers using interquartile range

```
# 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
```

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

```
# checking outliers post treatment
plt.figure(figsize=(10,10))
stone[cont].boxplot(vert=0)
plt.title('After Outlier Removal',fontsize=16)
plt.show()
```

- Checking outliers post treatment:-



- Splitting data into dependent and independent variables.

```
# splitting data into independent and dependent variables
X = stone.drop('price' , axis=1)

y = stone.pop("price")
```

APPLY LINEAR REGTESSION OVER UNSCALED DATA

- Splitting data into training and testing data set (70:30) and checking its shape to understand if splitting is done uniformly.

```
#splitting data into train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

```
#checking the dimentsions of training and test data
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
X_train (18876, 23)
X_test (8091, 23)
y_train (18876,)
y_test (8091,)
```

- Applying linear regression to train data set

```
# invoke the LinearRegression function and find the bestfit model on training data
from sklearn.linear_model import LinearRegression
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

```
LinearRegression()
```

- Getting co-efficient of variables:-

```
# Let us explore the coefficients for each of the independent attributes
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_model.coef_[idx]))
```

```
The coefficient for carat is 9218.67850430357
The coefficient for depth is -0.5630548626425688
The coefficient for table is -22.770879950337072
The coefficient for x is -1263.8023705631904
The coefficient for y is 999.8589177205646
The coefficient for z is -379.4714060265337
The coefficient for cut_Fair is -413.0399893956472
The coefficient for cut_Good is -49.31863418184363
The coefficient for cut_Ideal is 202.84754272597561
The coefficient for cut_Premium is 181.23085634387695
The coefficient for cut_Very Good is 78.28022450763463
The coefficient for color_D is 710.2017969233009
The coefficient for color_E is 515.7968304049731
The coefficient for color_F is 440.51143579969414
The coefficient for color_G is 289.36714034920027
The coefficient for color_H is -133.99261074447745
The coefficient for color_I is -615.8188949716563
The coefficient for color_J is -1206.0656977610329
The coefficient for clarity_I1 is -2834.3680619656425
The coefficient for clarity_IF is 1214.0179558999698
The coefficient for clarity_SI1 is -229.4162122543199
The coefficient for clarity_SI2 is -1042.9648642748944
The coefficient for clarity_VS1 is 582.6746998867205
The coefficient for clarity_VS2 is 310.14220034766464
The coefficient for clarity_VVS1 is 1016.339397864197
The coefficient for clarity_VVS2 is 983.5748844963122
```

- Getting intercept of linear regression model

```
# Let us check the intercept for the model
intercept = regression_model.intercept_

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

```
The intercept for our model is 233.19977311655748
```

- Getting R-square of training and testing dataset

```
# R square on training data
regression_model.score(X_train, y_train)
```

```
0.9409100676795843
```

```
# R square on testing data
regression_model.score(X_test, y_test)
```

```
0.9404523816541941
```

- Getting RMSE for training and testing dataset:-

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

```
845.5333400727277
```

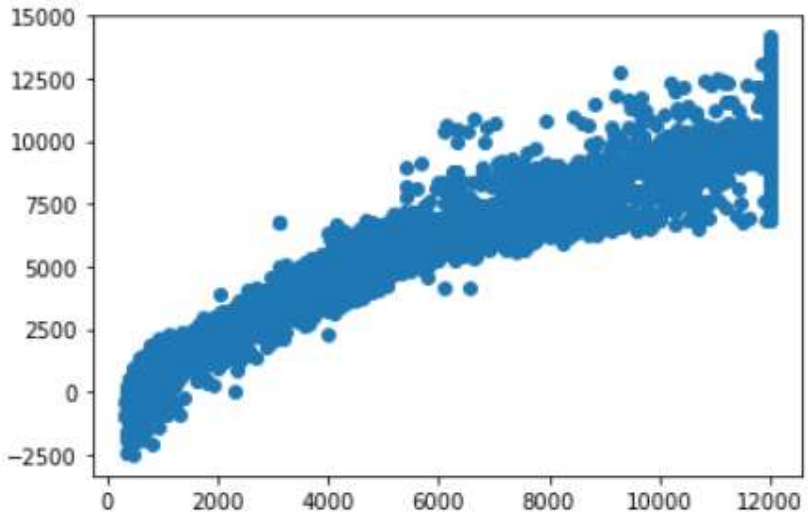
```
#RMSE on Testing data
predicted_test=regression_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

```
842.6556351155222
```

- Plotting linear regression model:-

```
# plotting the actual test set with predicted values
plt.scatter(y_test, predicted_test)
```

```
<matplotlib.collections.PathCollection at 0x1a24c0ffa58>
```



APPLY LINEAR REGTESSION OVER SCALED DATA

- Scaling data using z-score after treating null values, “0” values, outliers and encoding variables with string data types.

```
from scipy.stats import zscore
stone_scaled  = stone.apply(zscore)
```

- Describing data post scaling

```
stone_scaled.describe()
```

	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	cut_Premium
count	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04	2.696700e+04
mean	2.549474e-16	-4.038892e-15	-5.372850e-16	5.002117e-17	4.806067e-16	9.046381e-16	5.001911e-17	-2.946762e-16	-1.083924e-15	-1.307236e-15
std	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00	1.000019e+00
min	-1.283660e+00	-2.255418e+00	-2.751722e+00	-1.776019e+00	-1.808415e+00	-3.373895e+00	-9.830272e-01	-3.154791e-01	-8.183397e-01	-5.863285e-01
25%	-8.511548e-01	-5.325255e-01	-6.655736e-01	-9.059102e-01	-9.051956e-01	-9.168035e-01	-8.046833e-01	-3.154791e-01	-8.183397e-01	-5.863285e-01
50%	-2.023971e-01	4.177220e-02	-2.019851e-01	-3.580172e-02	-1.986200e-02	-2.592840e-02	-3.926774e-01	-3.154791e-01	-8.183397e-01	-5.863285e-01
75%	5.544870e-01	6.160699e-01	7.251919e-01	7.277629e-01	7.223874e-01	7.212572e-01	4.673489e-01	-3.154791e-01	1.221986e+00	1.705528e+00
max	2.662950e+00	2.338963e+00	2.811340e+00	3.178273e+00	3.163762e+00	3.178348e+00	2.375397e+00	3.169783e+00	1.221986e+00	1.705528e+00

- We can see the changes in statistical values of variables
- Splitting data into dependent and independent variables post scaling the data

```
# splitting data into independent and dependent variables
X = stone_scaled.drop('price' , axis=1)

y = stone_scaled.pop("price")
```

- Splitting data into training and testing dataset's and checking the shape of the same to get confirmation over uniform splitting

```
#splitting data into train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

```
#checking the dimentsions of training and test data
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
X_train (18876, 23)
X_test (8091, 23)
y_train (18876,)
y_test (8091,)
```

- Applying linear regression model into training dataset

```
# invoke the LinearRegression function and find the bestfit model on training data
from sklearn.linear_model import LinearRegression
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

```
LinearRegression()
```

- Checking co-efficient of training dataset

```
# Let us explore the coefficients for each of the independent attributes
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_model.coef_[idx]))
```

```
The coefficient for carat is 9218.67850430357
The coefficient for depth is -0.5630548626425688
The coefficient for table is -22.770879950337072
The coefficient for x is -1263.8023705631904
The coefficient for y is 999.8589177205646
The coefficient for z is -379.4714060265337
The coefficient for cut_Fair is -413.0399893956472
The coefficient for cut_Good is -49.31863418184363
The coefficient for cut_Ideal is 202.84754272597561
The coefficient for cut_Premium is 181.23085634387695
The coefficient for cut_Very Good is 78.28022450763463
The coefficient for color_D is 710.2017969233009
The coefficient for color_E is 515.7968304049731
The coefficient for color_F is 440.51143579969414
The coefficient for color_G is 289.36714034920027
The coefficient for color_H is -133.99261074447745
The coefficient for color_I is -615.8188949716563
The coefficient for color_J is -1206.0656977610329
The coefficient for clarity_I1 is -2834.3680619656425
The coefficient for clarity_IF is 1214.0179558999698
The coefficient for clarity_SI1 is -229.4162122543199
The coefficient for clarity_SI2 is -1042.9648642748944
The coefficient for clarity_VS1 is 582.6746998867205
The coefficient for clarity_VS2 is 310.14220034766464
The coefficient for clarity_VVS1 is 1016.339397864197
The coefficient for clarity_VVS2 is 983.5748844963122
```

- Checking intercept of the model

```
# Let us check the intercept for the model
intercept = regression_model.intercept_

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

```
The intercept for our model is 0.0015309472423281909
```

- We can see that after scaling the data the intercept of linear regression model changes near to “0”.
- R-square values of training and test dataset’s

```
# R square on training data
regression_model.score(X_train, y_train)
```

```
0.9409104958833862
```

```
# R square on testing data
regression_model.score(X_test, y_test)
```

```
0.9404530609704427
```

- RSME of training and testing dataset’s

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

```
0.2436108271250871
```

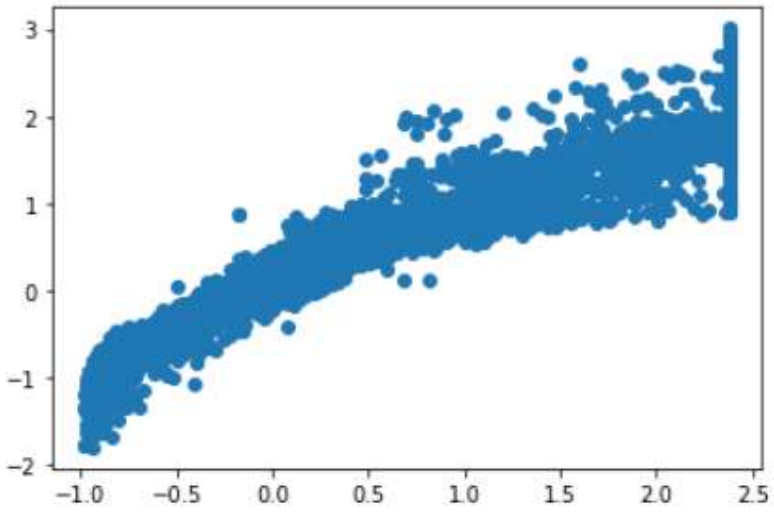
```
#RMSE on Testing data
predicted_test=regression_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

```
0.24278121205080586
```

- Plotting linear regression model

```
# plotting the actual test set with predicted values
plt.scatter(y_test, predicted_test)
```

```
<matplotlib.collections.PathCollection at 0x1cd96c45630>
```



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

Solution:- Below are the insights and recommendations after above analysis

- ✓ The variable's "carat", "x", "y" and "z" are contributing the maximum towards the price.
- ✓ "X", "y" and "z" are highly correlated.
- ✓ The variable's "depth" and "table" doesn't impact the price so much.
- ✓ Higher the Carat more the price.
- ✓ Higher the dimensions of the stone (that is variable's "x", "y" and "z") more the price.
- ✓ Out of all types in "cut" variable the "Ideal" types of cut is giving the best price.
- ✓ The types from color variable "D", "E", "F" and "G" increases the price. However other color types "H", "I" and "J" have a negative impact in the price.
- ✓ Clarity of "IF" category have the best price and the clarity into "I1", "SI1" and "SI2" is impacting the price in negative manner.

RECOMMENDATIONS: -

- ✓ The business needs to eliminate/minimize the sales of gem stones with color categories of "H", "I" and "J".
- ✓ To concentrate more on cut types of "Ideal", "Premium" and "Verygood".
- ✓ Business should avoid clarity category of "I1", "SI1" and "SI2".
- ✓ For high level spending customers they should target product with ideal cut, "D" color and "IF" clarity.
- ✓ For mid-range spenders they can have varieties in combination of various color, cut and clarity. My color they should recommend "D" or "E", for cut they should recommend "premium" or "very good" and from the clarity they can recommend "VVS1" or "VVS2". Any combination of above will work well when the target customers are of middle class.
- ✓ For customers with low on spend the business should recommend, "verygood" cut along with "F" or "G" color type and clarity of "VS1" and "VS2".

- ✓ Neither the less apart from the features mentioned above the dimensions ("X", "Y" and "Z") plays a important part in deciding the price of the gem stone.

----------END OF PROBLEM 1----------

PROBLEM – 2

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.

Solution:-

- Lets start with importing libraries and loading data into jupyter notepad.

```
#importing nupy and pandas
import numpy as np
import pandas as pd
```

```
#loding data
holiday  = pd.read_csv(r'C:\Users\abhay\Downloads\Holiday_package.csv')
```

- Checking head of data

```
#checking head of data
holiday.head()
```

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

We can see that data has been loaded correctly.

- Checking info of data

```
#checking info
holiday.info()
```

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

- From the above info data we can conclude that there are 8 variables, 7 independent and 1 dependent variable.
- Out of 8 variable's we have 2 variable's with object/string data type and others are of int data type.
- Checking shape of data

```
#checking shape
holiday.shape
```

```
(872, 8)
```

- Describing data

```
#describing data
holiday.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	872.0	436.500000	251.869014	1.0	218.75	436.5	654.25	872.0
Salary	872.0	47729.172018	23418.668531	1322.0	35324.00	41903.5	53469.50	236961.0
age	872.0	39.955275	10.551675	20.0	32.00	39.0	48.00	62.0
educ	872.0	9.307339	3.036259	1.0	8.00	9.0	12.00	21.0
no_young_children	872.0	0.311927	0.612870	0.0	0.00	0.0	0.00	3.0
no_older_children	872.0	0.982798	1.086786	0.0	0.00	1.0	2.00	6.0

- Checking for null values

```
#checking null values
holiday.isnull().sum()
```

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

- Checking for duplicate values

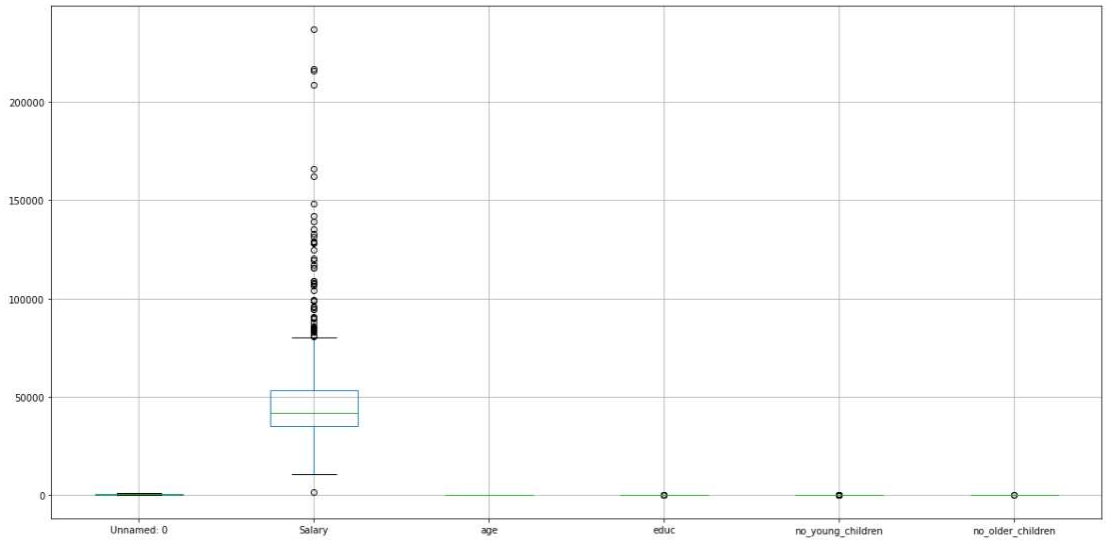
```
#checking duplicates
holiday.duplicated().sum()
```

```
0
```

- Plotting box plot to check for outliers

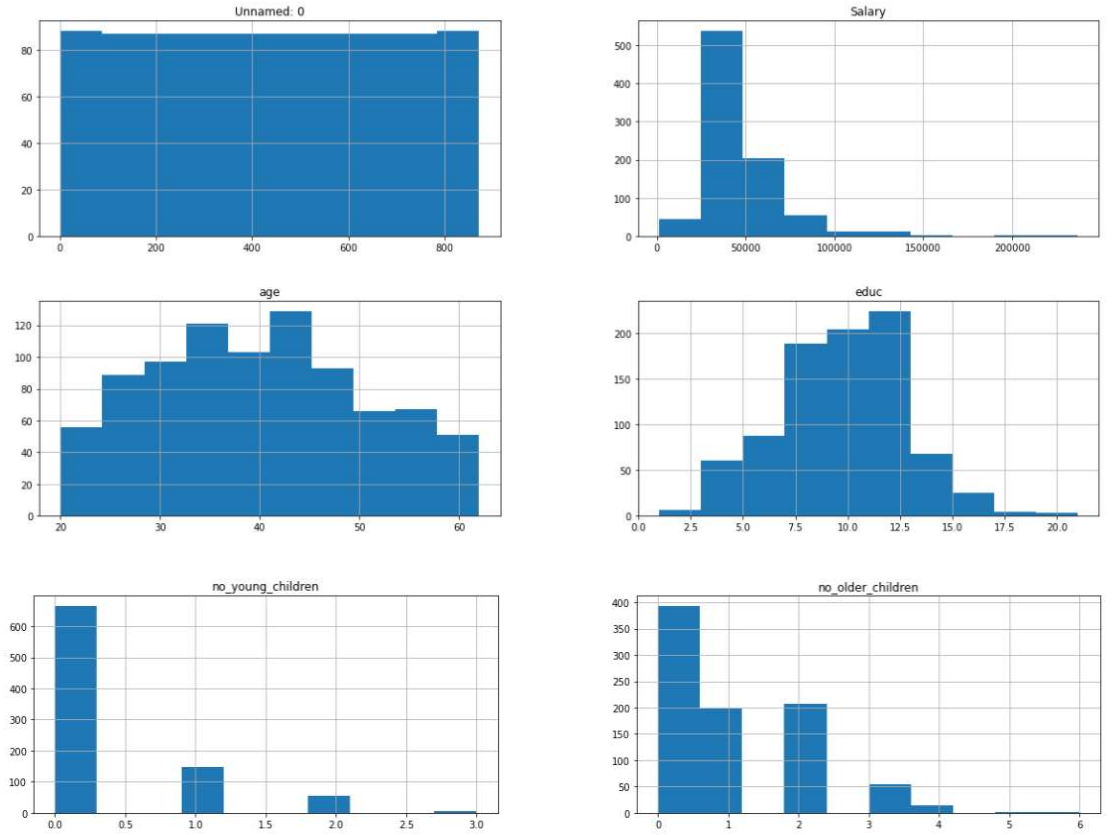
```
#checking for outliers using box plot
holiday.boxplot(figsize=(20,10))
```

```
<AxesSubplot:>
```

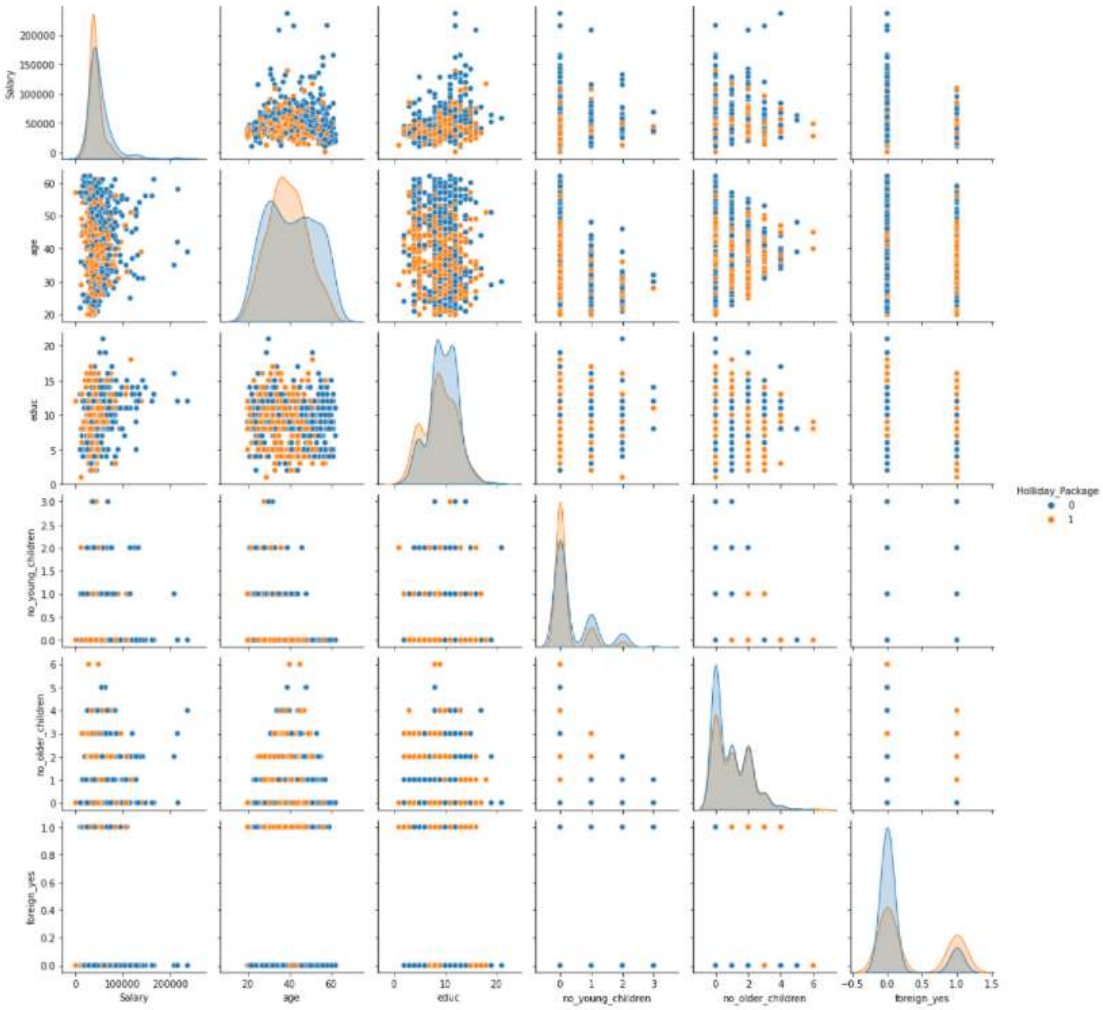


- Plotting histogram to check on data distribution of each variable

```
#checking for skewness
holiday.hist(figsize=(20,15));
```



- Plotting pair plot



- Checking covariance and corelation of data

```
#checking covariance
holiday.cov()
```

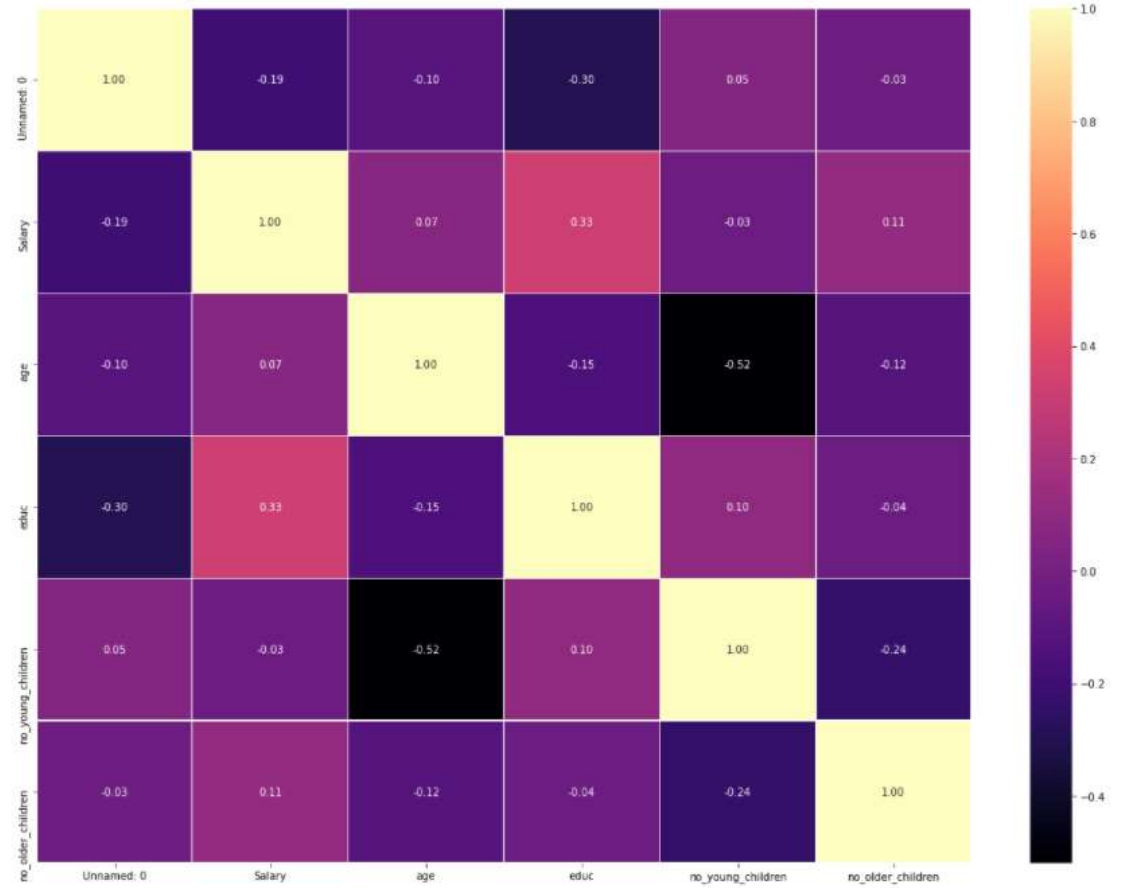
	Unnamed: 0	Salary	age	educ	no_young_children	no_older_children
Unnamed: 0	6.343800e+04	-1.139867e+06	-275.815729	-226.374282	8.049369	-7.076349
Salary	-1.139867e+06	5.484340e+08	17719.779229	23218.662341	-425.752915	2895.613755
age	-2.758157e+02	1.771978e+04	111.337837	-4.783024	-3.356871	-1.332573
educ	-2.263743e+02	2.321866e+04	-4.783024	9.218867	0.183012	-0.119851
no_young_children	8.049369e+00	-4.257529e+02	-3.356871	0.183012	0.375610	-0.158807
no_older_children	-7.076349e+00	2.895614e+03	-1.332573	-0.119851	-0.158807	1.181104

```
#checking correlation
holiday.corr()
```

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

- Creating heat map for corelation

```
#heat map
fig,ax = plt.subplots(figsize=(20, 15))
sns.heatmap(holiday.corr(), ax=ax, annot=True, linewidths=0.05, fmt= '.2f',cmap="magma")
plt.show()
```



- Checking standard deviation of every variable

```
#checking standard deviations of each variable
print(holiday.std())
```

```
Unnamed: 0            251.869014
Salary              23418.668531
age                    10.551675
educ                    3.036259
no_young_children       0.612870
no_older_children       1.086786
dtype: float64
```

- Deleting the unnamed column as its just a sl no and wont play any significant role in further analysis

```
# deleting the unnamed column as its a seriam number column and wont play any part in our further analysis
holiday = holiday.drop('Unnamed: 0', axis=1)
```

- Describing data again to confirm if column has been deleted to not

```
# describing data with all variables included
holiday.describe(include="all")
```

	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
count	872	872.000000	872.000000	872.000000	872.000000	872.000000	872
unique	2	NaN	NaN	NaN	NaN	NaN	2
top	no	NaN	NaN	NaN	NaN	NaN	no
freq	471	NaN	NaN	NaN	NaN	NaN	656
mean	NaN	47729.172018	39.955275	9.307339	0.311927	0.982798	NaN
std	NaN	23418.668531	10.551675	3.036259	0.612870	1.086786	NaN
min	NaN	1322.000000	20.000000	1.000000	0.000000	0.000000	NaN
25%	NaN	35324.000000	32.000000	8.000000	0.000000	0.000000	NaN
50%	NaN	41903.500000	39.000000	9.000000	0.000000	1.000000	NaN
75%	NaN	53469.500000	48.000000	12.000000	0.000000	2.000000	NaN
max	NaN	236961.000000	62.000000	21.000000	3.000000	6.000000	NaN

- Below are the findings from the above EDA performed:-
  - ✓ The data set has 8 variables, 9 independent and 1 dependent variable and below are the details of the variable's:-

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

- Out of these 8 variables 6 are of integer data types and 2 are of string/object data types.
- There are 872 observations. (number of rows).
- There are 0 null values in data.
- Data has NIL duplicate observations.
- Almost all variable has outliers derived from box plot except the "Age" variable.
- None of the variable's are normally distributed.
- Form pair plot and correlation matrix we can conclude that all variables are truly independent in nature and there exists no co-linearity among them.
- The "unnamed" variable can be dropped form the data as it just represents the sl. No and will not play any significant role in further analysis.

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).

Solution:-

- Encoding the dependent variable, Holliday_package using LabelEncoder under sklearn.preprocessing

```
from sklearn.preprocessing import LabelEncoder

## Defining a Label Encoder object instance
LE = LabelEncoder()
```

```
holiday['Holliday_Package'] = LE.fit_transform(holiday['Holliday_Package'])
holiday.head()
```

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

- Using hot encoding for “Foreign” variable

```
# encoding string data type variable to get dummies
holiday = pd.get_dummies(holiday, columns=['foreign'])
```

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

- Splitting data into dependent and independent variable’s

```
# splitting data into independent and dependent variables
X = holiday.drop('Holliday_Package' , axis=1)

y = holiday.pop("Holliday_Package")
```

- Splitting data into training and testing data sets (70:30) and checking the shape of the same:-

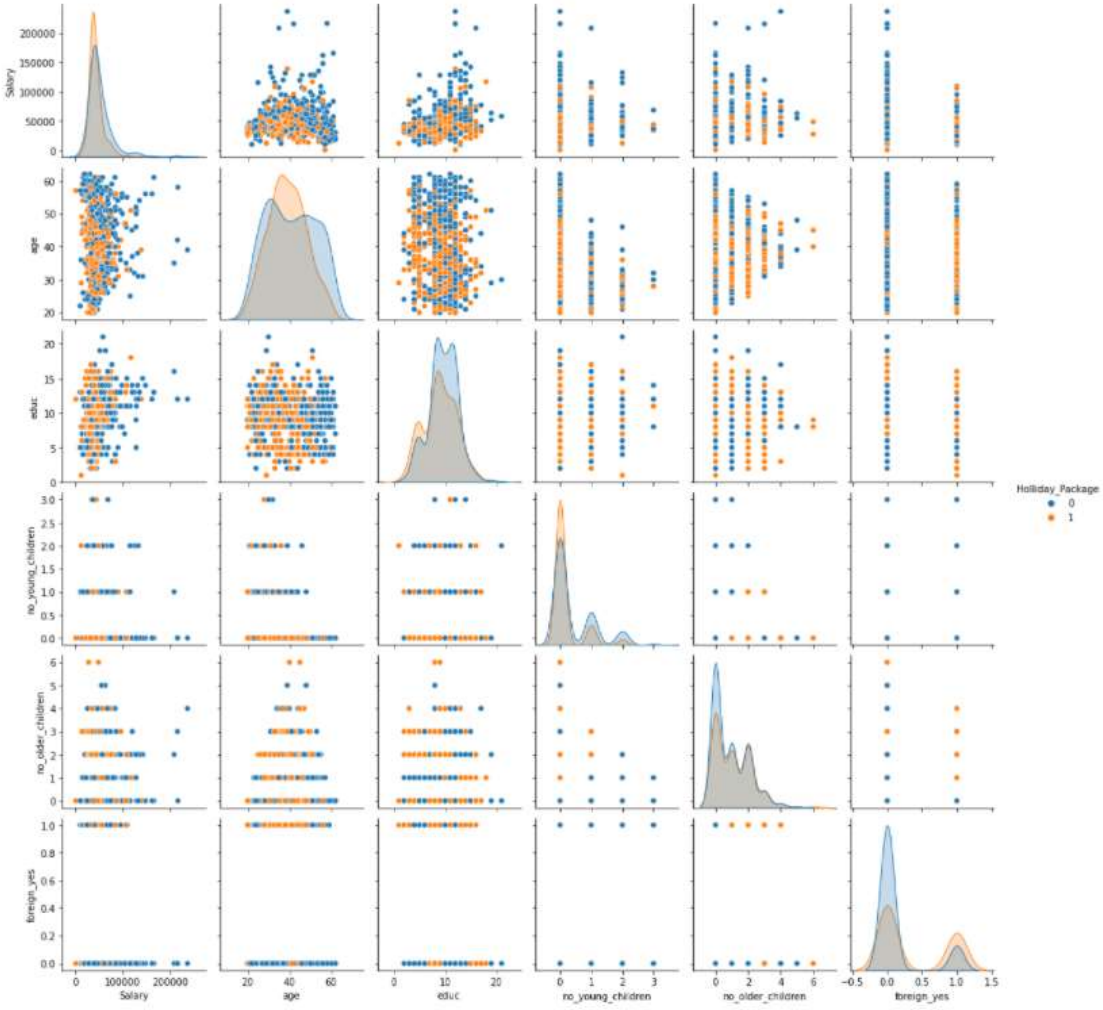
```
#splitting data into train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

```
#checking the dimentsions of training and test data
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
X_train (610, 7)
X_test (262, 7)
y_train (610,)
y_test (262,)
```

LOGISTIC REGRESSION

- Pairplot:-



- Importing logistic regression library and fitting into training data set

```
# Fit the model on original data
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
# Fit the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

- Predicting training and testing data set

```
# predict on train and test data set
ytrain_predict = model.predict(X_train)
ytest_predict = model.predict(X_test)
```

- Getting probability of test data set:-

```
# getting the probability of outcome
ytest_predict_prob=model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

	0	1
0	0.628618	0.371382
1	0.517716	0.482284
2	0.559470	0.440530
3	0.729021	0.270979
4	0.504794	0.495206

Linear Discriminant Analysis (LDA)

- Importing library for linear discriminant analysis

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import scale
```

- Fitting LDA model to train dataset

```
clf = LinearDiscriminantAnalysis()
model=clf.fit(X_train,y_train)
model
```

```
LinearDiscriminantAnalysis()
```

- Predicting train and test dataset

```
# Training Data Class Prediction with a cut-off value of 0.5
pred_class_train = model.predict(X_train)

# Test Data Class Prediction with a cut-off value of 0.5
pred_class_test = 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.

Solution:-

- Accuracy of train data set using logistic regression:-

```
# Accuracy - Training Data
model.score(X_train, y_train)
```

```
0.519672131147541
```

- Accuracy of test dataset using logistic regression

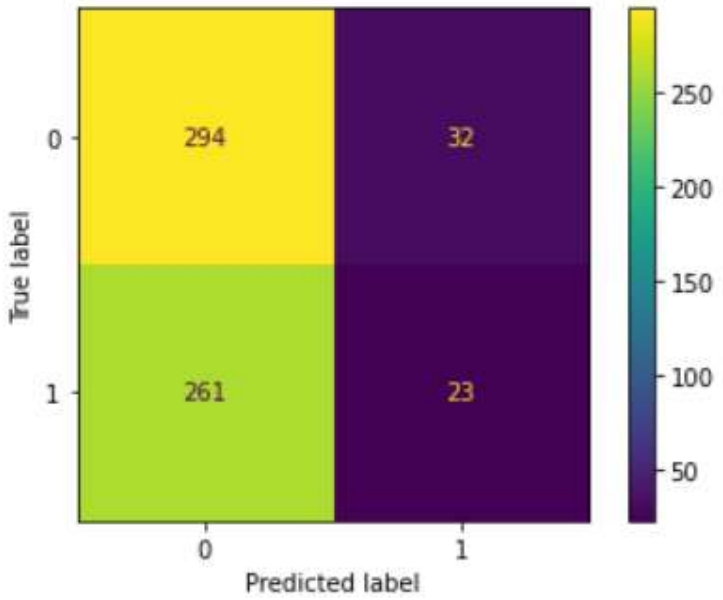
```
# Accuracy - Test Data
model.score(X_test, y_test)
```

```
0.5305343511450382
```

- Confusion matrix of train dataset using logistic regression

```
# confusion matrix for train data set
confusion_matrix(y_train, ytrain_predict)
```

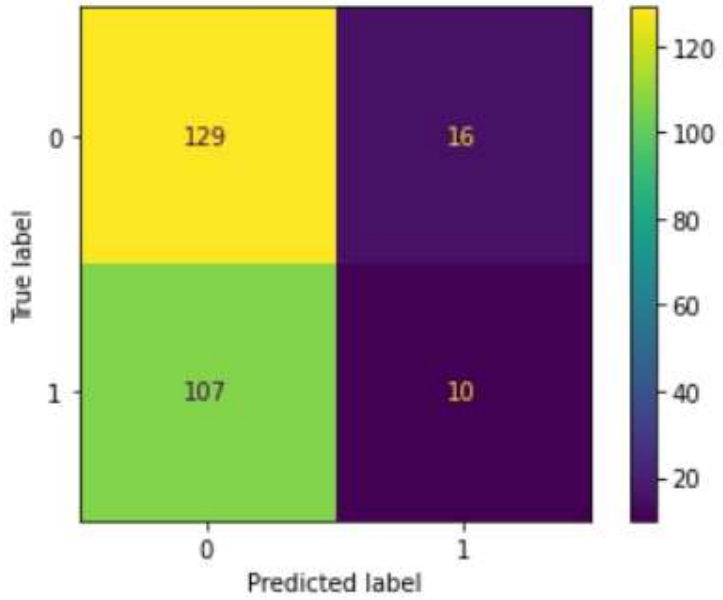
```
array([[294,  32],
       [261,  23]], dtype=int64)
```



- Confusion matrix of test dataset using logistic regression

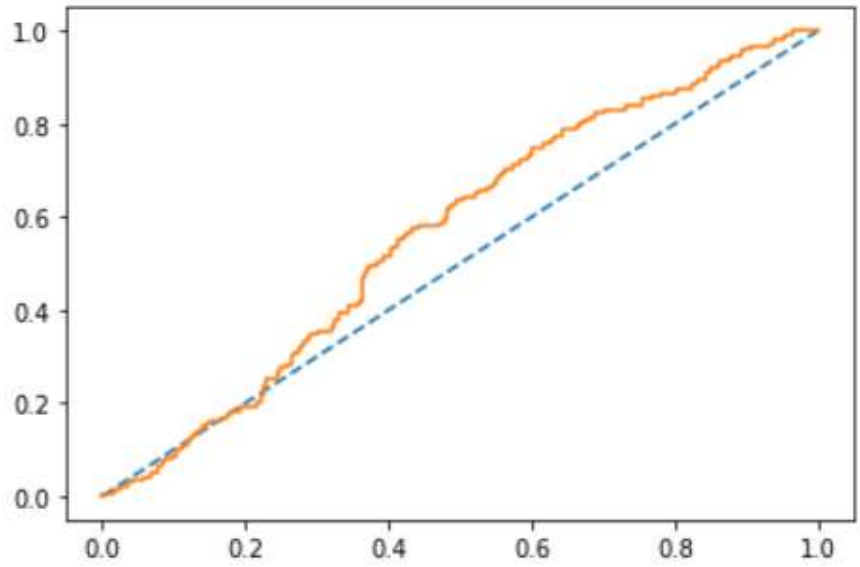
```
# confusion matrix for test data set
confusion_matrix(y_test, ytest_predict)
```

```
array([[129,  16],
       [107,  10]], dtype=int64)
```



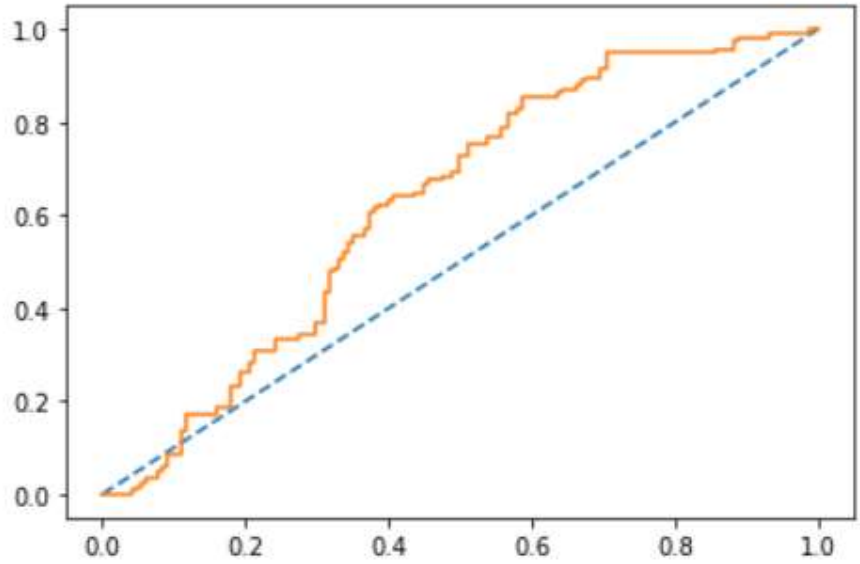
- ROC curve and AUC score for train dataset using logistic regression

```
AUC: 0.567
```



- ROC curve and AUC score for test dataset using logistic regression

```
AUC: 0.567
```



- Classification report of train dataset using logistic regression

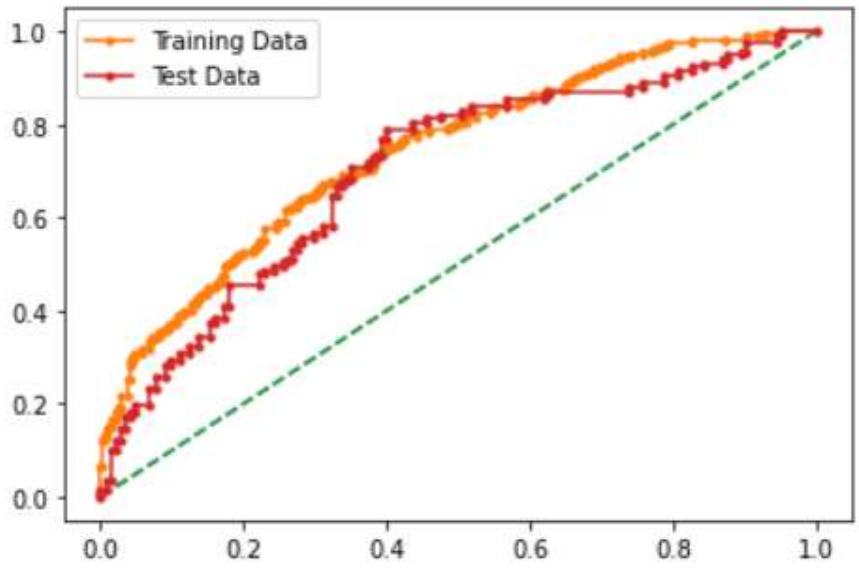
	precision	recall	f1-score	support
0	0.53	0.90	0.67	326
1	0.42	0.08	0.14	284
accuracy			0.52	610
macro avg	0.47	0.49	0.40	610
weighted avg	0.48	0.52	0.42	610

- Classification report for test dataset using logistic regression

	precision	recall	f1-score	support
0	0.55	0.89	0.68	145
1	0.38	0.09	0.14	117
accuracy			0.53	262
macro avg	0.47	0.49	0.41	262
weighted avg	0.47	0.53	0.44	262

- ROC curve and AUC score of train and test dataset using LDA:-

```
AUC for the Training Data: 0.742
AUC for the Test Data: 0.703
```



- Classification report for train and test dataset using LDA:-

```
Classification Report of the training data:

              precision    recall  f1-score   support

           0       0.67      0.77      0.72       326
           1       0.68      0.56      0.61       284

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


Classification Report of the test data:

              precision    recall  f1-score   support

           0       0.66      0.71      0.69       145
           1       0.61      0.56      0.58       117

    accuracy                           0.64       262
   macro avg       0.64      0.63      0.63       262
weighted avg       0.64      0.64      0.64       262
```

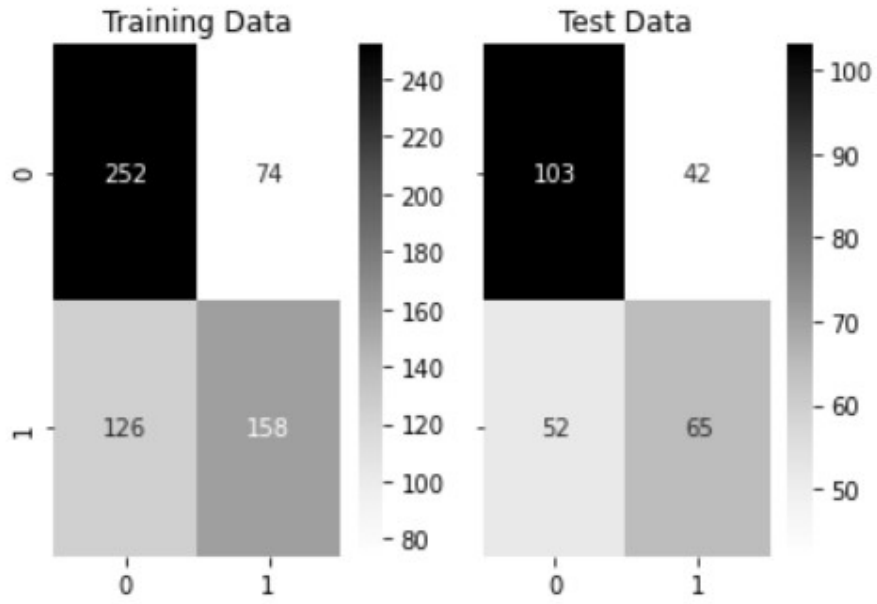
- Confusion matrix of test and train dataset using LDA:-

```
confusion_matrix(y_train, pred_class_train)
```

```
array([[252,  74],
       [126, 158]], dtype=int64)
```

```
confusion_matrix(y_test, pred_class_test)
```

```
array([[103,  42],
       [ 52,  65]], dtype=int64)
```



- Accuracy of test and train dataset using LDA:-

```
#Accuracy - Training dataset
model.score(X_train, y_train)
```

```
0.6721311475409836
```

```
# Accuracy - Test Data
model.score(X_test, y_test)
```

```
0.6412213740458015
```

- Post comparison on the performance of logistic regression and LDA we have found that LDA is more optimized and accuracy/ F1 score along with precision and recall is also better when compared to logistic regression.
- LDA works well when dataset is small, variables are well separated and target variable have two class.
- The AUC score and ROC curve of LDA is better than logistic regression.

2.4 Inference: Basis on these predictions, what are the insights and recommendations. Please explain and summarise the various steps performed in this project. There should be proper business interpretation and actionable insights present.

Solution:-

INSIGHTS:-

- ✓ Data has 872 observations and 8 variables.
- ✓ no null values and NIL duplicates.
- ✓ Except salary all other are categorical columns.
- ✓ None of the variables are co-related. We hardly see any correlation.

- ✓ From pair plot we can see that the kernel density across diagonal in pair plot is overlapping and eclipsing each other for dependent variable.
- ✓ Such attributes/variables are unable to discriminate between the two classes, because the probability of determining both the class is 50%.
- ✓ Such attributes are either weak or poor predictors.
- ✓ For the variables "no_young_childer", "no_older_childern" and "foreign" are completely overlapping each other and these are poor predictors in the dataset given. These variables cannot discriminate between the two classes of target variable.
- ✓ Other variable's "Salary", "age" and "education" there is a major overlap however there is also density difference between the two classes. These are weak predictors for the target variable as they can discriminate between the classes but not completely.

RECOMMENDATIONS:-

- ✓ The business should segregate customers into 3 segments based on their profile considering their salary, education and age into like High, Medium and Low.
- ✓ Every customer segment can be offered in a different way.
- ✓ Customized products/plans can be designed for each customer segment.
- ✓ Foreigner's and non-foreigner's can be treated separately based on their origin and life style, this will help business gaining customers trust and confidence.
- ✓ Business can roll out/design different family floater plans for small families and different for big/large families.
- ✓ Different customer desks can be setup for foreigner and non-foreigner.
- ✓ Discounts can be given to a family with high number of younger kids.
- ✓ Other offers can be rolled out for families with adult kids.
- ✓ The holiday package should have everything for everyone, considering their profile, age and families.
- ✓ Business should be ready for different seasons and regular mailer and promotional offers should be communicated to the customers so that they won't miss on anything worthy and exciting.

VARIOUS STEPS PERFORMED IN THIS PROJECT

- While performing EDA we have below conclusions:
  - ✓ None of the variable are co-related.
  - ✓ Need to convert the variable's "Holliday_package" and "Foreign" into int data type as they are string data types.
  - ✓ From pair plot found that all the variables are either weak or poor predictors for the classes. As they overlap each other at very high extent and won't be able to discriminate between two classes of the dependent variable.
  - ✓ Post EDA we have checked if data is balanced or not by checking in the variable count for each class of dependent variable, and found that data is almost balanced.
  - ✓ Using LabelEncoder library under sklearn.preprocessing we have labelled the two classes of the dependent variable as "0" and "1". Hence, the variable "Holliday_package" is converted into int data type.
  - ✓ Using getDummies function under pandas to encode "foreign" variable and hence this variable also converted to int data type.

- ✓ Created two data sets by splitting data into dependent and independent variables.
- ✓ Divided data into training and testing data sets in 70:30 ratio and also checked the shape of each dataset.
- ✓ From sklearn.linear_model we have imported LogisticRegression and fitted the model into training datasets.
- ✓ We predicted for training and test dataset.
- ✓ Measured accuracy of model for training and test dataset.
- ✓ From sklearn.metrics imported required libraries for generating confusion matrix, plotting confusion matrix, getting AUC score, getting ROC curve and also classification report for training and testing datasets.
- ✓ Got accuracy, confusion matrix, plot of confusion matrix, classification report, AUC score and ROC curve for both training and testing datasets.
- ✓ In the similar way we have imported LinearDiscriminantAnalysis library from sklearn.discriminant_analysis.
- ✓ Fitted model into train datasets.
- ✓ Predicted for train and test data.
- ✓ Calculated accuracy, AUC score for both training and testing dataset.
- ✓ Plotted ROC curve for training and testing data sets.
- ✓ Generated confusion matric and classification report for training and testing data sets.
- ✓ With all the above analysis and model building we can conclude that LDA is more optimized for this case study by comparing its accuracy, F1 score, precision and recall scores with logistic regression.
- ✓ However, the variable in this case study are weak or poor predictors of the classes of dependent variables.

----------END OF REPORT----------