Project - Predictive Modeling

11.04.2021

—

Peeyush Vardhan
Phone: +91-9515456727
Email: peeyushvrdhn@gmail.com

Problem 1: Linear Regression - Gem Stones co ltd Case Study

Problem Statement

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

Dataset for Problem 1: cubic_zirconia.csv

Data Dictionary:

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.

Problem 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:

Initial Information and shape of the data:

Top 10 values:

```
1 df.head(10)
```

	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
8	9	1.21	Good	H	SI1	63.8	64.0	6.72	6.63	4.26	5407
9	10	0.35	Ideal	F	VS2	60.5	57.0	4.52	4.60	2.76	706

Bottom 10 values:

```
1 df.tail(10)
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
26957	26958	2.09	Premium	H	SI2	60.6	59.0	8.27	8.22	5.00	17805
26958	26959	1.37	Premium	E	SI2	61.0	57.0	7.25	7.19	4.40	6751
26959	26960	1.05	Very Good	E	SI2	63.2	59.0	6.43	6.36	4.04	4281
26960	26961	1.10	Very Good	D	SI2	NaN	63.0	6.76	6.69	3.94	4361
26961	26962	0.25	Premium	F	VVS2	62.0	59.0	4.04	3.99	2.49	740
26962	26963	1.11	Premium	G	SI1	62.3	58.0	6.61	6.52	4.09	5408
26963	26964	0.33	Ideal	H	IF	61.9	55.0	4.44	4.42	2.74	1114
26964	26965	0.51	Premium	E	VS2	61.7	58.0	5.12	5.15	3.17	1656
26965	26966	0.27	Very Good	F	VVS2	61.8	56.0	4.19	4.20	2.60	682
26966	26967	1.25	Premium	J	SI1	62.0	58.0	6.90	6.88	4.27	5166

```
1 df.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
```

```
1 df.shape
```

```
(26967, 11)
```

Insights:

- The present variables are both numeric and categorical types in nature - i.e. float, int, and object data types are present
- There are 11 variables and 26967 records

Basic statistical details:

```
#Calculating basic statistical data
df.describe().T
```

	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

```
#Calculating basic statistical data including all information\
df.describe(include = "all").T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Unnamed: 0	26967	NaN	NaN	NaN	13484	7784.85	1	6742.5	13484	20225.5	26967
carat	26967	NaN	NaN	NaN	0.798375	0.477745	0.2	0.4	0.7	1.05	4.5
cut	26967	5	Ideal	10816	NaN	NaN	NaN	NaN	NaN	NaN	NaN
color	26967	7	G	5661	NaN	NaN	NaN	NaN	NaN	NaN	NaN
clarity	26967	8	SI1	6571	NaN	NaN	NaN	NaN	NaN	NaN	NaN
depth	26270	NaN	NaN	NaN	61.7451	1.41286	50.8	61	61.8	62.5	73.6
table	26967	NaN	NaN	NaN	57.4561	2.23207	49	56	57	59	79
x	26967	NaN	NaN	NaN	5.72985	1.12852	0	4.71	5.69	6.55	10.23
y	26967	NaN	NaN	NaN	5.73357	1.16606	0	4.71	5.71	6.54	58.9
z	26967	NaN	NaN	NaN	3.53806	0.720624	0	2.9	3.52	4.04	31.8
price	26967	NaN	NaN	NaN	3939.52	4024.86	326	945	2375	5360	18818

Insights:

- The target variable will be: price
- Looking at the statistical analysis it is seen that:
  - Categorical data - cut color and clarity
  - Continuous data - carat, depth, table, x, y, z, and price

Checking for duplicate records:

```
#Checking Null Value
df.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
```

```
#Checking duplicates in data
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
```

```
Number of duplicate rows = 0
```

- There are no duplicate rows.

Unique values in categorical variables

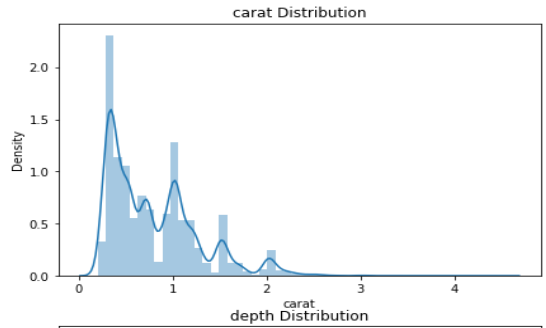
```
CUT :  5
Fair            781
Good           2441
Very Good      6030
Premium        6899
Ideal         10816
Name: cut, dtype: int64
```

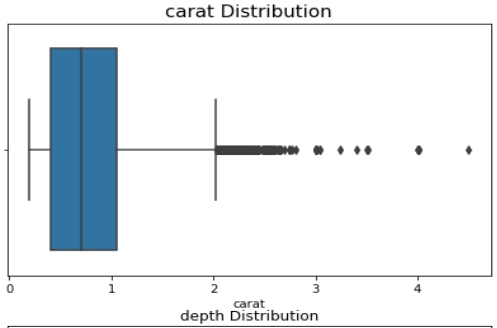
- **Total CUT is 5.**
- **Preferred CUT would be Ideal**

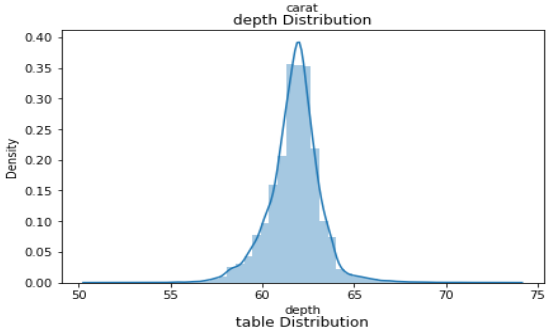
```
COLOR :  7
J    1443
I    2771
D    3344
H    4102
F    4729
E    4917
G    5661
Name: color, dtype: int64
```

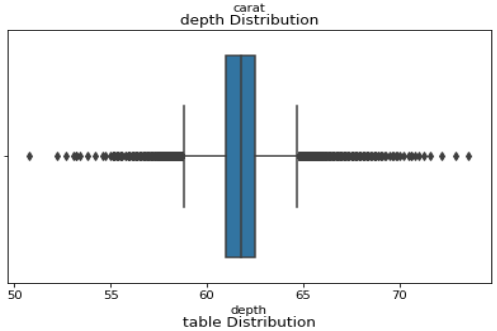
```
CLARITY :  8
I1        365
IF        894
VVS1     1839
VVS2     2531
VS1      4093
SI2      4575
VS2      6099
SI1      6571
Name: clarity, dtype: int64
```

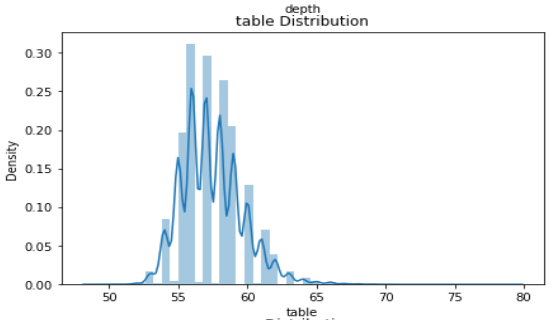
Univariate and bivariate analysis:

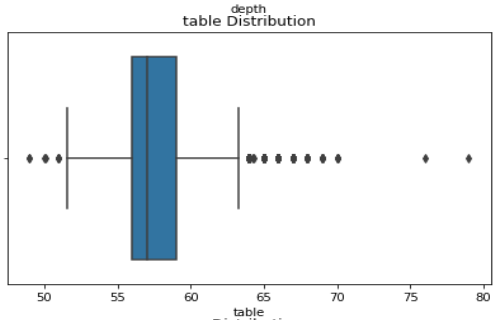


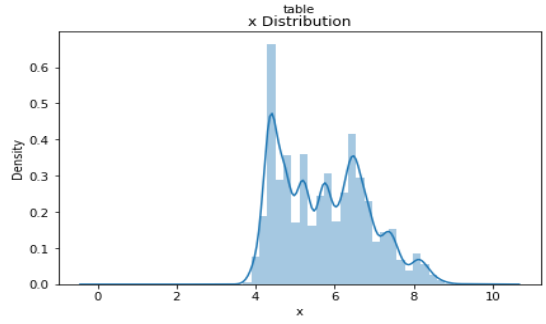


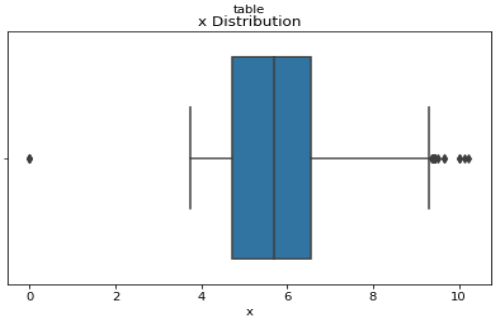


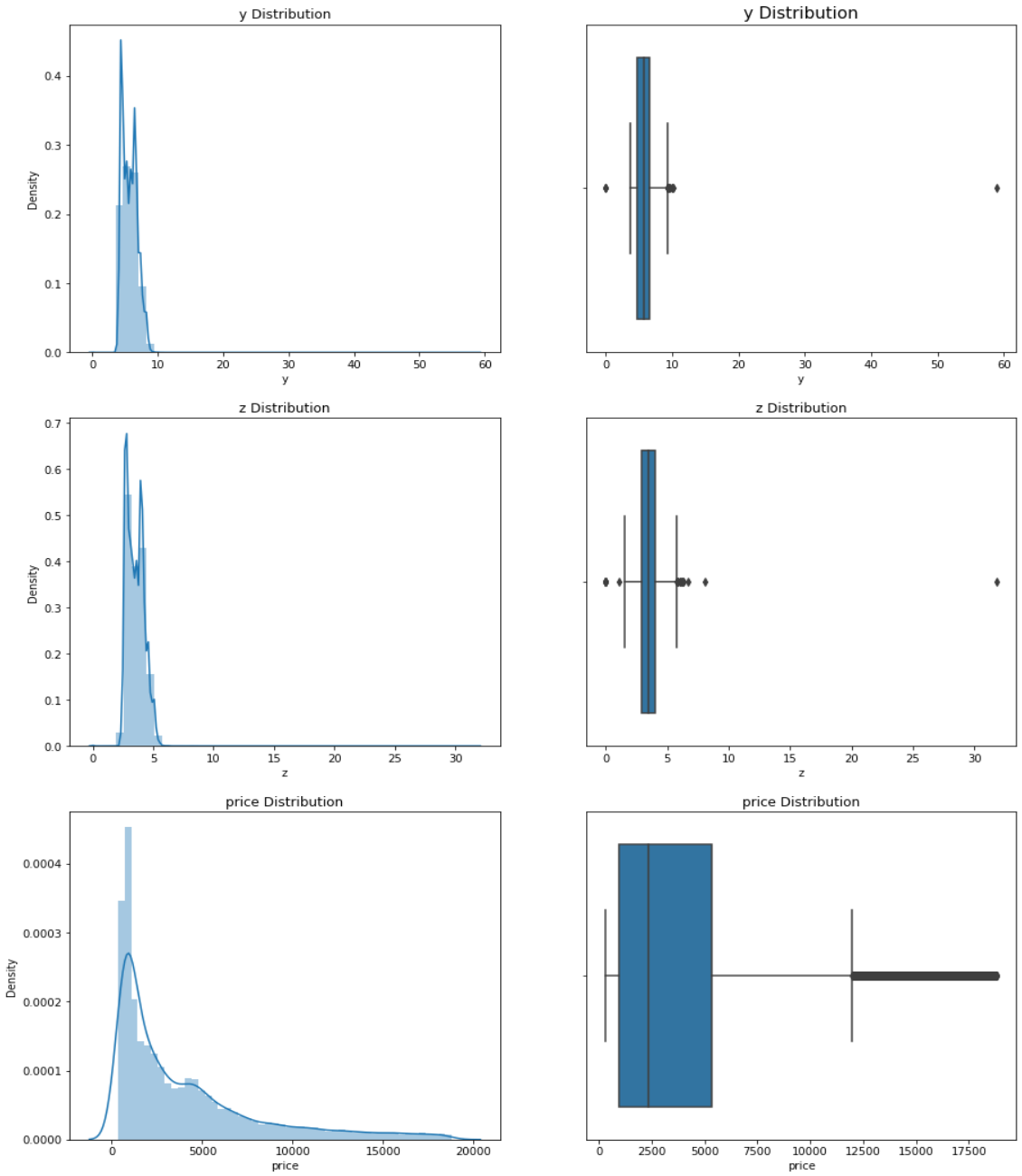












Insights:

- The distribution of data in carat has positive skewness. The data range is 0 to 1, a major portion of data lies in this range.
- The distribution of depth is in the normal distribution and it ranges from 55 to 65.
- The distribution of data in the table has positive skewness. The maximum distribution range is between 55 to 65.
- The distribution of x has positive skewness. The distribution range is 4 to 8. X represents the length of the cubic zirconia in mm.

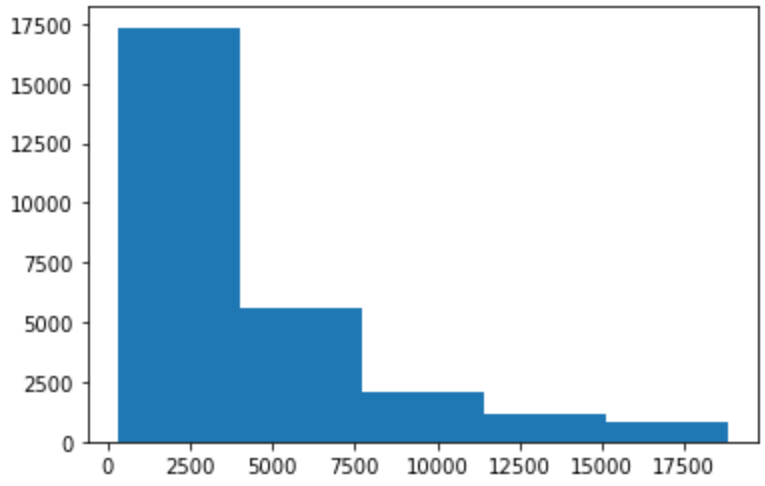
- The distribution of Y is excessively positively skewed. Y represents the width of the cubic zirconia in mm.)
- The distribution of z is asymmetrically positively skewed. Z represents the height of the cubic zirconia in mm
- Price has positive skewness. The distribution range is between RS 100 to 8000

Measurement of skewness:

```
#Measuring the skewness
df.skew()
```

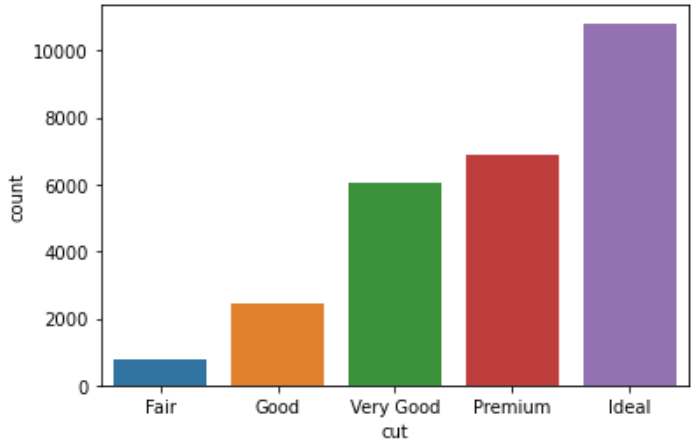
```
Unnamed: 0    0.000000
carat         1.116481
depth        -0.028618
table         0.765758
x             0.387986
y             3.850189
z             2.568257
price         1.618550
dtype: float64
```

Distribution of Price using Histogram:

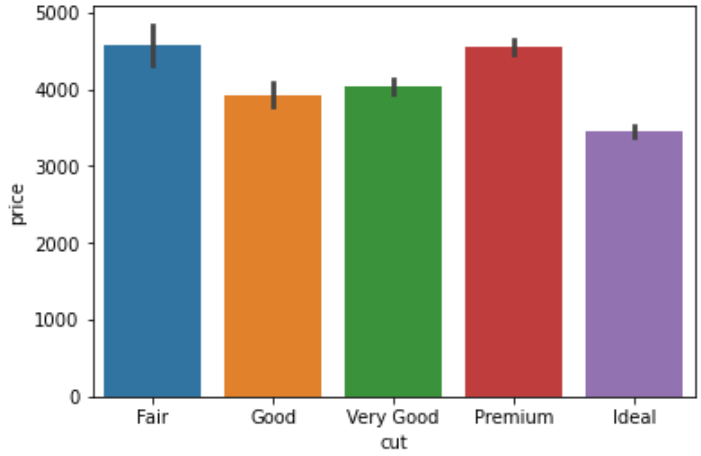


Graphical comparison of categorical variables:

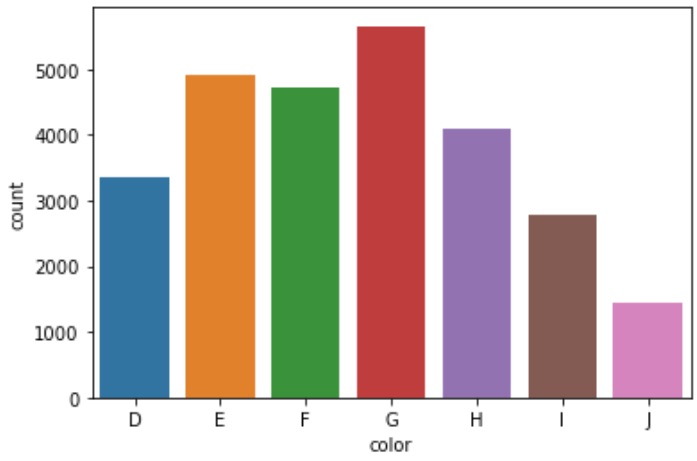
Representation of CUT, in the order of Fair, Good, Very Good, Premium, Ideal



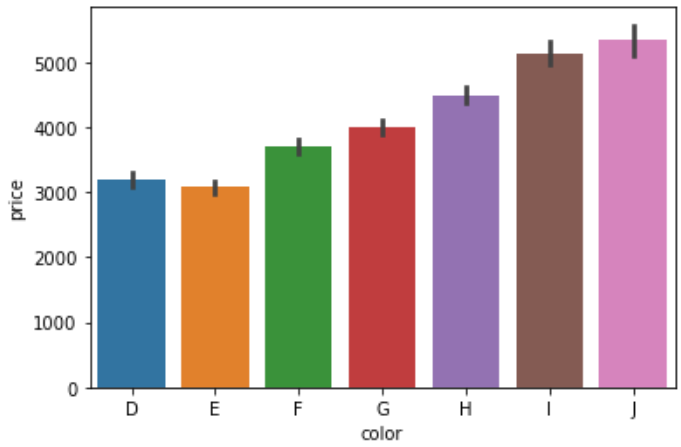
Representation of CUT with Price, in the order of Fair, Good, Very Good, Premium, Ideal



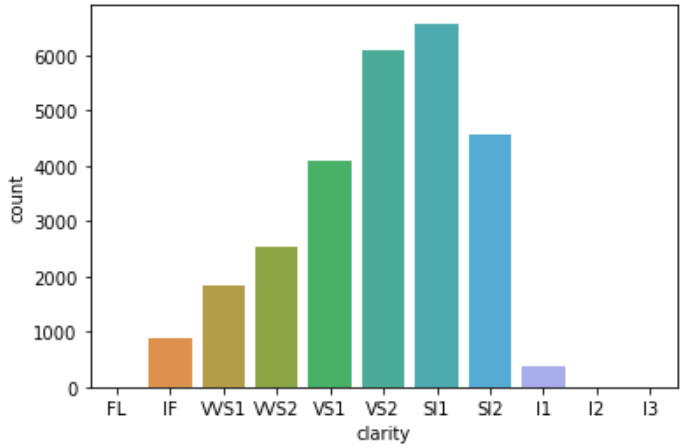
Representation of color, in the order of D, E, F, G, H, I, J



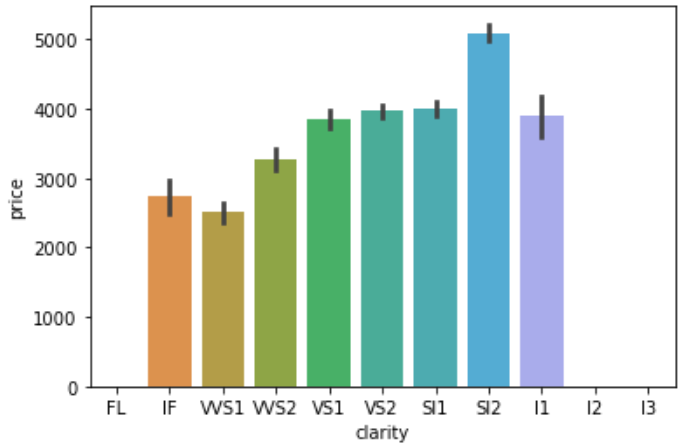
Representation of color with Price, in the order of D, E, F, G, H, I, J



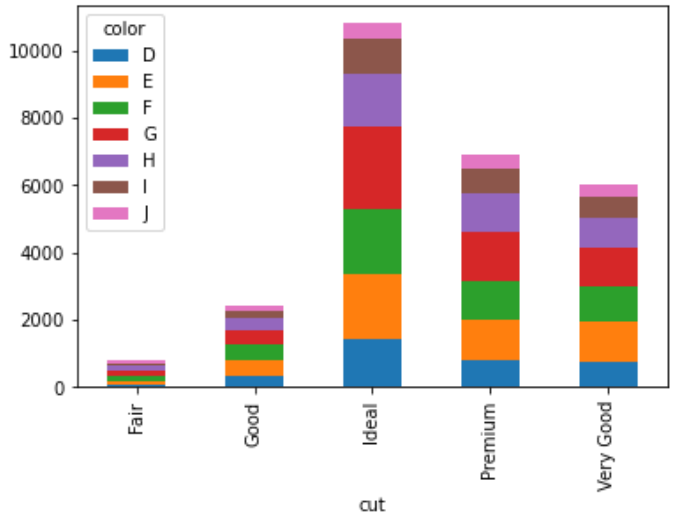
Representation of clarity, in the order of FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3



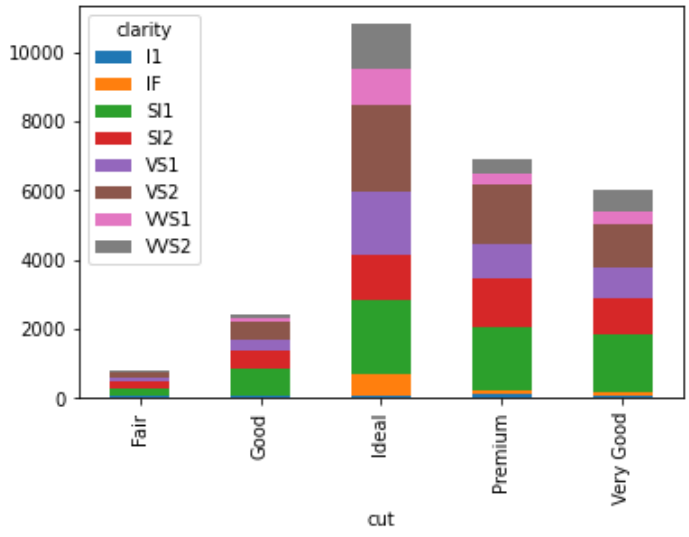
Representation of clarity with Price, in the order of FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3



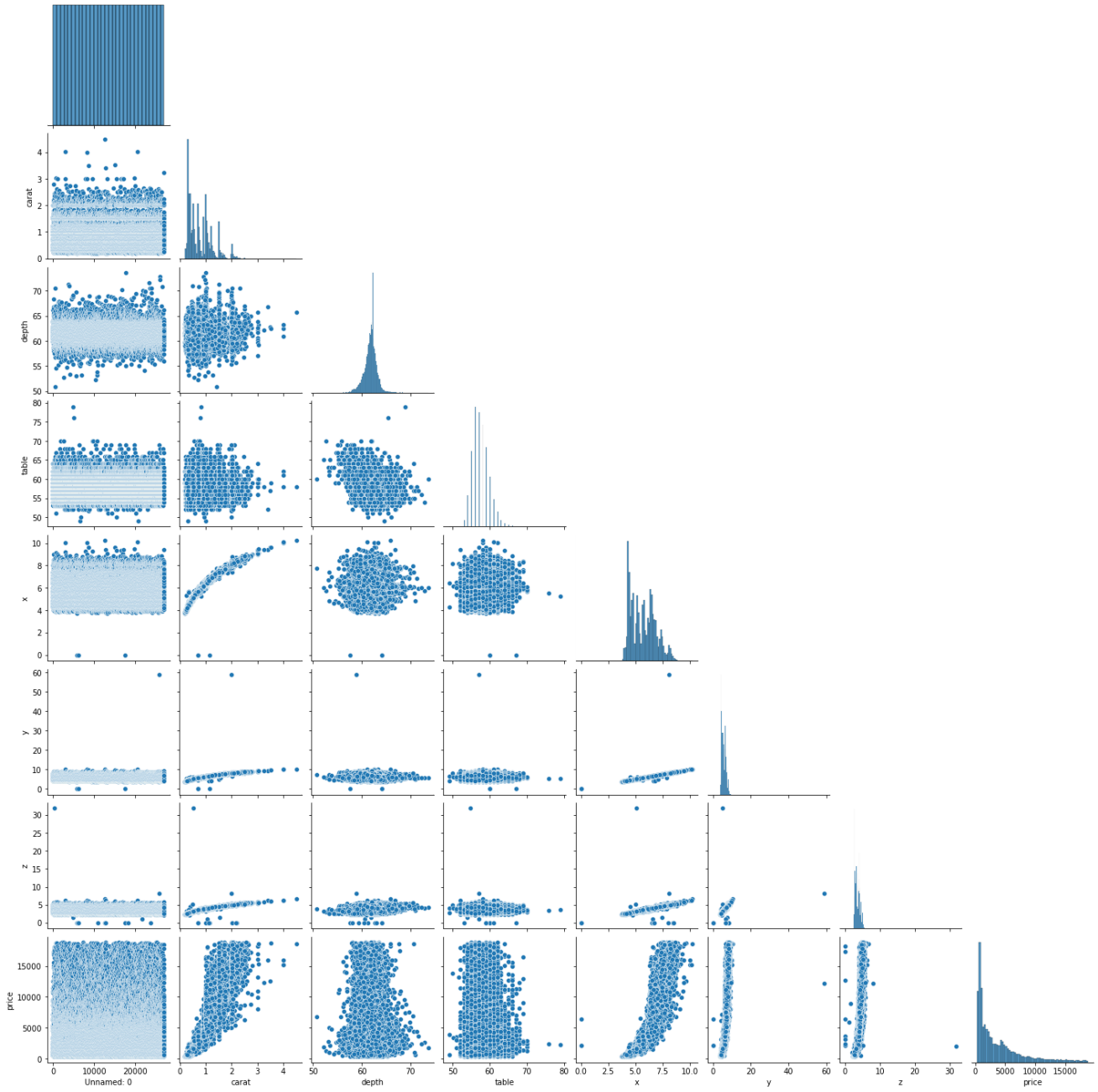
Representation of the comparison between CUT and color



Representation of the comparison between CUT and clarity



Multivariate Analysis to display data distribution



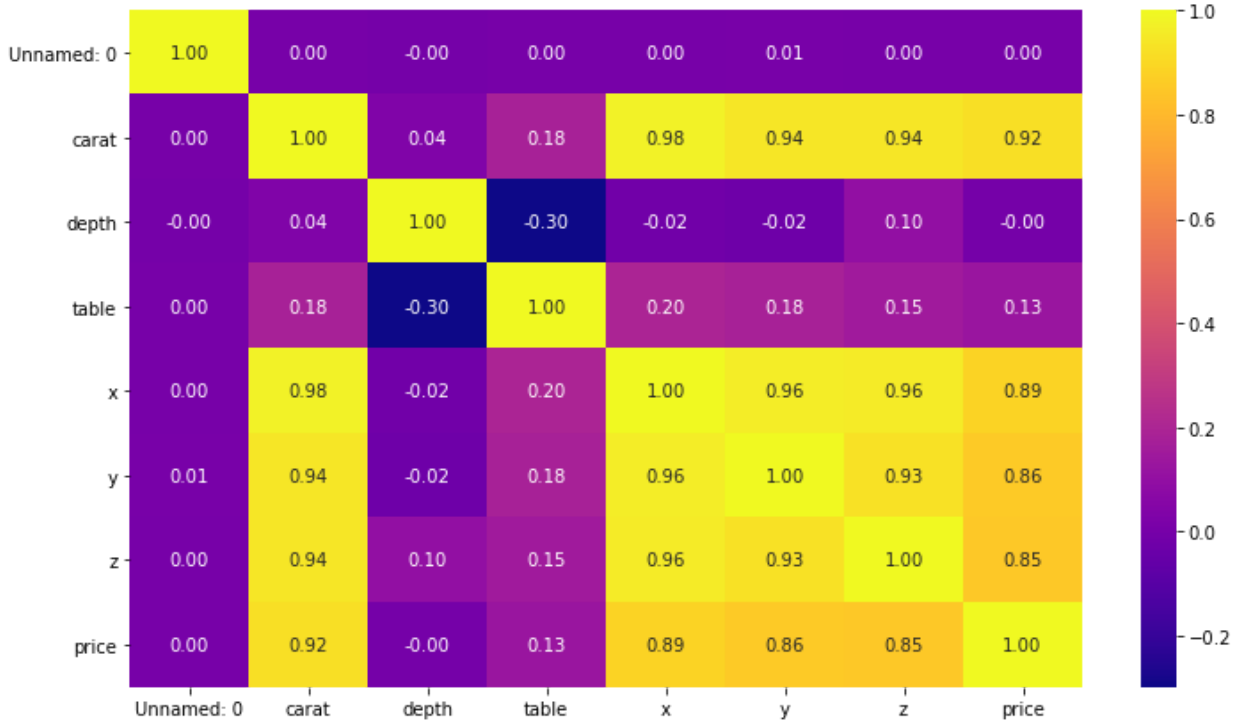
Correlation Matrix:

	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

Insight:

- Multicollinearity is present in the dataset.

Correlation Matrix - Heatmap:



Problem 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:

```
df.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
```

- We can see that there are 697 null values found in-depth
- To solve this we will do the median imputation as depth is a continuous variable. Please refer to the Jupyter notebook to go through the codes

After median imputation:

```
for column in df.columns:
    if df[column].dtype != 'object':
        median = df[column].median()
        df[column] = df[column].fillna(median)

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

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

Answering: Do you think scaling is necessary in this case?

_____Yes, my suggestion would be to scale the data due to observed high multicollinearity. This might cause the following issues in the linear regression model:

- The coefficient calculations can fluctuate wildly. Also, the coefficients turn extremely sensitive to small changes in the model.
- Multicollinearity diminishes the precision of the estimated coefficients, which weakens the statistical measures of the regression model.

Hence, to ignore these issues, scaling is necessary for the current dataset. Please refer to the Jupyter notebook to go through the codes for scaling.

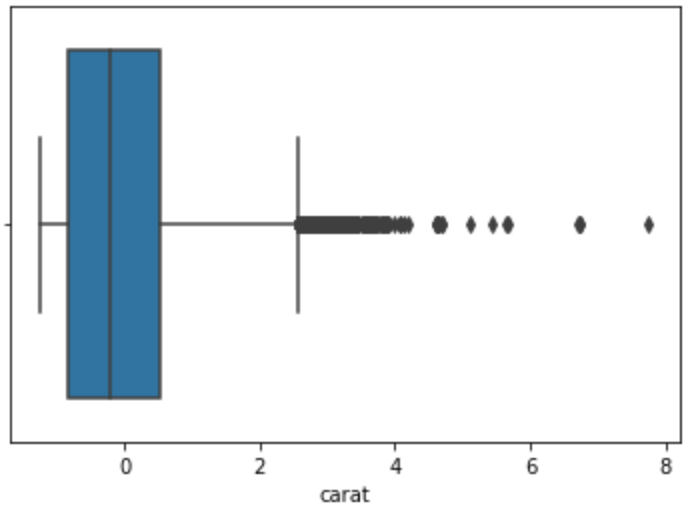
Here's the scaled data:

```
1 df.head()
```

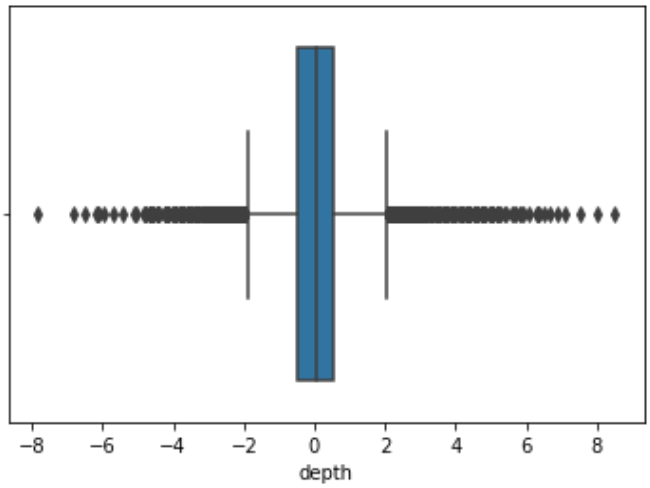
	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	-1.731904	-1.043125	Ideal	E	SI1	0.253399	0.244112	-1.295920	-1.240065	-1.224865	-0.854851
1	-1.731776	-0.980310	Premium	G	IF	-0.679158	0.244112	-1.162787	-1.094057	-1.169142	-0.734303
2	-1.731647	0.213173	Very Good	E	VVS2	0.325134	1.140496	0.275049	0.331668	0.335404	0.584271
3	-1.731519	-0.791865	Ideal	F	VS1	-0.105277	-0.652273	-0.807766	-0.802041	-0.806936	-0.709945
4	-1.731390	-1.022187	Ideal	F	VVS1	-0.966099	0.692304	-1.224916	-1.119823	-1.238796	-0.785257

Finding Outlier and Outlier Treatment:

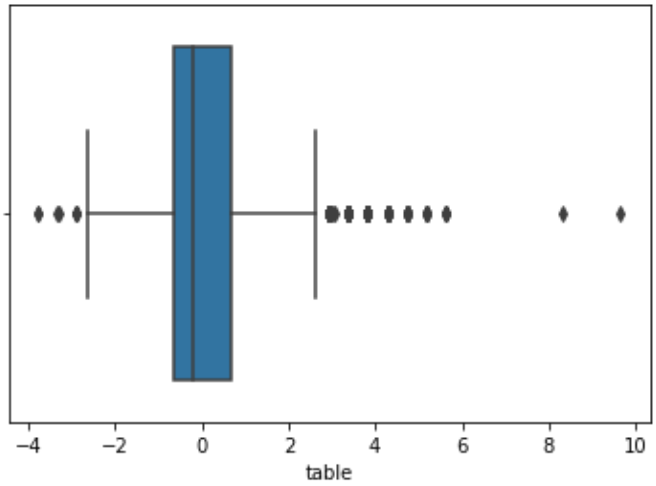
Carat



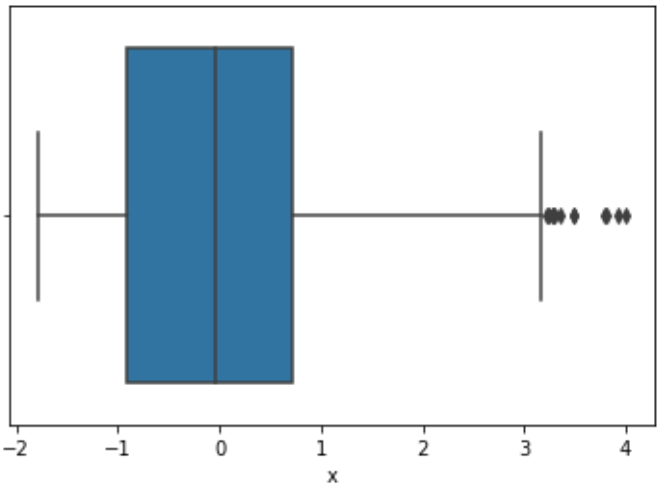
Depth



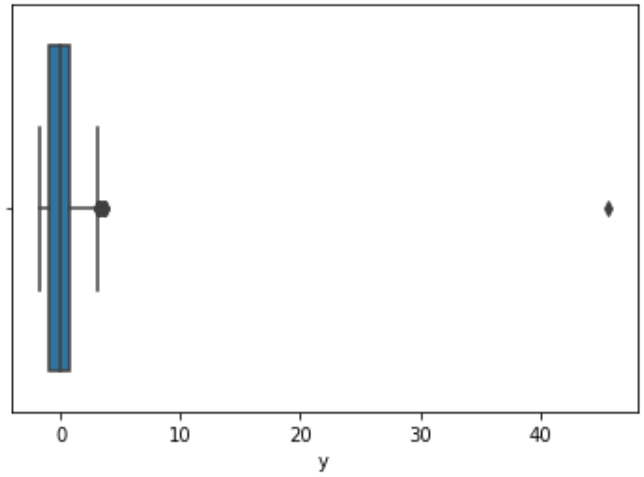
Table



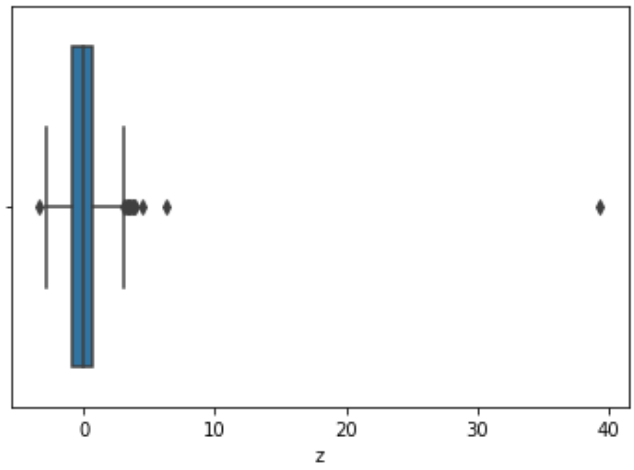
X



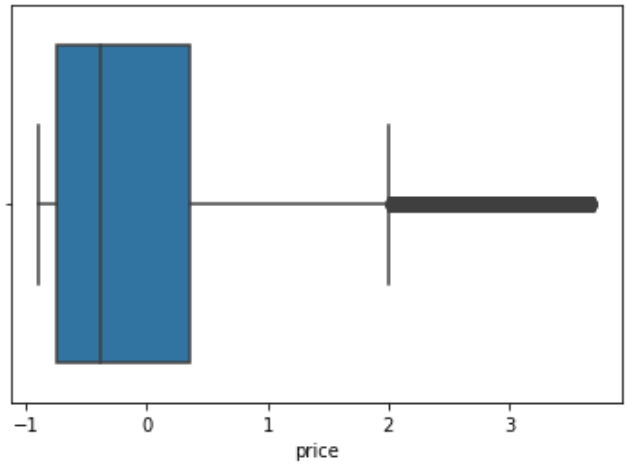
Y



Z



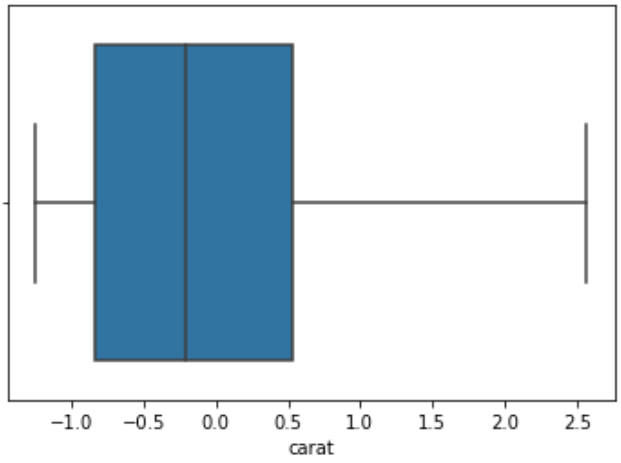
Price



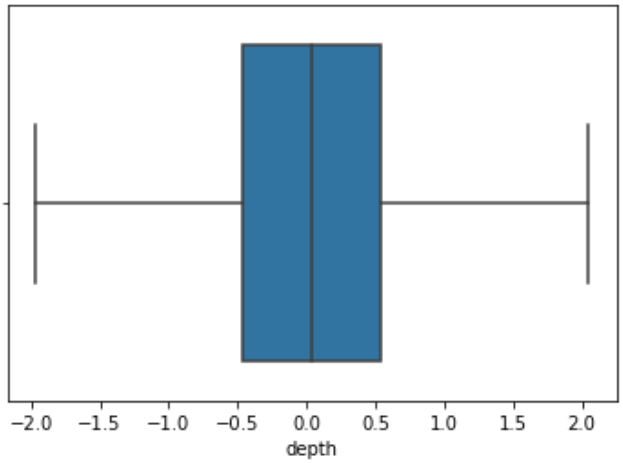
Outlier Treatment:

Please check the Jupyter notebook attached to view the codes of outlier treatment: Post outlier treatment, graphs are given below:

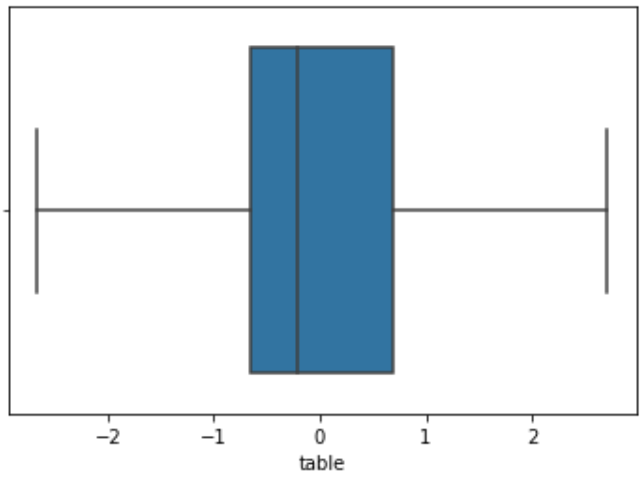
Carat



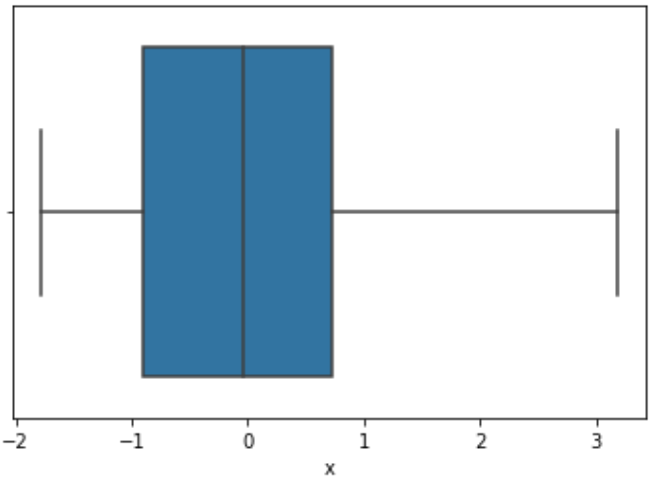
Depth



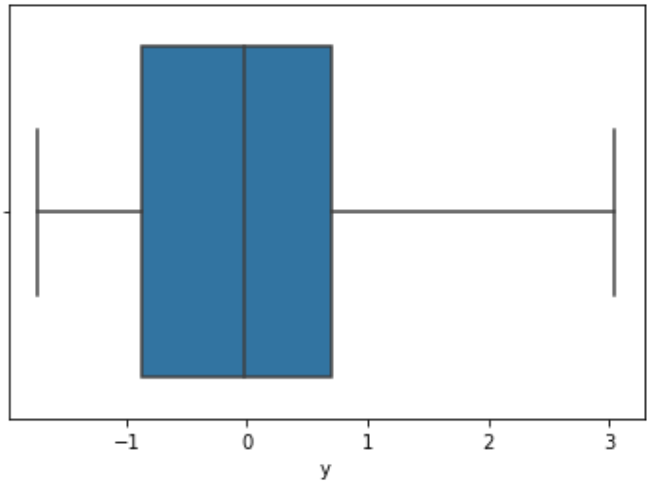
Table



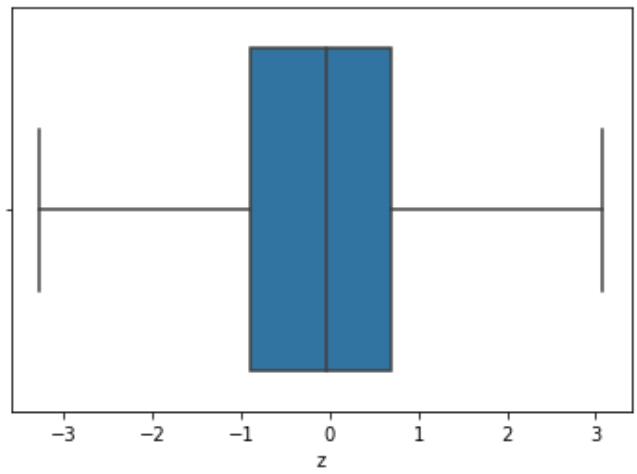
X



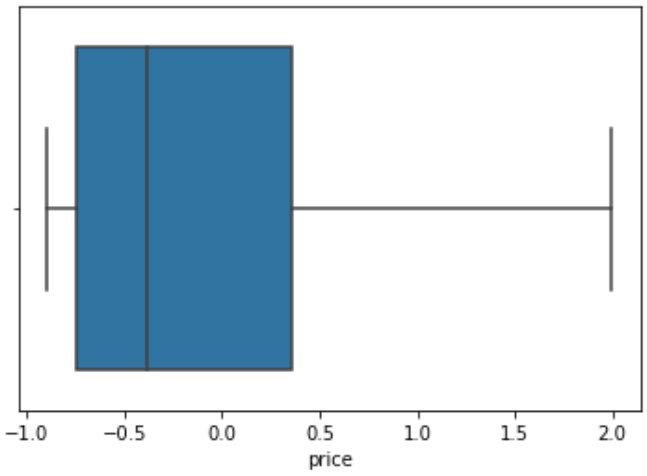
Y



Z



Price



Problem 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 R-square, RMSE.

Solution:

Step 1: Convert categorical to dummy variables in data

Categorical values cannot be given in the linear regression model. Hence, we will encode categorical values to an integer by converting categorical to dummy variables. It is a way to make the categorical variable into a series of dichotomous variables.

Please refer to the Jupyter notebook to go through the codes.

```
data = pd.get_dummies(df, columns=['cut','color','clarity'],drop_first=True)
data.head(10)
```

	Unnamed: 0	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	...	color_H	color_I	color_J	clarity_IF	clarity_
0	-1.731904	-1.043125	0.253399	0.244112	-1.295920	-1.240065	-1.224865	-0.854851	0	1	...	0	0	0	0	
1	-1.731776	-0.980310	-0.679158	0.244112	-1.162787	-1.094057	-1.169142	-0.734303	0	0	...	0	0	0	1	
2	-1.731647	0.213173	0.325134	1.140496	0.275049	0.331668	0.335404	0.584271	0	0	...	0	0	0	0	
3	-1.731519	-0.791865	-0.105277	-0.652273	-0.807766	-0.802041	-0.806936	-0.709945	0	1	...	0	0	0	0	
4	-1.731390	-1.022187	-0.966099	0.692304	-1.224916	-1.119823	-1.238796	-0.785257	0	1	...	0	0	0	0	
5	-1.731262	0.464433	-0.177012	-0.652273	0.647821	0.649450	0.627955	1.382872	0	1	...	0	0	0	0	
6	-1.731133	0.443495	1.401161	1.140496	0.550190	0.486265	0.683679	0.223123	1	0	...	1	0	0	0	
7	-1.731005	-0.624359	-0.177012	2.036880	-0.568127	-0.578734	-0.584040	-0.627177	0	0	...	0	0	0	0	
8	-1.730877	0.862261	1.472896	2.709169	0.878584	0.769692	1.004092	0.365047	1	0	...	1	0	0	0	
9	-1.730748	-0.938433	-0.894364	-0.204081	-1.074032	-0.973815	-1.085556	-0.803401	0	1	...	0	0	0	0	

10 rows × 25 columns

Step 2: Data Split: Split the data into train and test (70:30)

We will drop the id column.

```
# dropping the id column
data_model = data.drop(columns=['Unnamed: 0'], axis=1)
```

```
data_model.columns
```

```
Index(['carat', 'depth', 'table', 'x', 'y', 'z', 'price', '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'],
      dtype='object')
```

```
#Train and test split of X and y in 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3 , random_state=1)
```

Step 3: Applying the Linear Regression Model

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

```
LinearRegression()
```

```
# As further deep dive, explore 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_[0][idx]))
```

```
The coefficient for carat is 1.1009417847804512
The coefficient for depth is 0.0056051434455706645
The coefficient for table is -0.013319500386804255
The coefficient for x is -0.3050434981963348
The coefficient for y is 0.30391448957926764
The coefficient for z is -0.13916571567988092
The coefficient for cut_Good is 0.09403402912977939
The coefficient for cut_Ideal is 0.1523107462056743
The coefficient for cut_Premium is 0.14852774839849367
The coefficient for cut_Very Good is 0.125838818784527
The coefficient for color_E is -0.04705442233369834
The coefficient for color_F is -0.06268437439142852
The coefficient for color_G is -0.10072161838356827
The coefficient for color_H is -0.207673133116617
The coefficient for color_I is -0.3239541927462753
The coefficient for color_J is -0.4685893027501587
The coefficient for clarity_IF is 0.9997691394634917
The coefficient for clarity_SI1 is 0.638978581827134
The coefficient for clarity_SI2 is 0.4295966234831559
The coefficient for clarity_VS1 is 0.8380875826737574
The coefficient for clarity_VS2 is 0.7660244466083616
The coefficient for clarity_VVS1 is 0.9420769630114076
The coefficient for clarity_VVS2 is 0.9313670288415697
```

Checking the intercept for the model

```
# Checking the intercept for the model
intercept = regression_model.intercept_[0]
print("The intercept for the model is {}".format(intercept))
```

```
The intercept for the model is -0.756762786304939
```

R square for training and test data

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

```
0.9419557931252712
```

```
# R square for test data
regression_model.score(X_test, y_test)
```

```
0.9381643998102491
```

Root Mean Square Error (RMSE) for training and test data

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

```
0.20690072466418796
```

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

```
0.21647817772382874
```

Variance inflation factor (VIF) and statsmodel

Please refer to the Jupiter notebook to go through the codes.

```
carat --> 33.35086119845924
depth --> 4.573918951598579
table --> 1.7728852812618994
x --> 463.5542785436457
y --> 462.769821646584
z --> 238.65819968687333
cut_Good --> 3.609618194943713
cut_Ideal --> 14.34812508118844
cut_Premium --> 8.623414379121153
cut_Very Good --> 7.848451571723688
color_E --> 2.371070464762613
```

Statsmodels

```
Intercept      -0.662077
carat           1.102593
depth           0.004668
table          -0.016875
x              -0.373553
y               0.403978
z              -0.170463
cut_Good       -0.015859
cut_Ideal       0.035978
cut_Premium     0.038824
color_E        -0.046800
color_F        -0.063456
color_G        -0.100920
color_H        -0.207857
color_I        -0.324126
color_J        -0.469230
clarity_IF      1.017422
clarity_SI1     0.658350
clarity_SI2     0.447550
clarity_VS1     0.856709
clarity_VS2     0.785028
clarity_VVS1    0.961351
clarity_VVS2    0.950398
dtype: float64
```

Insights:

- Multicollinearity is still present in the dataset. The exemplary interpretation of VIF is less than 5%.
- The obtained stats model displays that its features do not add value to the model, hence those features can be removed and that'll ultimately lead to reducing the VIF value. This will build a better linear regression model.

Inferential statistical data

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.941
Model:                            OLS   Adj. R-squared:                  0.941
Method:                 Least Squares   F-statistic:                 1.379e+04
Date:                Sun, 11 Apr 2021   Prob (F-statistic):               0.00
Time:                        11:13:36   Log-Likelihood:                 2879.6
No. Observations:               18870   AIC:                            -5713.
Df Residuals:                   18847   BIC:                            -5533.
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      -0.6621      0.014    -46.666      0.000      -0.690      -0.634
carat           1.1026      0.009    121.607      0.000       1.085       1.120
depth           0.0047      0.004      1.265      0.206      -0.003       0.012
table          -0.0169      0.002     -8.099      0.000      -0.021      -0.013
x              -0.3736      0.032    -11.806      0.000      -0.436      -0.312
y               0.4040      0.033     12.183      0.000       0.339       0.469
z              -0.1705      0.024     -7.045      0.000      -0.218      -0.123
cut_Good       -0.0159      0.006     -2.665      0.008      -0.028      -0.004
cut_Ideal       0.0360      0.004      8.175      0.000       0.027       0.045
cut_Premium     0.0388      0.005      8.446      0.000       0.030       0.048
color_E        -0.0468      0.006     -8.350      0.000      -0.058      -0.036
color_F        -0.0635      0.006    -11.168      0.000      -0.075      -0.052
color_G        -0.1009      0.006    -18.222      0.000      -0.112      -0.090
color_H        -0.2079      0.006    -35.215      0.000      -0.219      -0.196
color_I        -0.3241      0.007    -49.352      0.000      -0.337      -0.311
color_J        -0.4692      0.008    -58.037      0.000      -0.485      -0.453
clarity_IF      1.0174      0.016     63.636      0.000       0.986       1.049
clarity_SI1     0.6584      0.014     48.189      0.000       0.632       0.685
clarity_SI2     0.4476      0.014     32.536      0.000       0.421       0.475
clarity_VS1     0.8567      0.014     61.441      0.000       0.829       0.884
clarity_VS2     0.7850      0.014     57.135      0.000       0.758       0.812
clarity_VVS1    0.9614      0.015     65.043      0.000       0.932       0.990
clarity_VVS2    0.9504      0.014     66.178      0.000       0.922       0.979
==============================================================================
Omnibus:                     4625.241   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17350.397
Skew:                           1.190   Prob(JB):                         0.00
Kurtosis:                       7.050   Cond. No.                         55.5
==============================================================================

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

Model Score

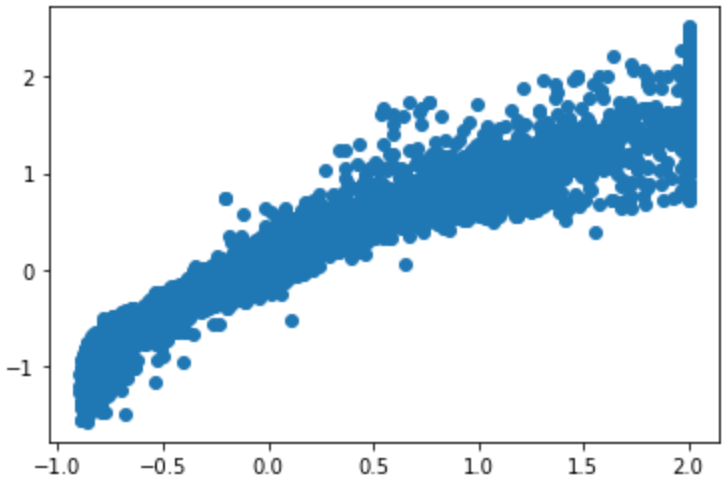
```
1  # Finding the model score i.e. coeff of determinant
2
3  regression_model.score(X_test, y_test)
```

```
0.9381643998102491
```

```
1  #Root Mean Squared Error
2  np.sqrt(mse)
```

```
0.2077253437089782
```

mpg prediction



If improved accuracy is expected, dropping the depth column in the iteration for better results would be a solution.

Here's the formula:
(-0.66) * Intercept + (1.1) * carat + (-0.02) * table + (-0.38) * x + (0.39) * y + (-0.15) * z + (-0.01) * cut_Good + (0.04) * cut_Ideal + (0.04) * cut_Premium + (-0.05) * color_E + (-0.06) * color_F + (-0.1) * color_G + (-0.21) * color_H + (-0.32) * color_I + (-0.47) * color_J + (1.02) * clarity_IF + (0.66) * clarity_SI1 + (0.45) * clarity_SI2 + (0.86) * clarity_VS1 + (0.79) * clarity_VS2 + (0.96) * clarity_VVS1 + (0.95) * clarity_VVS2 +

Problem 1.4:

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

Solution:

Overview:

In this business case study, we're expected to help the Gem Stones co ltd to predict the price for the cubic zirconia and apprehension on the different price ranges. It should be noted that jewelry pricing is a vaguely understood process that typically results in very high

retail prices, and then shockingly low resale prices. Below are the insights from the exploratory data analysis and linear regression model build from the dataset is given.

Business insights:

- While finding the unique categorical value during the exploratory data analysis it was observed that the ideal cut had given profit to the company.
- In terms of the colors, H, I, J turned out to be profit-generating.
- Similarly, the clarity levels VS1, VS2, SI1 were the most profitable among all. And SI2 turned out to be the costliest in terms of price
- While comparing the cut and color, J was found as the most profitable and similarly WS2 during the comparison of cut and clarity.
- Coming to the built linear regression model, in the training set 94% discrepancy in the price has been explained

Business Recommendations:

- By talking in favor of revenue-generation from the product sales the focus will have to be customer preference, market preference, highest selling items, and factors pressing customer demand.
- In terms of Cut, the customer preference and sale goes with ideal, premium, and very good cuts. Hence, these highly selling products will be the prime focus in the marketing campaigns.
- The marketing ads can be broadcasted by focusing on the cut perfection, customer acceptance, quality, and pricing.
- Talking about the best 5 attributes that are most important: Cut, Carat, Y (Width of the stone), clarity VS1, VS2, SI1, and price.

Problem 2: Logistic Regression and LDA - Holiday Package Case Study

Problem Statement

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

Dataset for Problem 2: Holiday_Package.csv

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

Problem 2.1:

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

Solution:

Getting the basic info of the data:

Top 10 values:

```
1 df.head(10)
```

	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
5	6	yes	61590	42	12	0	1	no
6	7	no	94344	51	8	0	0	no
7	8	yes	35987	32	8	0	2	no
8	9	no	41140	39	12	0	0	no
9	10	no	35826	43	11	0	2	no

Bottom 10 values:

```
1 df.tail(10)
```

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
862	863	no	66900	35	10	1	1	yes
863	864	no	35290	51	9	0	1	yes
864	865	no	25527	41	5	1	0	yes
865	866	yes	44057	35	9	0	2	yes
866	867	yes	22643	42	14	0	0	yes
867	868	no	40030	24	4	2	1	yes
868	869	yes	32137	48	8	0	0	yes
869	870	no	25178	24	6	2	0	yes
870	871	yes	55958	41	10	0	1	yes
871	872	no	74659	51	10	0	0	yes

Data shape and basic info:

```
1 df.shape
```

```
(872, 8)
```

```
1 df.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
```

Insights:

- The present variables are both numeric and categorical types in nature - i.e. int, and object data types are present
- There are 8 variables and 872 records

Calculation of basic statistical data

```
#Calculating basic statistical data
df.describe().T
```

	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

```
#Calculating basic statistical data including all information\
df.describe(include = "all").T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Unnamed: 0	872	NaN	NaN	NaN	436.5	251.869	1	218.75	436.5	654.25	872
Holliday_Package	872	2	no	471	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Salary	872	NaN	NaN	NaN	47729.2	23418.7	1322	35324	41903.5	53469.5	236961
age	872	NaN	NaN	NaN	39.9553	10.5517	20	32	39	48	62
educ	872	NaN	NaN	NaN	9.30734	3.03626	1	8	9	12	21
no_young_children	872	NaN	NaN	NaN	0.311927	0.61287	0	0	0	0	3
no_older_children	872	NaN	NaN	NaN	0.982798	1.08679	0	0	1	2	6
foreign	872	2	no	656	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- **Target Variable:** Holiday package

Check for duplicate data:

```
#Checking Null Value
df.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 duplicates in data
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
```

```
Number of duplicate rows = 0
```

Finding unique values in categorical variables:

```
HOLLIDAY_PACKAGE :  2
yes    401
no     471
Name: Holliday_Package, dtype: int64


FOREIGN :  2
yes    216
no     656
Name: foreign, dtype: int64
```

Finding target percentage:

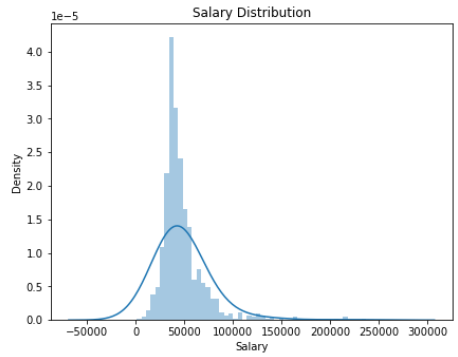
```
1 df.Holliday_Package.value_counts(1)
```

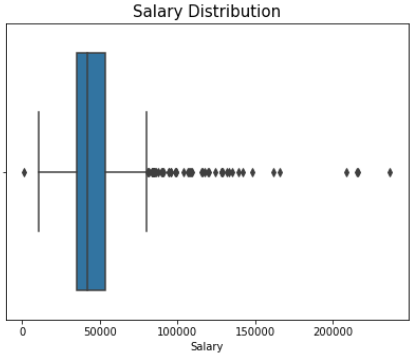
```
no     0.540138
yes    0.459862
Name: Holliday_Package, dtype: float64
```

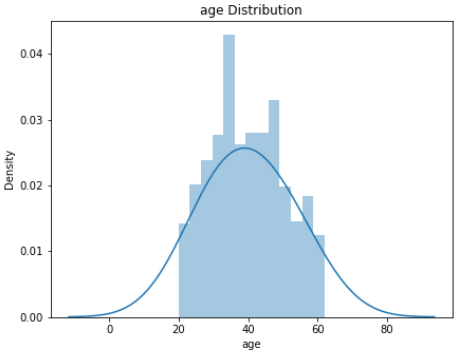
- This result indicates that 45% of the employees have an interest in getting the holiday package.

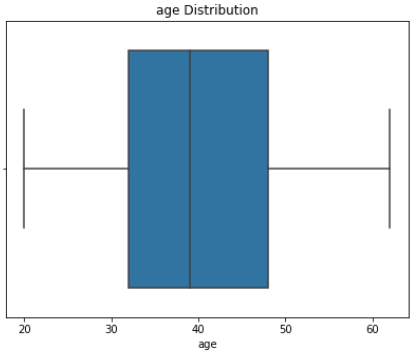
Performing univariate and bivariate analysis

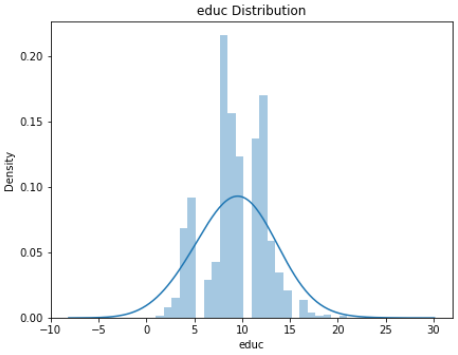
- Please refer to the Jupyter notebook to look into the codes.

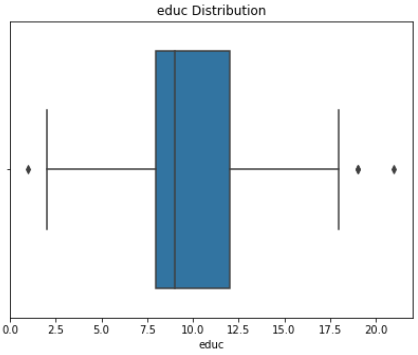

Salary Distribution
1e−5
Density
Salary

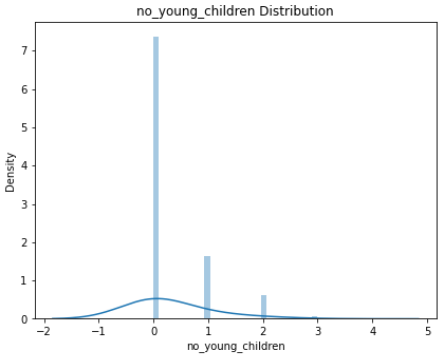

Salary Distribution
Salary

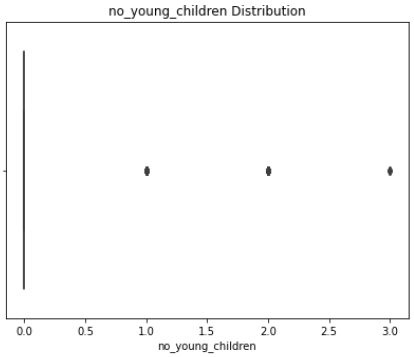

age Distribution
Density
age

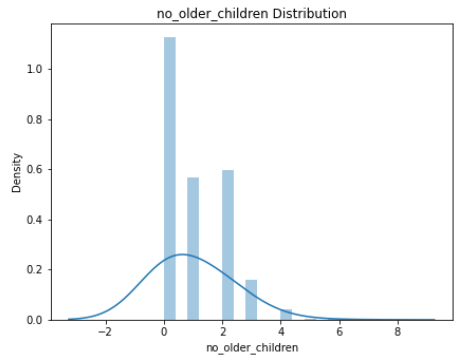

age Distribution
age

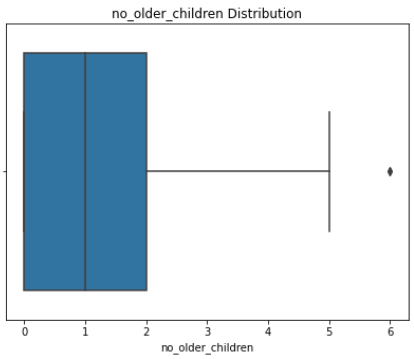

educ Distribution
Density
educ


educ Distribution
educ


no_young_children Distribution
Density
no_young_children


no_young_children Distribution
no_young_children


no_older_children Distribution
Density
no_older_children


no_older_children Distribution
no_older_children

Insights:

- The distribution of salary data has positive skewness. The data range is 0 to 50,000.
- The distribution of age is in the normal distribution and it ranges from 30 to 50.
- The distribution of education in the table has slightly negative skewness. The distribution range is between 7.5 to 12.5
- The distribution of the number ofnumber of young children (younger than 7 years) has the least value.
- The distribution of the number of older children has no skewness and it ranges in between 0 to 2

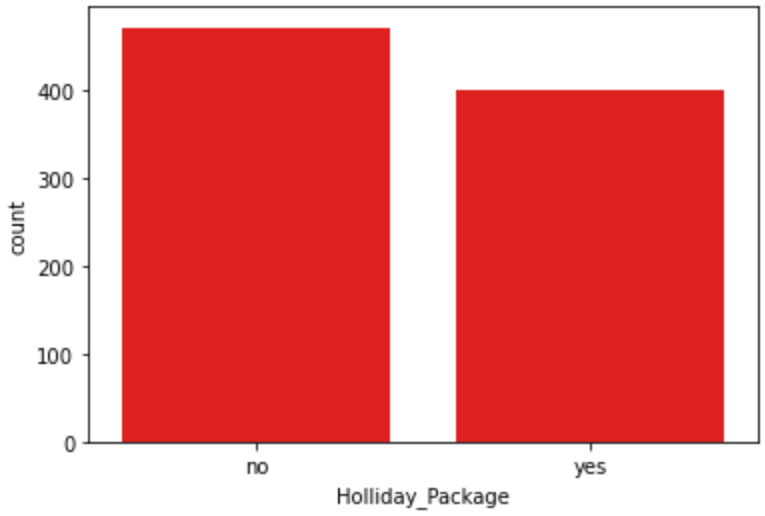
Skewness Measurement:

```
1 #Measuring the skewness
2 df.skew()
```

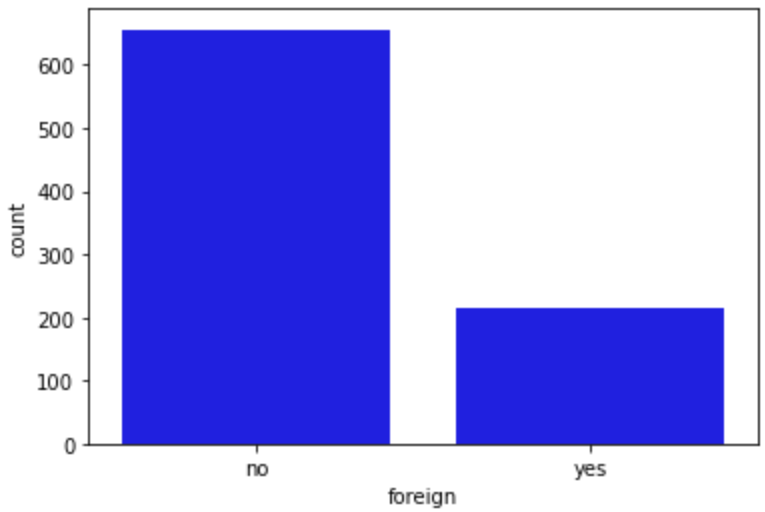
```
Unnamed: 0           0.000000
Salary               3.103216
age                  0.146412
educ                -0.045501
no_young_children    1.946515
no_older_children    0.953951
dtype: float64
```

Graphical comparison of categorical variables:

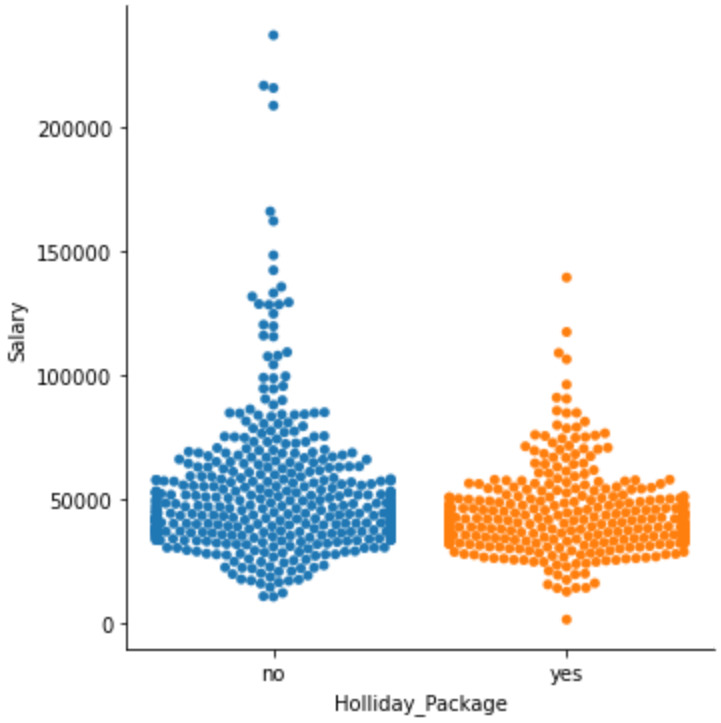
Holiday Package:



Foreign:

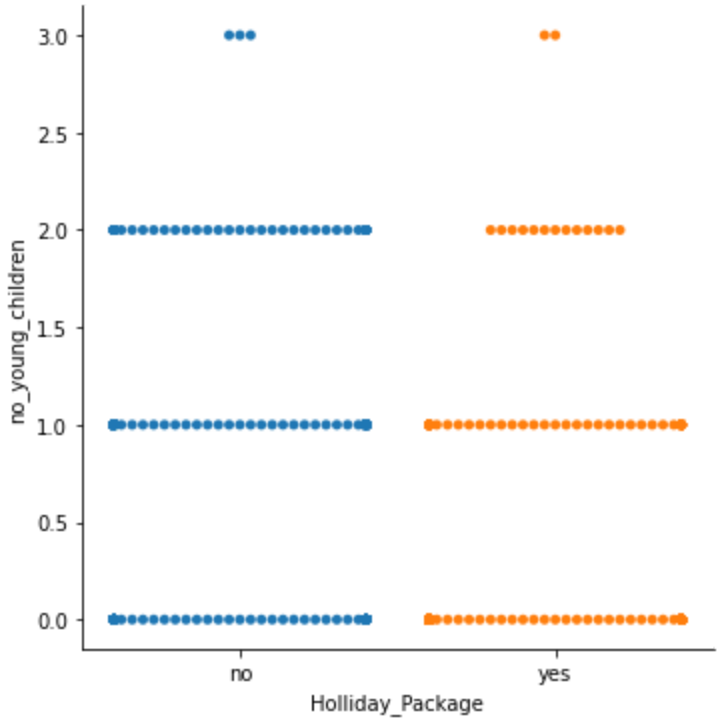


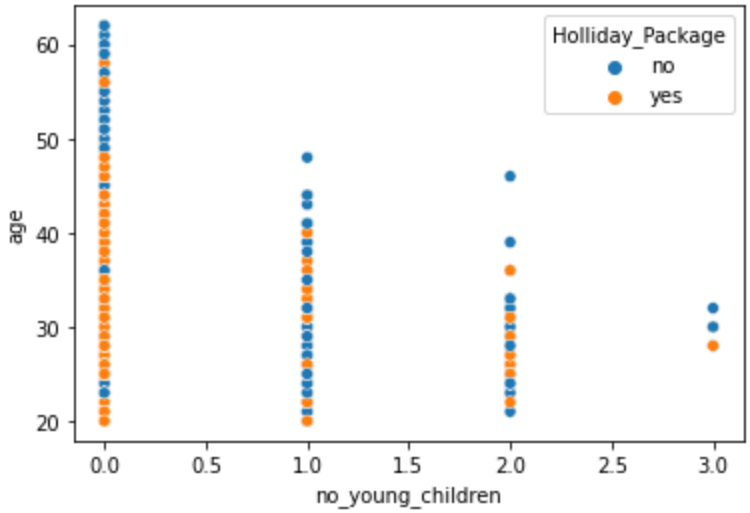
Comparison between Holiday Package and Salary:



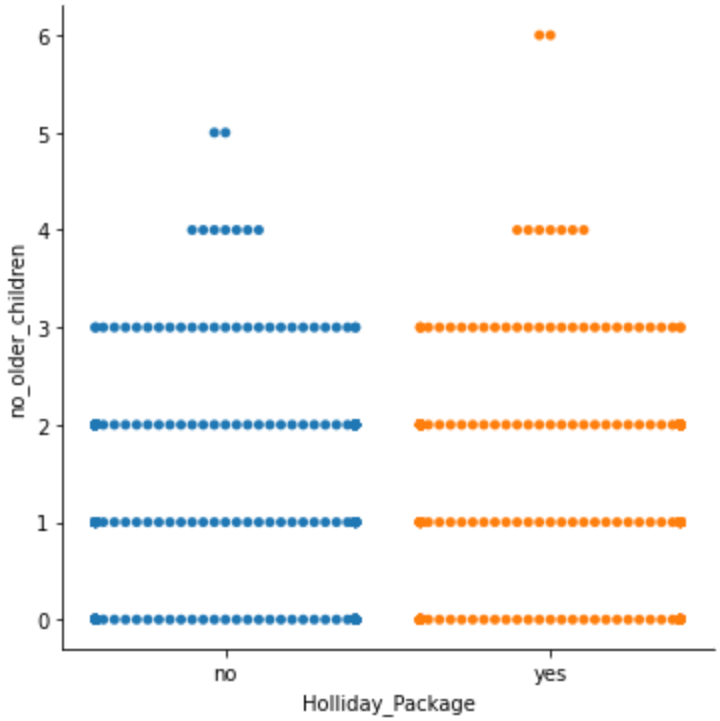
- It can be observed that employee below salary range 150000 have tendency of mandatory opting for holiday package

Comparison between Holiday Package & No of young children:

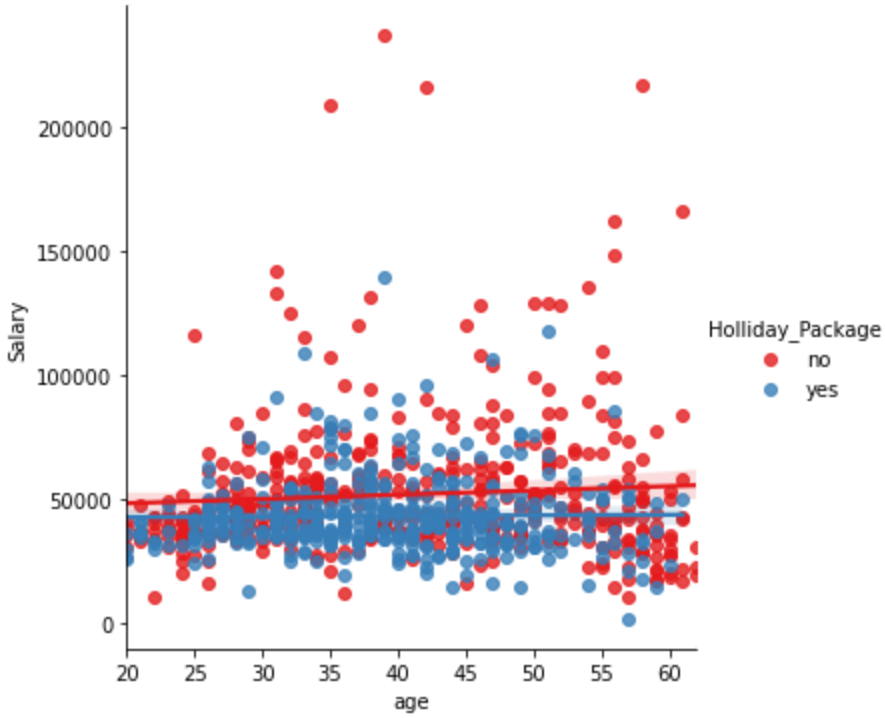


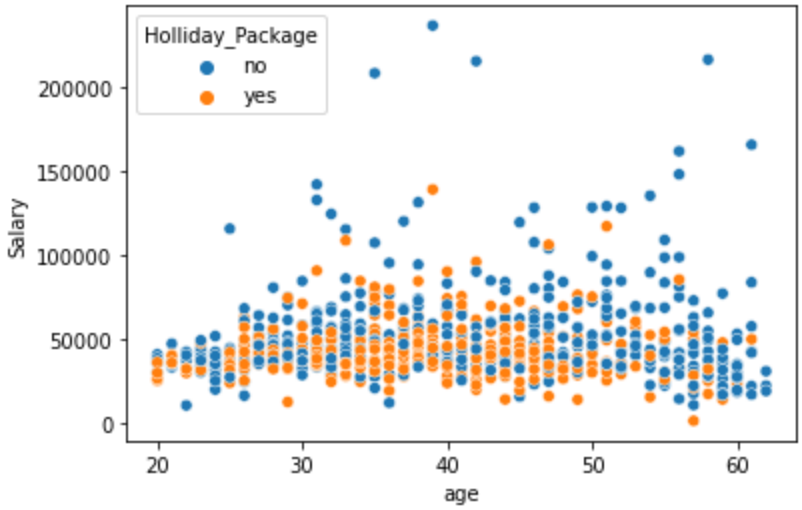


Comparison between Holiday Package & No of older children:



Distribution range Salary v/s Age for opting Holiday_Package:

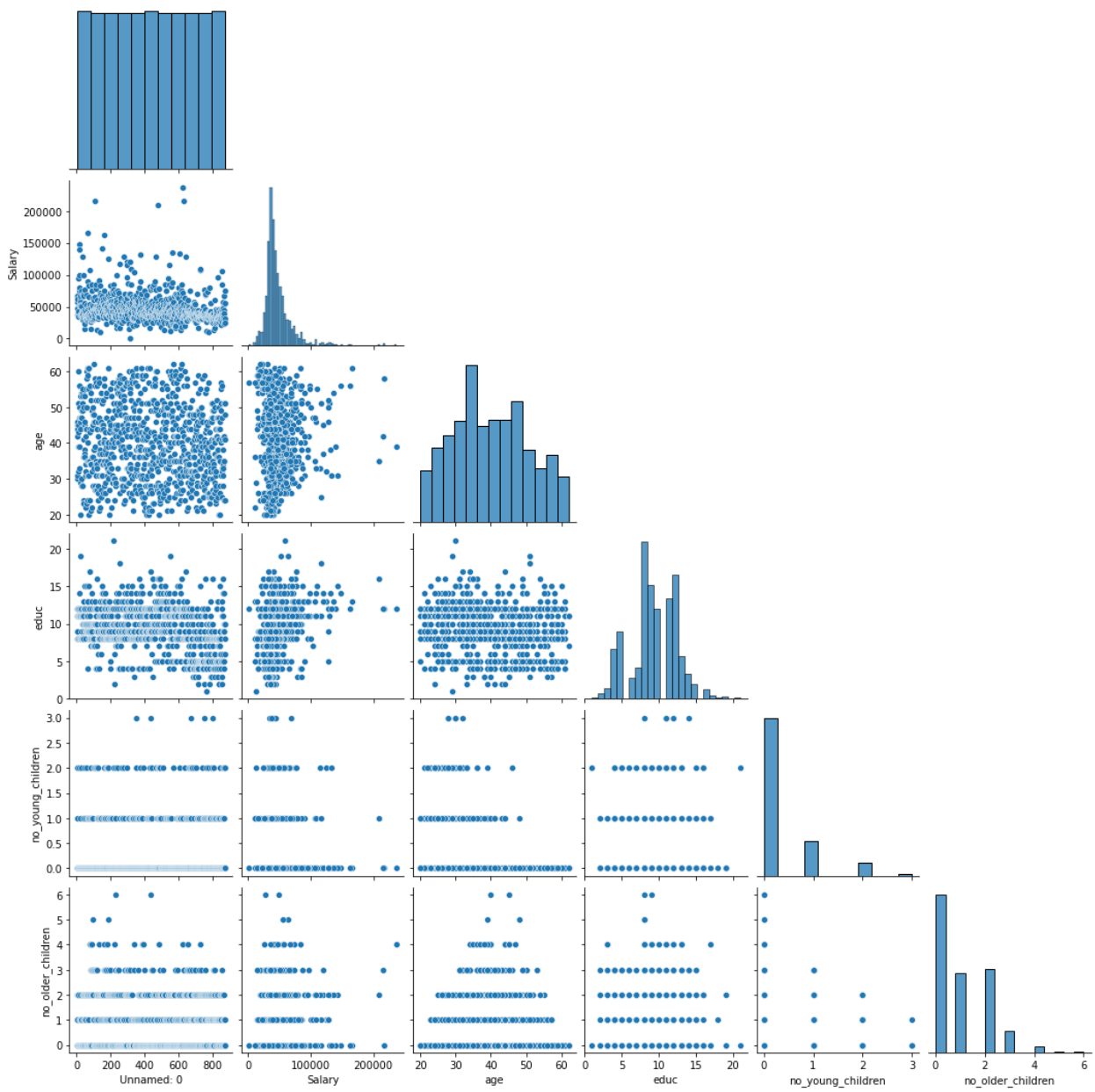




Insights:

- The holiday package is preferred for employees salary range below 50,000
- Employees' salaries ranging below 50,000 are of age range 30 -50. Hence, this age group has actively opted for the holiday package
- Employees aged over 50 to 60 have shown a tendency of not opting for the holiday package.

Pairplot to display data distribution (Bivariate analysis):

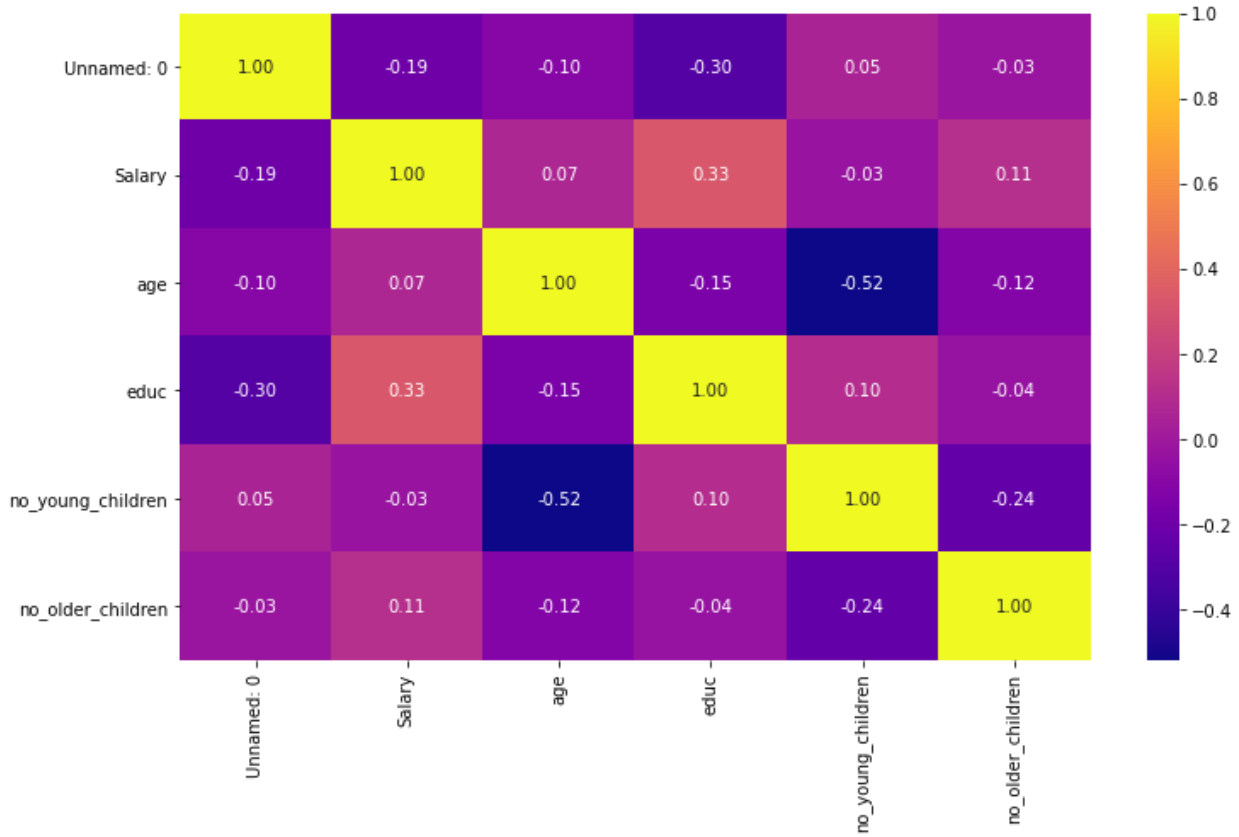


Correlation Check

```
1 df.corr().T
```

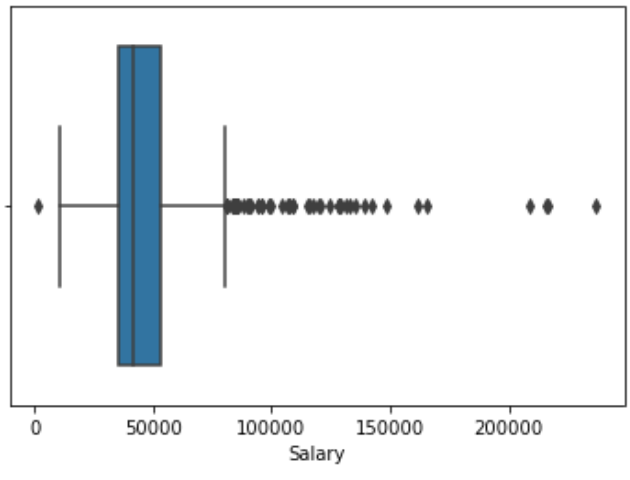
	Unnamed: 0	Salary	age	educ	no_young_children	no_older_children
Unnamed: 0	1.000000	-0.193249	-0.103782	-0.296015	0.052146	-0.025852
Salary	-0.193249	1.000000	0.071709	0.326540	-0.029664	0.113772
age	-0.103782	0.071709	1.000000	-0.149294	-0.519093	-0.116205
educ	-0.296015	0.326540	-0.149294	1.000000	0.098350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

Heatmap displaying the correlation:

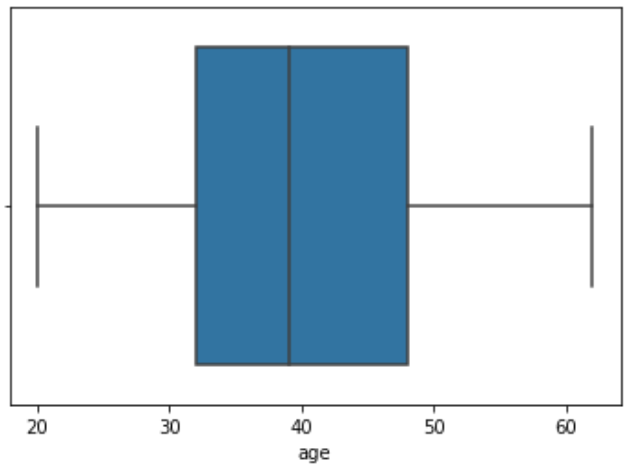


Finding Outlier and Outlier Treatment:

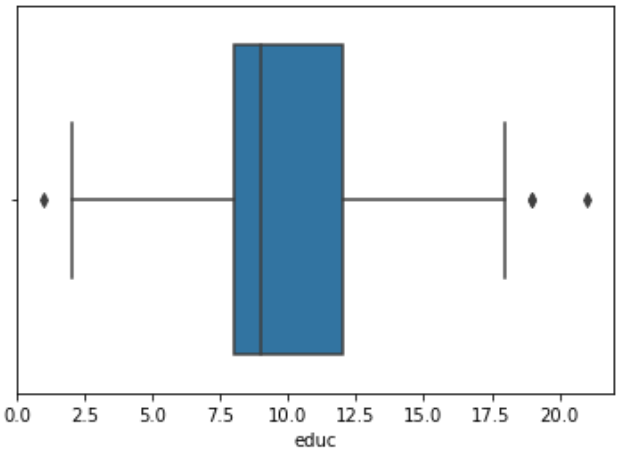
Salary



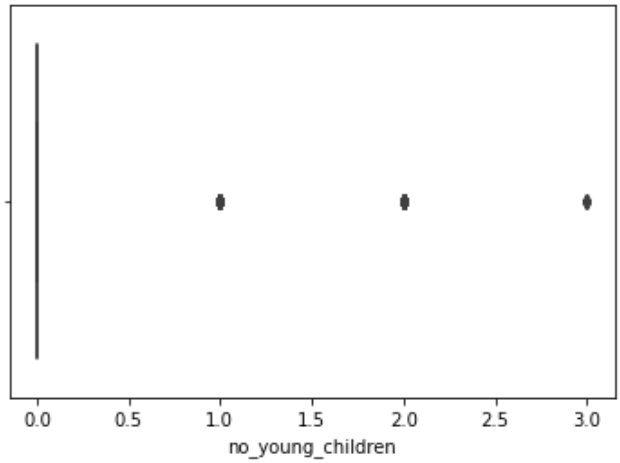
Age



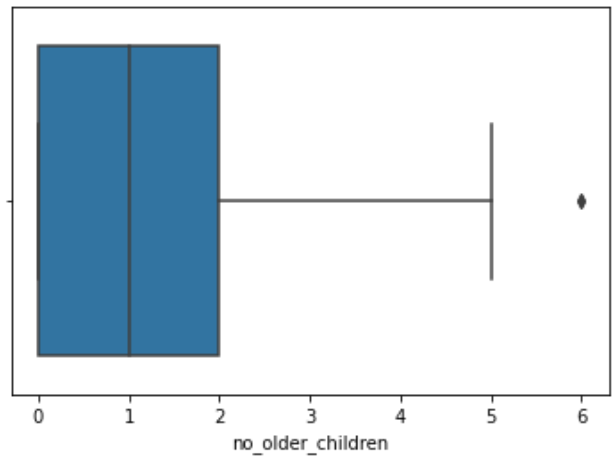
Educ



No_young_children



No_older_children



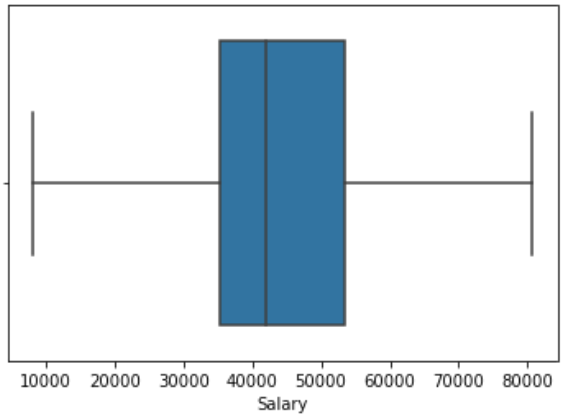
Observation from the boxplots

- Outlier found in following columns - Salary, age, educ, no_young_children, no_older_children
- Outlier treatment is required

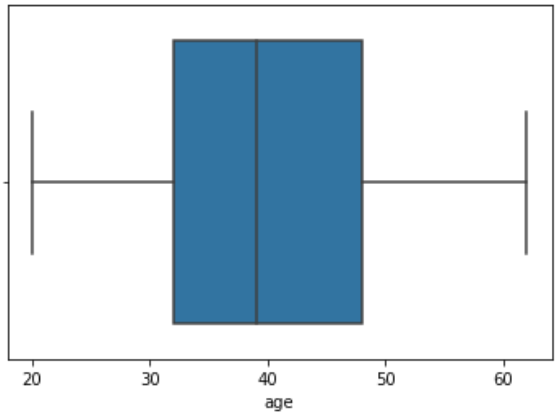
Outlier Treatment:

Please check the Jupyter notebook attached to view the codes of outlier treatment: Post outlier treatment, graphs are given below:

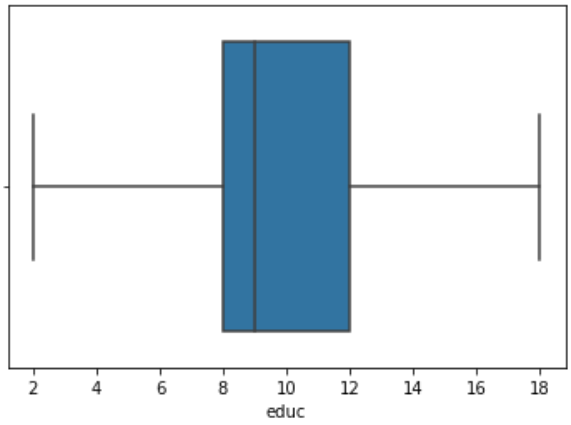
Salary



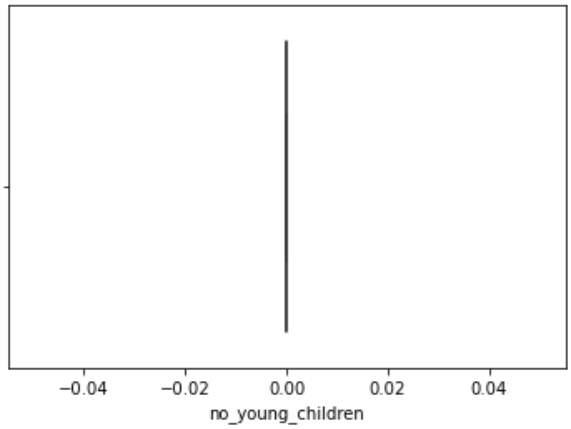
Age



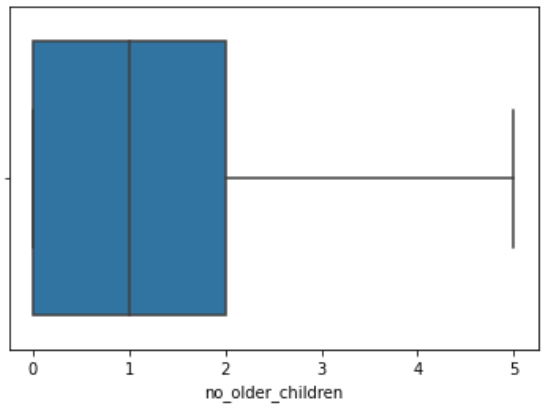
Educ



No_young_children



No_older_children



- No more outliers in the data, all outliers have been treated.

Problem 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:

Logistic Regression Model:

Step 1: Convert categorical to dummy variables in data

```
1 data_df = pd.get_dummies(df2, columns=['Holliday_Package','foreign'], drop_first = True)
```

```
1 data_df.head()
```

	Salary	age	educ	no_young_children	no_older_children	Holliday_Package_yes	foreign_yes
0	48412.0	30.0	8.0	0.0	1.0	0	0
1	37207.0	45.0	8.0	0.0	1.0	1	0
2	58022.0	46.0	9.0	0.0	0.0	0	0
3	66503.0	31.0	11.0	0.0	0.0	0	0
4	66734.0	44.0	12.0	0.0	2.0	0	0

Step 2: Data Split: Split the data into train and test (70:30).

```
1 X = data.drop('Holliday_Package_yes', axis=1)
2 y = data['Holliday_Package_yes']
```

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

Step 3: Building Logistic Regression Model (Grid Search method):

- Please refer to the Jupyter notebook attached to view the codes of the model.

```
#Applying GridSearchCV
grid={'penalty':['l1','l2','none'],
      'solver':['lbfgs', 'liblinear'],
      'tol':[0.0001,0.000001]}
```

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

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

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

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

```
print(grid_search.best_params_,'\n')
print(grid_search.best_estimator_)
```

```
{'penalty': 'l2', 'solver': 'liblinear', 'tol': 1e-06}

LogisticRegression(max_iter=100000, n_jobs=2, solver='liblinear', tol=1e-06)
```

```
best_model = grid_search.best_estimator_
```

- Grid-search is used to find the optimal hyperparameters of a model which results in the most ‘accurate’ predictions.
- It’s seen here that the Grid search method gave a liblinear solver. This liblinear solver is most suitable for small datasets.
- Penalty and tolerance have been found using this method

Linear Discriminant Analysis (LDA) Model:

Step 1: Data check (Optional):

- I have checked all the data info again, just to be assured that progress is going fine.

```
1 df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 872 entries, 0 to 871
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Holliday_Package   872 non-null    object
 1   Salary             872 non-null    float64
 2   age                872 non-null    float64
 3   educ               872 non-null    float64
 4   no_young_children  872 non-null    float64
 5   no_older_children  872 non-null    float64
 6   foreign            872 non-null    object
dtypes: float64(5), object(2)
memory usage: 47.8+ KB
```

```
1 df1.head(10)
```

	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
0	0	48412.00	30.0	8.0	0.0	1.0	0
1	1	37207.00	45.0	8.0	0.0	1.0	0
2	0	58022.00	46.0	9.0	0.0	0.0	0
3	0	66503.00	31.0	11.0	0.0	0.0	0
4	0	66734.00	44.0	12.0	0.0	2.0	0
5	1	61590.00	42.0	12.0	0.0	1.0	0
6	0	80687.75	51.0	8.0	0.0	0.0	0
7	1	35987.00	32.0	8.0	0.0	2.0	0
8	0	41140.00	39.0	12.0	0.0	0.0	0
9	0	35826.00	43.0	11.0	0.0	2.0	0

Step 2: Data Split: Split the data into train and test (70:30):

```
1 X = df1.drop('Holliday_Package',axis=1)
2 Y = df1.pop('Holliday_Package')
```

```
1 X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.30,random_state=1,stratify = Y)
```

Step 3: Building Linear Discriminant Analysis (LDA) Model:

- Please refer to the Jupyter notebook attached to view the codes of the model.

```
#Building a LDA Model
clf = LinearDiscriminantAnalysis()
model=clf.fit(X_train,Y_train)
```

```
# Training data class Prediction, cut-off value = 0.5
pred_class_train = model.predict(X_train)

# Test data class Prediction, cut-off value = 0.5
pred_class_test = model.predict(X_test)
```

Problem 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:

Logistic Regression Model:

Prediction on the training set

```
# Prediction on the training set

ytrain_predict = best_model.predict(X_train)
ytrain_predict
```

```
array([1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0], dtype=uint8)
```

Prediction on the test set

```
# Prediction on the test set

ytest_predict = best_model.predict(X_test)
ytest_predict
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
      dtype=uint8)
```

Probabilities on the test set

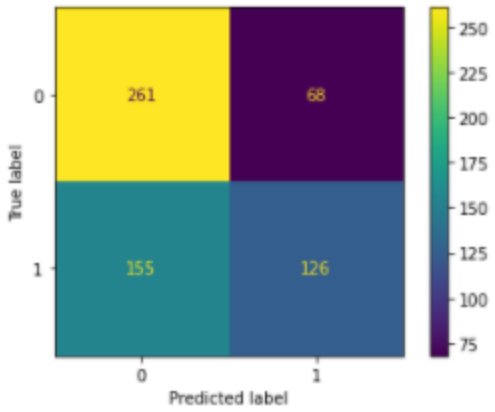
```
#Probabilities on the test set

ytest_predict_prob=best_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

	0	1
0	0.636523	0.363477
1	0.576651	0.423349
2	0.650835	0.349165
3	0.568064	0.431936
4	0.536356	0.463644

Confusion matrix on the training data

```
#Confusion matrix on the training data

plot_confusion_matrix(best_model,X_train,y_train)
print(classification_report(y_train, ytrain_predict),'\n');
```

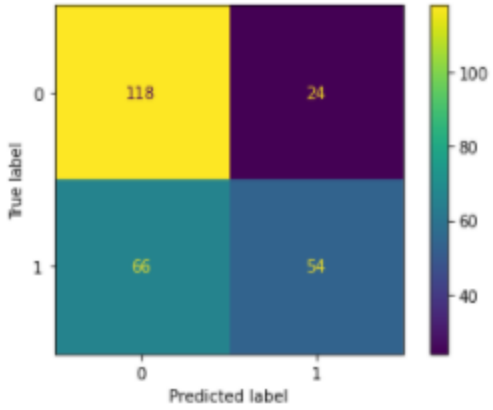
```
              precision    recall  f1-score   support

           0       0.63      0.79      0.70       329
           1       0.65      0.45      0.53       281

    accuracy                           0.63       610
   macro avg       0.64      0.62      0.62       610
weighted avg       0.64      0.63      0.62       610
```



Confusion matrix on the test data

```
#Confusion matrix on the test data

plot_confusion_matrix(best_model,X_test,y_test)
print(classification_report(y_test, ytest_predict),'\n');
```

```
              precision    recall  f1-score   support

           0       0.64      0.83      0.72       142
           1       0.69      0.45      0.55       120

    accuracy                           0.66       262
   macro avg       0.67      0.64      0.63       262
weighted avg       0.66      0.66      0.64       262
```

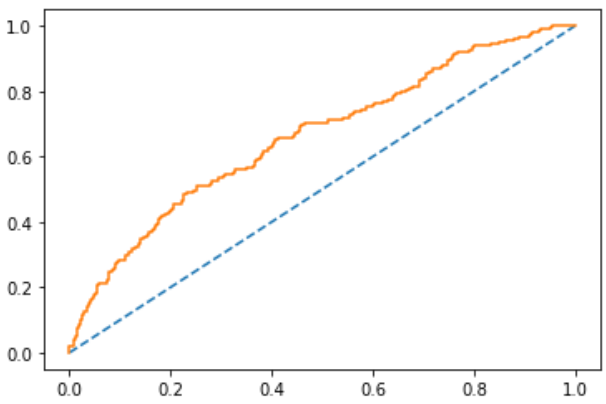


Accuracy of the training Data

```
#Accuracy of the training Data

lr_train_acc = best_model.score(X_train, y_train)
lr_train_acc
```

```
0.6344262295081967
```

AUC and ROC for the training data



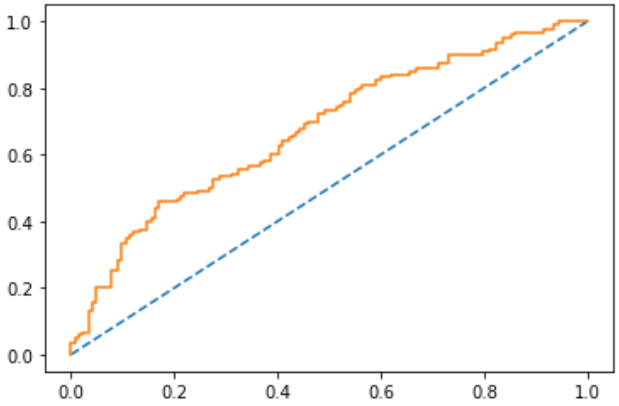
- The Accuracy is: 0.6344262295081967
- Area under the curve is: 0.661

Accuracy of the test Data

```
#Accuracy of the test Data

lr_test_acc = best_model.score(X_test, y_test)
lr_test_acc
```

```
0.6564885496183206
```

AUC and ROC for the training data



- The Accuracy is: 0.6564885496183206
- Area under the curve is: 0.675

Linear Discriminant Analysis (LDA) Model:

Prediction on the training set

```
pred_class_train
```

```
array([1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0]], dtype=int8)
```

Prediction on the test set

```
pred_class_test
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
      dtype=int8)
```

Training data probability prediction

```
# Training data probability prediction
pred_prob_train = model.predict_proba(X_train)
```

```
lda_train_acc = model.score(X_train,Y_train)
lda_train_acc
```

```
0.6327868852459017
```

- The accuracy of training data is: 0.6327868852459017

Test data probability prediction

```
# Test data probability prediction
pred_prob_test = model.predict_proba(X_test)
```

```
lda_test_acc = model.score(X_test,Y_test)
lda_test_acc
```

```
0.6564885496183206
```

- The accuracy of test data is: 0.6564885496183206

Classification report and confusion matrix of training data

```
print(classification_report(Y_train, pred_class_train))
```

```
              precision    recall  f1-score   support

           0       0.62      0.80      0.70       329
           1       0.65      0.44      0.52       281

    accuracy                           0.63       610
   macro avg       0.64      0.62      0.61       610
weighted avg       0.64      0.63      0.62       610
```

```
confusion_matrix(Y_train, pred_class_train)
```

```
array([[263,  66],
       [158, 123]], dtype=int64)
```

Classification report and confusion matrix of test data

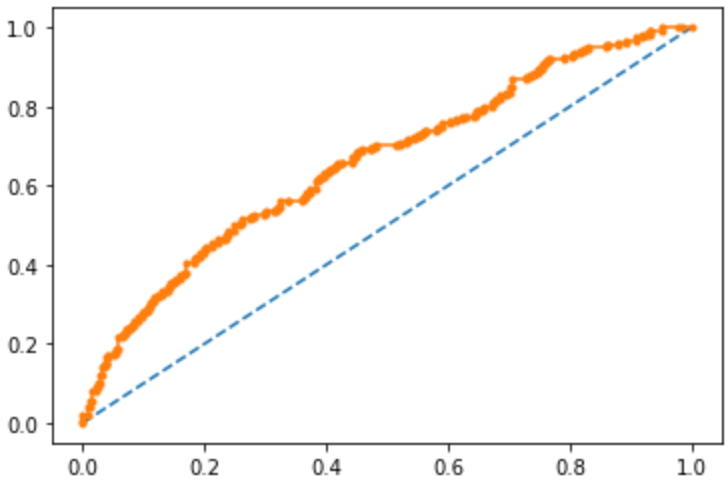
```
print(classification_report(Y_test, pred_class_test))
```

```
              precision    recall  f1-score   support

           0       0.64      0.83      0.72       142
           1       0.69      0.45      0.55       120

    accuracy                           0.66       262
   macro avg       0.67      0.64      0.63       262
weighted avg       0.66      0.66      0.64       262
```

```
confusion_matrix(Y_test, pred_class_test)
```

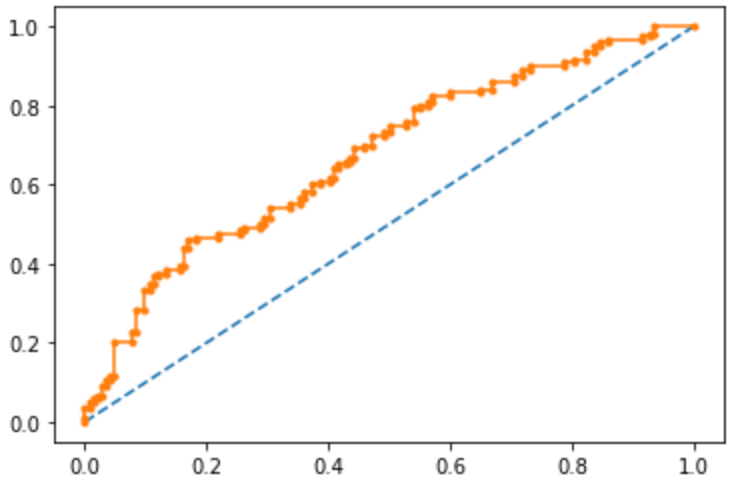
```
array([[118,  24],
       [ 66,  54]], dtype=int64)
```

AUC and ROC for the training data



- AUC for the Training Data: 0.661

AUC and ROC for the test data



- AUC for the Test Data: 0.675

Comparing the Linear Regression and Linear Discriminant Analysis (LDA) Models

- Please refer to the Jupyter notebook to look into the codes.

Precision, Recall and F1 Scores of Train and Test data of Linear Regression Model

• lr_train_precision 0.65 • lr_train_recall 0.45 • lr_train_f1 0.53	• lr_test_precision 0.69 • lr_test_recall 0.45 • lr_test_f1 0.55

Precision, Recall and F1 Scores of Train and Test data of Linear Regression Model

• lda_train_precision 0.65 • lda_train_recall 0.44 • lda_train_f1 0.53	• lda_test_precision 0.69 • lda_test_recall 0.45 • lda_test_f1 0.55

Comparing the Linear Regression and Linear Discriminant Analysis (LDA) Models - based on their respective Precision, Recall and F1 Scores of Train and Test data

- Please refer to the Jupyter notebook to look into the codes.

	LR Train	LR Test	LDA Train	LDA Test
Accuracy	0.63	0.66	0.63	0.66
AUC	0.66	0.68	0.66	0.68
Recall	0.45	0.45	0.44	0.45
Precision	0.65	0.69	0.65	0.69
F1 Score	0.53	0.55	0.52	0.55

Model Section:

- I would prefer to select the Linear Discriminant Analysis (LDA) Model
- It is a very close comparison as Accuracy and AUC are the same, and Recall, Precision and F1 scores are almost the same. However, according to the best practices when dealing with categorical independent variables, the LDA model is preferred. Hence, that would be my selection.

Problem 2.4:

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

Solution:

Overview:

In this business case, a travel agency is looking for statistical evident predictions on employees likely to prefer to opt for the holiday package or not. The given dataset has information about the economic, behavioral, and age group. Based on the exploratory data analysis and models built, below are the business insights.

Insights:

- The comparison between the employee age range, salary, holiday package preference shows that the holiday package is preferred by employees salary range below 50,000.
- Employees' salaries ranging below 50,000 are of age range 30 -50. Hence, this age group has actively opted for the holiday package

- Employees aged over 50 to 60 have shown a tendency of not opting for the holiday package.
- On the other hand, employees with a salary of more than 150,000 are also not opting for the holiday package. This salary range has employees from both 30 - 50 and 50 - 60 age groups
- The holiday package is also not preferred by the employees having young children.
- Employees with older children are opting for the package normally.
- We can see three major groups where strategy implementation is needed to grow the holiday package subscription:
    - Old age group - Employees aged between 50-60
    - Elite group - Salary range more than 150,000
    - Employees with small children

Business Recommendations:

The business recommendation can be given based on the targeting segments created. Let's look into all these one by one:

- Old age group - Employees aged between 50-60
    - The holiday package of the holy-places (religious places) can be pitched to these folks
    - A survey asking their preferences, beliefs can help in creating a personalized package, which will have a high probability of acceptance
- Elite group - Salary range more than 150,000
    - This group would majorly prefer to have an experience-based holiday package.
    - Looking at their salary range, they can be offered for abroad trip for the age group between 30 - 50. This package can have some personalized adventure or stay experience deal included
    - For the age group between 50 - 60, this package can have a family trip experience both domestic and international. This package can also have a whole trip experience with a personal local guide at a holy place.
- Employees with small children
    - These folks can be offered a trip to parks such as Disneyland (if higher salary range) or domestic parks such as Wonderla, Queens Land, etc. (if low salary range)
    - Kids' special experiences such as live cartoon shows will be an add-on advantage in the package and increase the chance of opting for the package.
- Apart from these groups, families with older children can be offered a family holiday plan to keep up/continue their tendency of opting for the holiday packages.
