#Prepared By

Jitendra Kumar Nayak

Date:31/10/2021

#Business Report

Predictive Modelling

Table of Contents

List of Figures

List of Tables

Problem 1

Executive Summary

The company Gem Stones Co. Ltd, a cubic zirconia manufacturer provided a 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. Company seeks help in predicting the price for the stone on the basis of the details given in the dataset so it can distinguish between higher profitable stones and lower profitable stones so as to have a better profit share. Also, the best 5 attributes that are most important need to be provided.

Introduction

The objective is to use Linear Regression supervised learning predictive models to predict the prices for different stones based on the features in the data set. Also, insights about data and the business problem have been provided with Exploratory Data Analysis.

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

Let us have a look at first and last few records of data:

	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

Table 1: Sample of Data(Problem 1) - 1

(above table generated with .head() function of PANDAS library in Python)

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

Table 2: Sample of Data(Problem 1) - 2

(above table generated with .tail() function of PANDAS library in Python)

Let us drop the 'Unnamed: 0' feature which seems to be just indices and proceed further.

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 worst and J the best.
Clarity	cubic zirconia Clarity refers to the absence of the Inclusions and Blemishes. (In order from Worst to Best) IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1
Depth	The Height of 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.

Exploratory Data Analysis

Let us check the data types of variables and the missing values(if any):

```
<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
```

Table 3: General Information of Data(Problem1)

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- shape of data is (26967x10), after dropping 'Unnamed: 0' feature
- there are total 26967 entries(rows) of stone samples in data indexed from 0 to 26967
- there are 10 columns in total(carat, cut, color, clarity, depth, table, x, y, z, price)
- variables 'carat', 'depth', 'table', 'x', 'y', and 'z' are of data-type float(64 bits)
- variables 'cut', 'color', and 'clarity' are object type data
- variable 'price' is of data-type integer(64 bits)
- the memory usage by data is 2.1+MB
- except for 'depth' feature which has 697 null values, there is no null value in data set as other all columns have 26967 non null objects

'df.isnull().sum()' also confirms the null value count in the data set.

There are 34 duplicate entries that have been identified by 'df.duplicated()', but we retain them as different samples may have the same values in the features.

(Heatmap and Pairplot provided after Univariate Analysis)

Let us now move to the general description of numeric data(minimum values, maximum values, measures of central tendencies like mean, median, mode etc.):

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode
carat	0.80	0.48	0.2	0.40	0.70	1.05	4.50	4.30	0.65	0.60	0.3
depth	61.75	1.41	50.8	61.00	61.80	62.50	73.60	22.80	1.50	0.02	62.0
table	57.46	2.23	49.0	56.00	57.00	59.00	79.00	30.00	3.00	0.04	56.0
x	5.73	1.13	0.0	4.71	5.69	6.55	10.23	10.23	1.84	0.20	4.38
y	5.73	1.17	0.0	4.71	5.71	6.54	58.90	58.90	1.83	0.20	4.35
z	3.54	0.72	0.0	2.90	3.52	4.04	31.80	31.80	1.14	0.20	2.69
price	3939.52	4024.86	326.0	945.00	2375.00	5360.00	18818.00	18492.00	4415.00	1.02	544

Table 4: General Description of Numeric Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- 'mean' and 'max' values in the features 'carat', 'z' and 'y' are distant apart, indicating the need for inspection
- rest all values seem to be correct
- the coefficient of variation is quite high for 'price' (1.02)

Let us now move to the general description of 'object' data:

	count	unique	top	freq
cut	26967	5	Ideal	10816
color	26967	7	G	5661
clarity	26967	8	SI1	6571

Table 5: General Description of 'Object' type data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

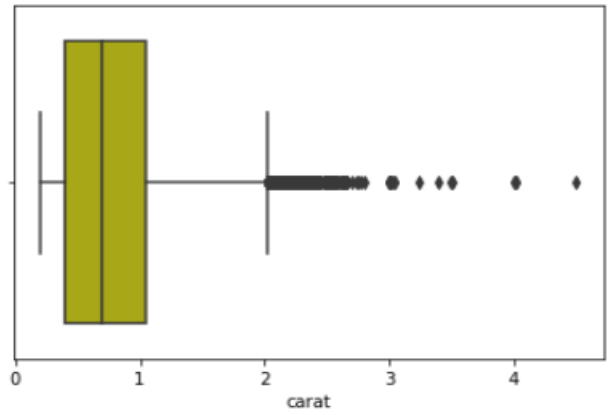
The object data seems to be fine.

Let us now go for the Univariate analysis of numeric features in the data set. We will go for general descriptions, distribution plots with kernel density estimates and boxplots for all the features one by one:

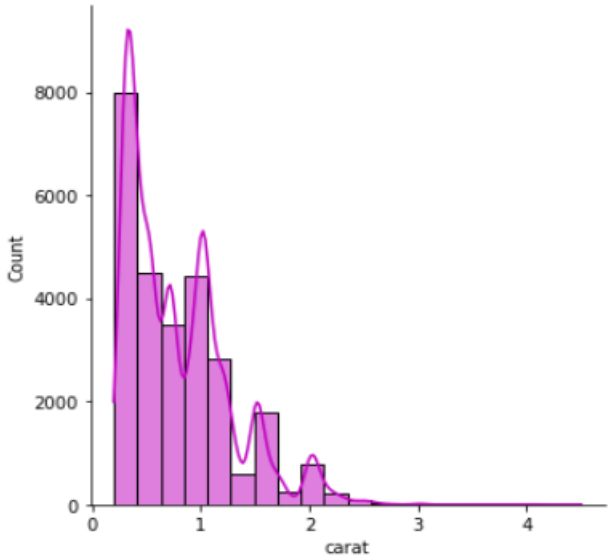
(zoom to at least 125% for better visibility)

```
Description of carat
----------------------------
count    26967.000000
mean         0.798375
std          0.477745
min          0.200000
25%          0.400000
50%          0.700000
75%          1.050000
max          4.500000
Name: carat, dtype: float64
```

BoxPlot of carat

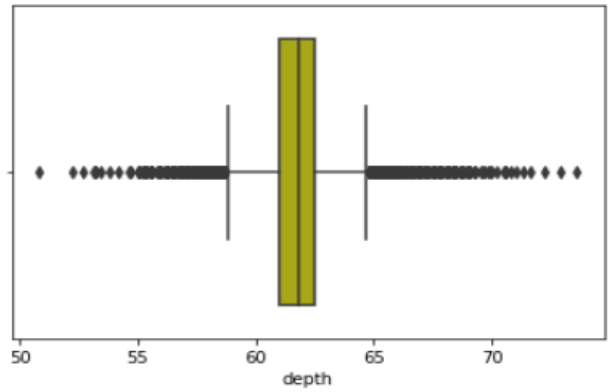


Distribution of carat

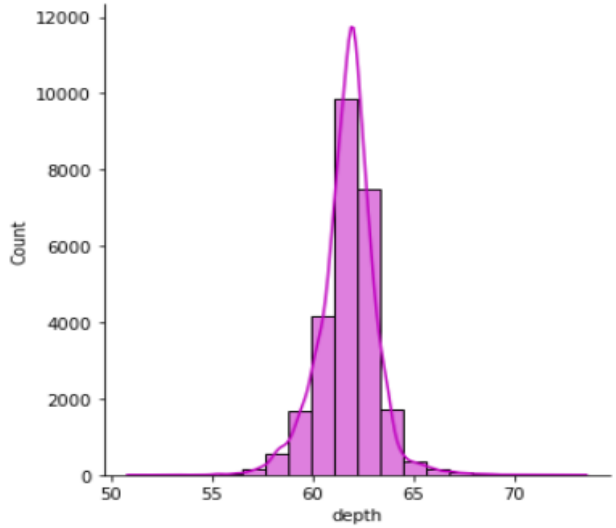



```
Description of depth
----------------------------
count    26270.000000
mean        61.745147
std          1.412860
min         50.800000
25%         61.000000
50%         61.800000
75%         62.500000
max         73.600000
Name: depth, dtype: float64
```

BoxPlot of depth

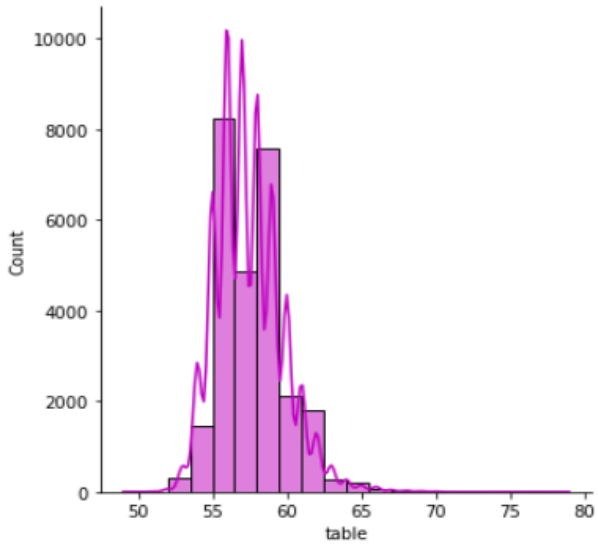


Distribution of depth

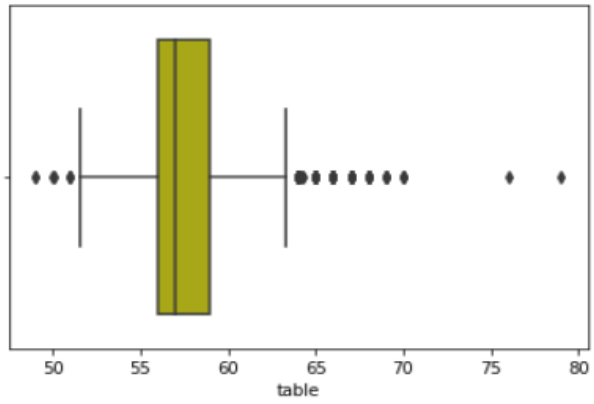


```
Description of table
----------------------------
count    26967.000000
mean        57.456080
std          2.232068
min         49.000000
25%         56.000000
50%         57.000000
75%         59.000000
max         79.000000
Name: table, dtype: float64
```

Distribution of table

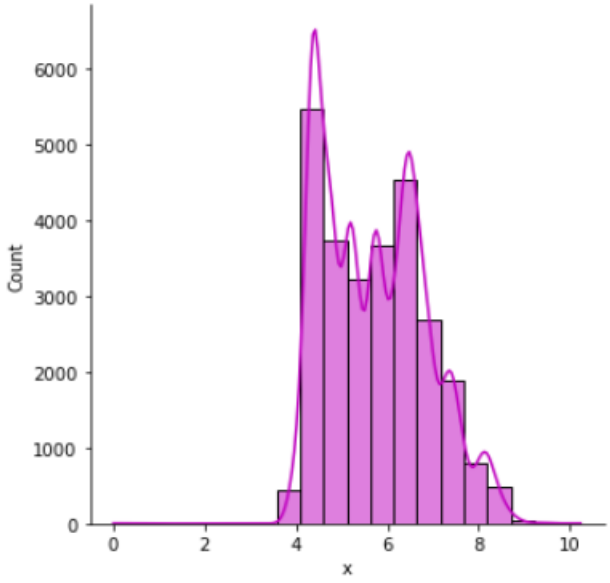


BoxPlot of table

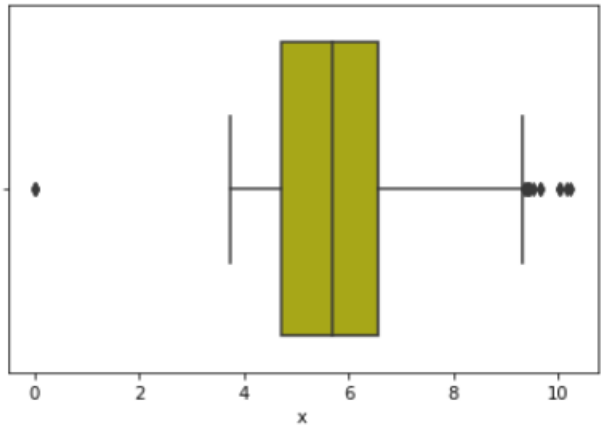



```
Description of x
------------------------
count    26967.000000
mean         5.729854
std          1.128516
min          0.000000
25%          4.710000
50%          5.690000
75%          6.550000
max         10.230000
Name: x, dtype: float64
```

Distribution of x

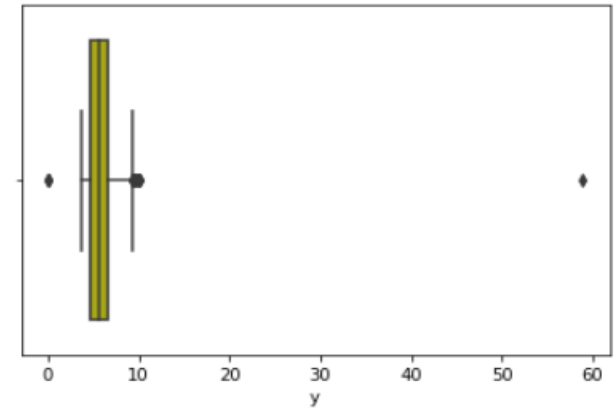


BoxPlot of x

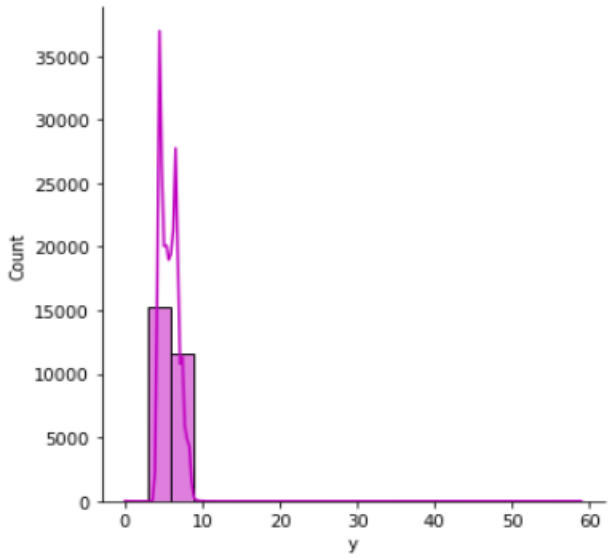


```
Description of y
------------------------
count    26967.000000
mean         5.733569
std          1.166058
min          0.000000
25%          4.710000
50%          5.710000
75%          6.540000
max         58.900000
Name: y, dtype: float64
```

BoxPlot of y

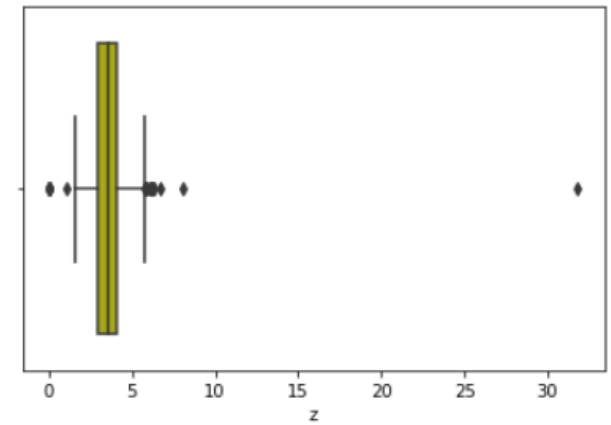


Distribution of y

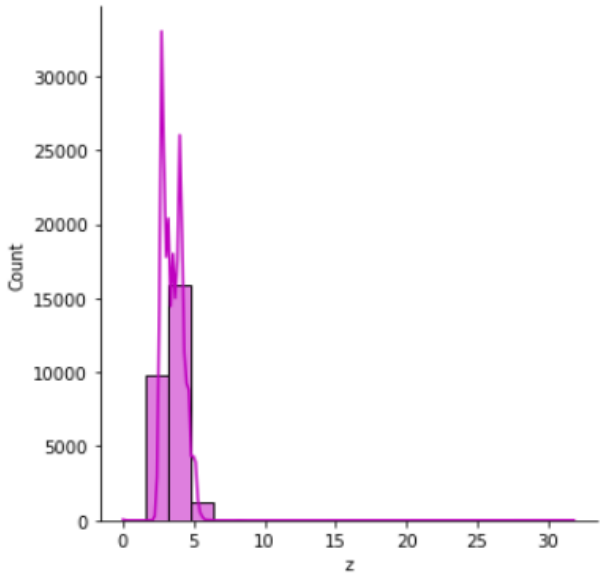



```
Description of z
------------------------
count    26967.000000
mean         3.538057
std          0.720624
min          0.000000
25%          2.900000
50%          3.520000
75%          4.040000
max         31.800000
Name: z, dtype: float64
```

BoxPlot of z

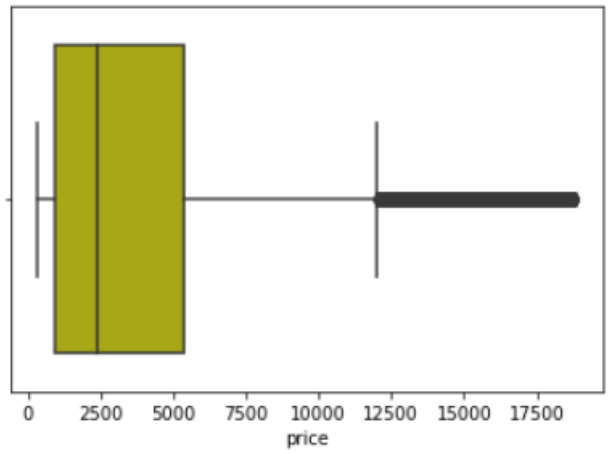


Distribution of z



```
Description of price
-----------------------------
count    26967.000000
mean      3939.518115
std       4024.864666
min        326.000000
25%        945.000000
50%       2375.000000
75%       5360.000000
max      18818.000000
Name: price, dtype: float64
```

BoxPlot of price



Distribution of price

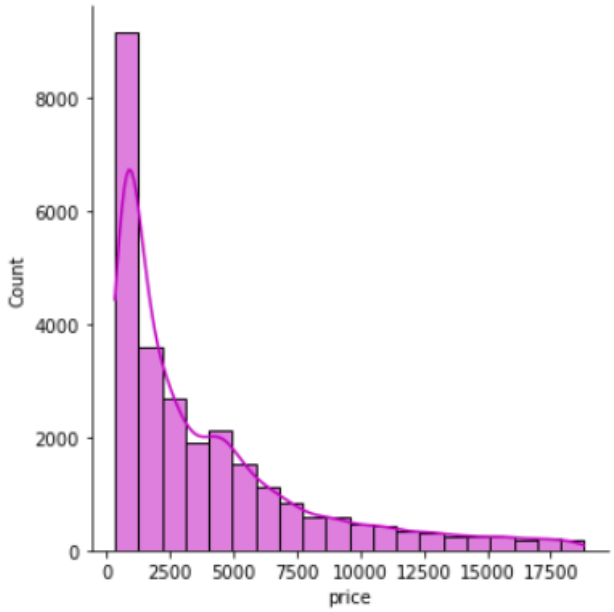


Figure 1: Univariate Analysis of Numeric Data(Problem 1)

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

We can see:

- the general description of all features
- none of the features are even nearly-normal, except for 'depth' whose distribution resembles a bell-curve, though not normal
- 'carat' and 'price' are highly right-skewed
- except for 'carat' and 'price' which have outliers to the right, all other features have outliers on both the sides
- 'y' and 'z' have an outlier each at the extreme right
- for right/positive skewness: MEAN>MEDIAN>MODE
- for left/negative skewness: MODE>MEDIAN>MEAN

Let us now go for the Univariate analysis of 'object' data features in the data set. We will check the unique values, count of unique values and proportion(i.e, normalized values) of unique values in each feature:

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

 CUT (normalized)
--------------------
Ideal         0.401083
Premium       0.255831
Very Good     0.223607
Good          0.090518
Fair          0.028961
Name: cut, dtype: float64
```

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

 CLARITY (normalized)
--------------------
SI1      0.243668
VS2      0.226165
SI2      0.169652
VS1      0.151778
VVS2     0.093855
VVS1     0.068194
IF       0.033152
I1       0.013535
Name: clarity, dtype: float64
```

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

 COLOR (normalized)
--------------------
G    0.209923
E    0.182334
F    0.175362
H    0.152112
D    0.124003
I    0.102755
J    0.053510
Name: color, dtype: float64
```

Tables 6: Univariate Analysis of Categorical Data(Problem 1)

(above tables generated with '.value_counts()' function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

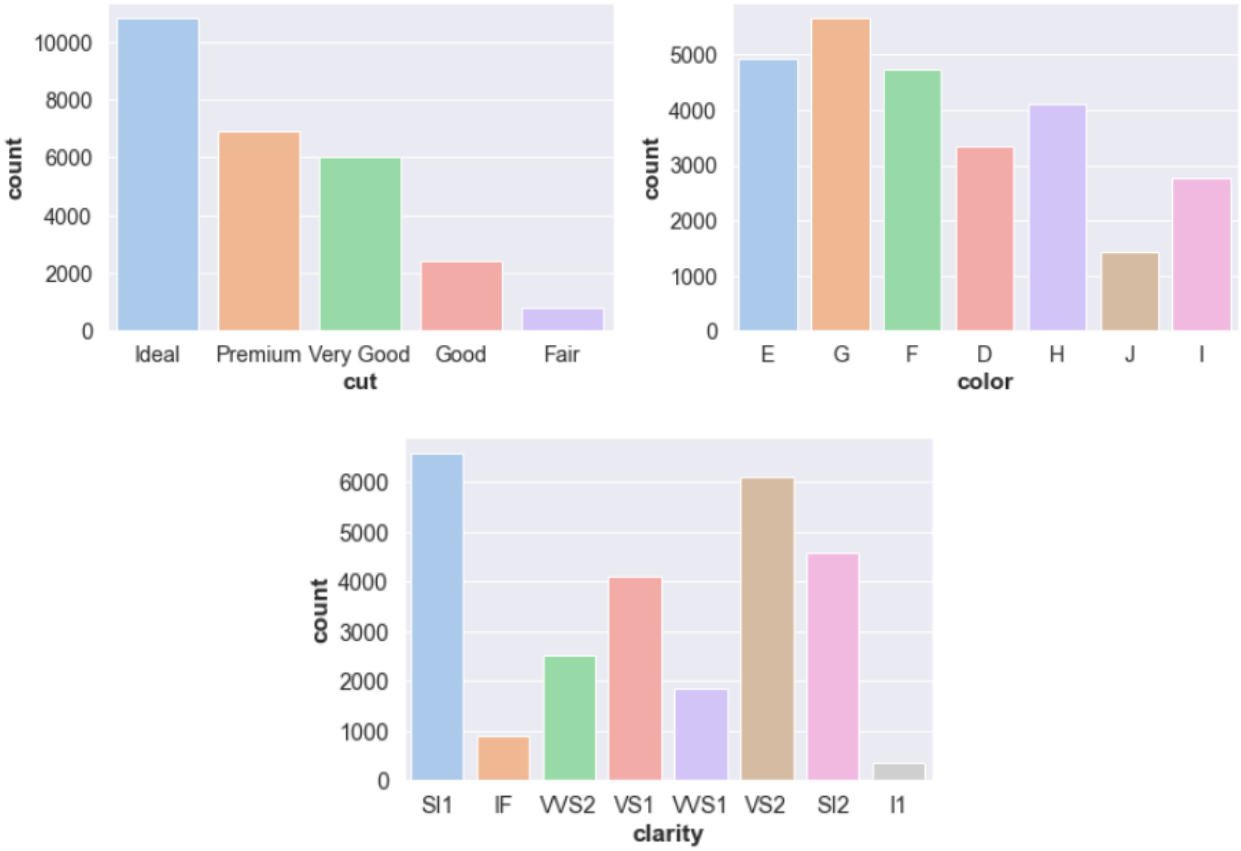


Figure 2: Count plots of Categorical Data(Problem 1)

(above plots generated using '.countplot()' function of Seaborn library in Python)

We can see:

- from the description, all the classes in the features seem to be correct without any anomalies
- 'Ideal' class outnumbers all other classes in 'cut' feature, 'Fair' class having the least samples
- features 'color' and 'clarity' have many sub-classes indicating a need for merging a few

Let us go for bivariate and multivariate analysis now.

To check the correlations amongst numerical features, let us have a look at the Pearson's Correlation coefficients in the form of a heatmap:

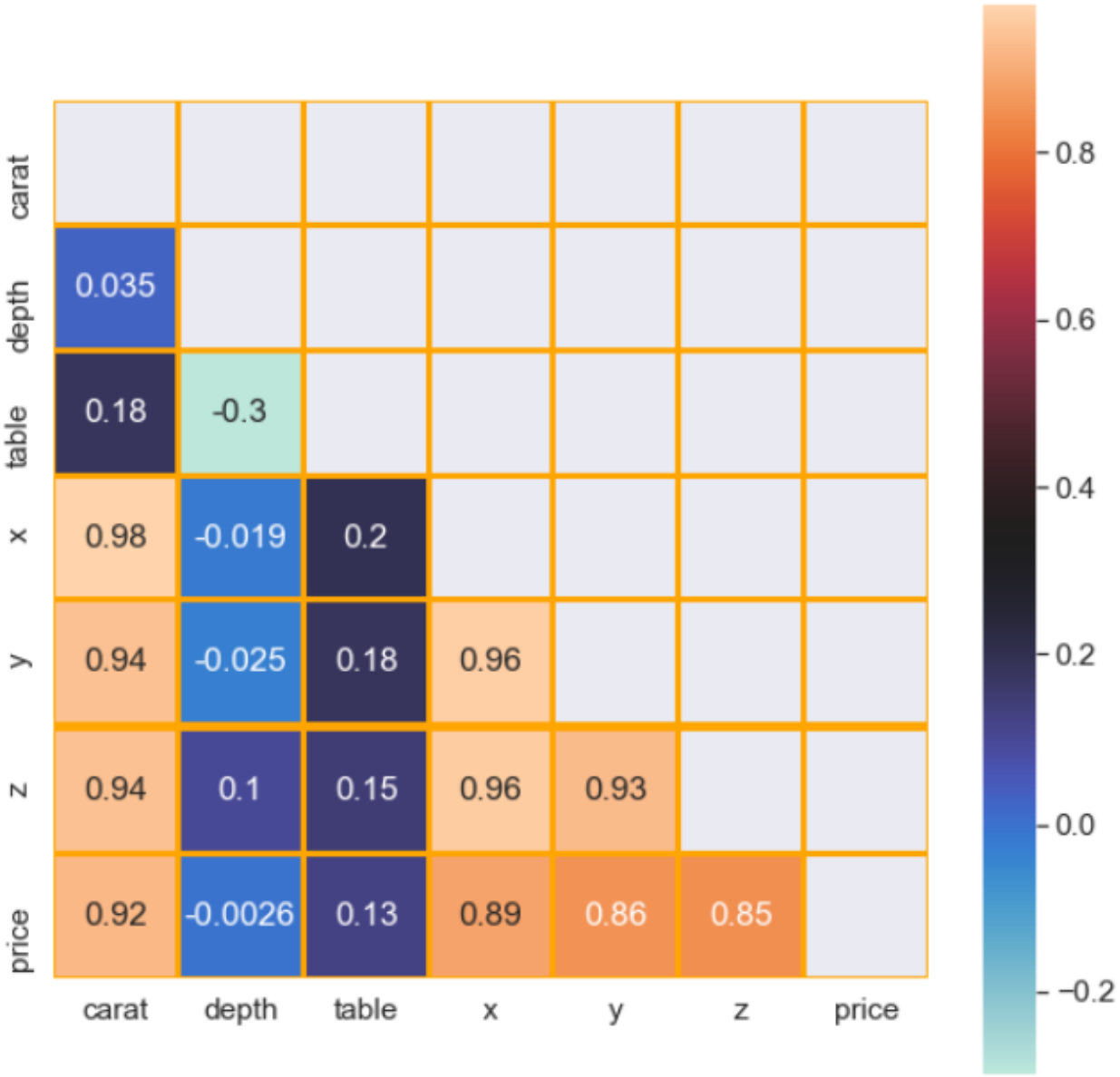


Figure 3: Correlation Heatmap(Problem 1)

(above plot generated using ‘:heatmap()’ function of Seaborn library in Python)

- Pearson’s Correlation coefficients near to 1 or -1 are highly positively correlated and highly negatively correlated respectively. Correlation values near to 0 are not or minimally correlated to each other.
- We see very high positive correlations amongst a few variables
- There are no high negative correlations
- ‘x’, ‘y’, ‘z’, ‘price’ and ’carat’ are very highly correlated to each other as per Pearson’s Correlation coefficients

Now, let us view the pairplot of the numeric features to confirm our observations of correlation heatmap:

(zoom to at least 150% for better visibility)

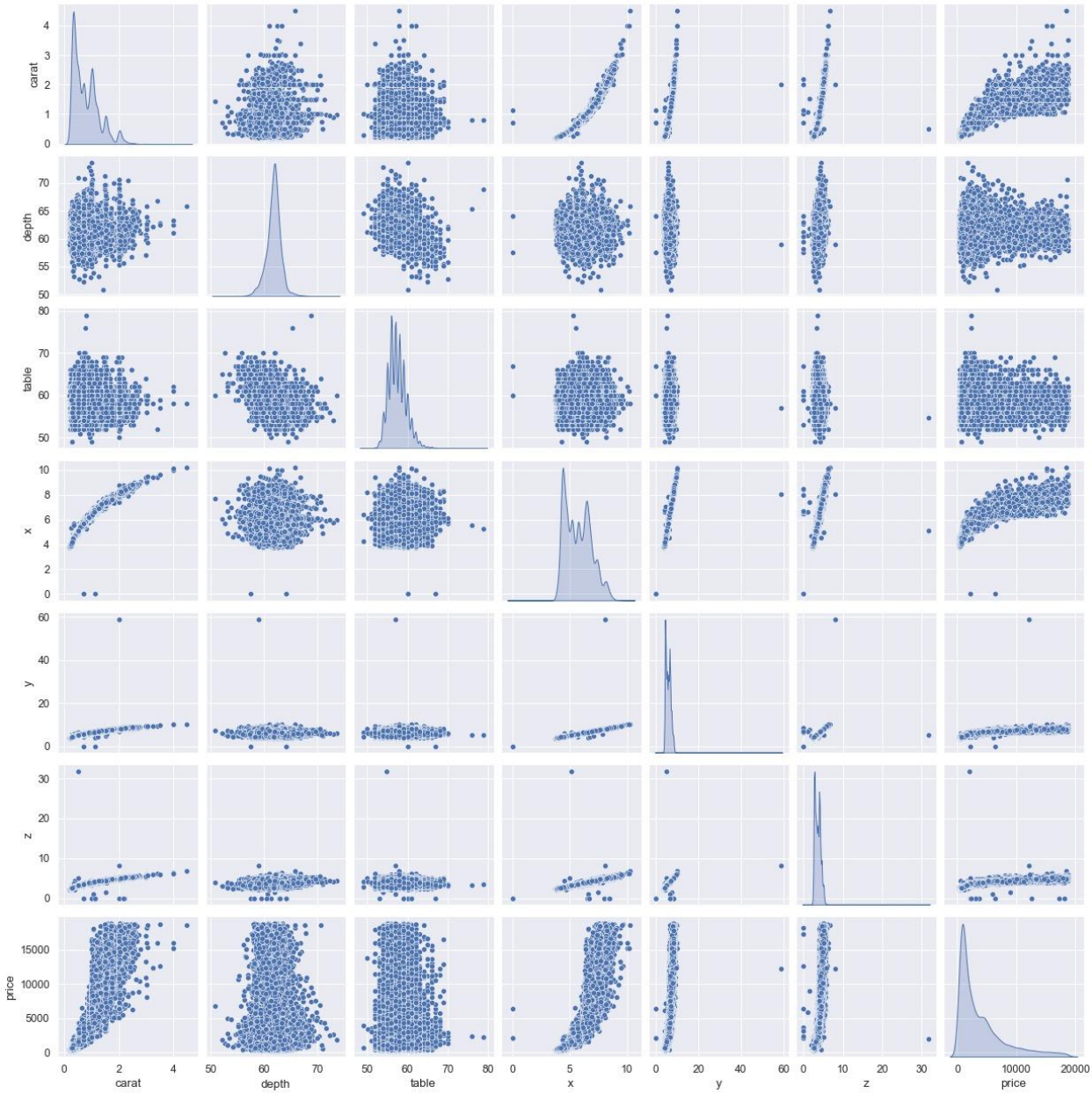


Figure 4: Pairplot(Problem 1)

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Pairplot shows the relationship between the numeric variables in the form of scatterplot and the distribution of the variables in the form of kernel density estimates.

The pattern of clouds of points confirm the high positive relationships amongst the features ‘x’, ‘y’, ‘z’, ‘price’ and ’carat’.

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? Check for the possibility of combining the sub levels of ordinal variables and take actions accordingly. Explain why you are combining these sublevels with appropriate reasoning.

- Using cascaded functions ‘.isnull().sum()’, we see feature depth has 697 null values. In a data set of 26967 samples, 697 null values do not have a significant effect. But, still let us have a look at the distribution of ‘depth’ feature to decide on the imputation strategy:

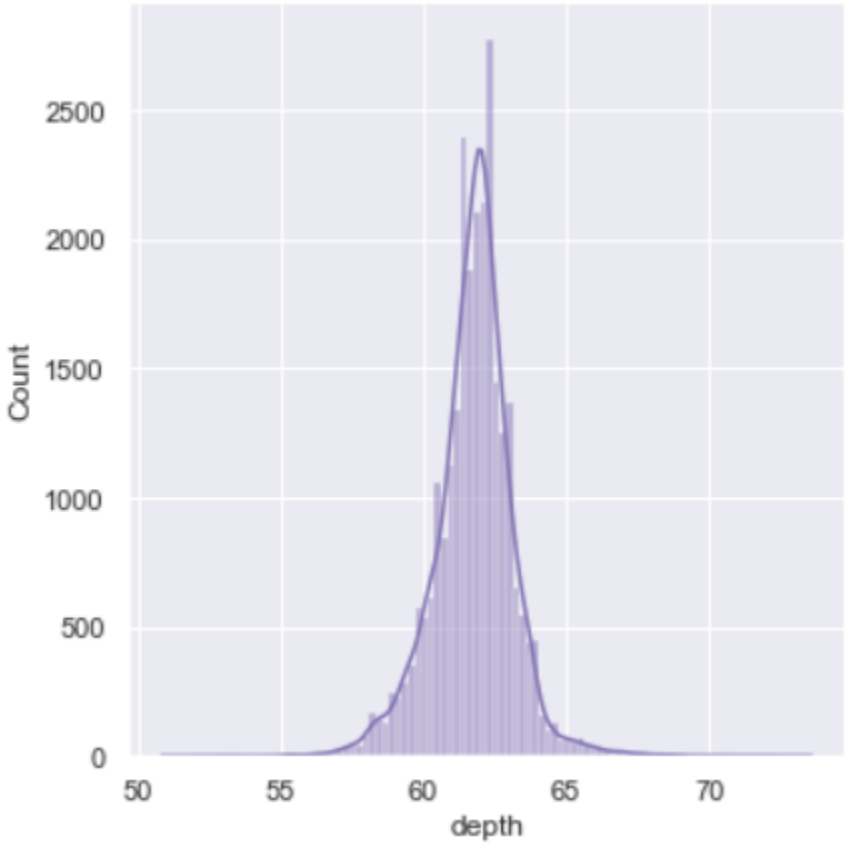


Figure 5: Distribution of feature ‘depth’

(above plot generated with ‘displot()’ function of ‘Seaborn’ library on Python)

As the distribution is somewhat of a bell shape symmetric on both sides, we can go ahead with replacing the null values in the ‘depth’ feature with the mean of the ‘depth’ feature. ‘.replace()’ function of PANDAS library has been used for the same.

- While checking for zeroes in the dataframe, we see the features ‘x’, ‘y’ and ‘z’ have zeroes in them at indices:

x: 5821, 6215, 17506

y: 5821, 6215, 17506

z: 5821, 6034, 6215, 10827, 12498, 12689, 17506, 18194, 23758

Zero values in above features isn't accurate, as 'x', 'y' and 'z' are the length, breadth and height respectively of the stones and they can't be 0. Let us check their distributions:

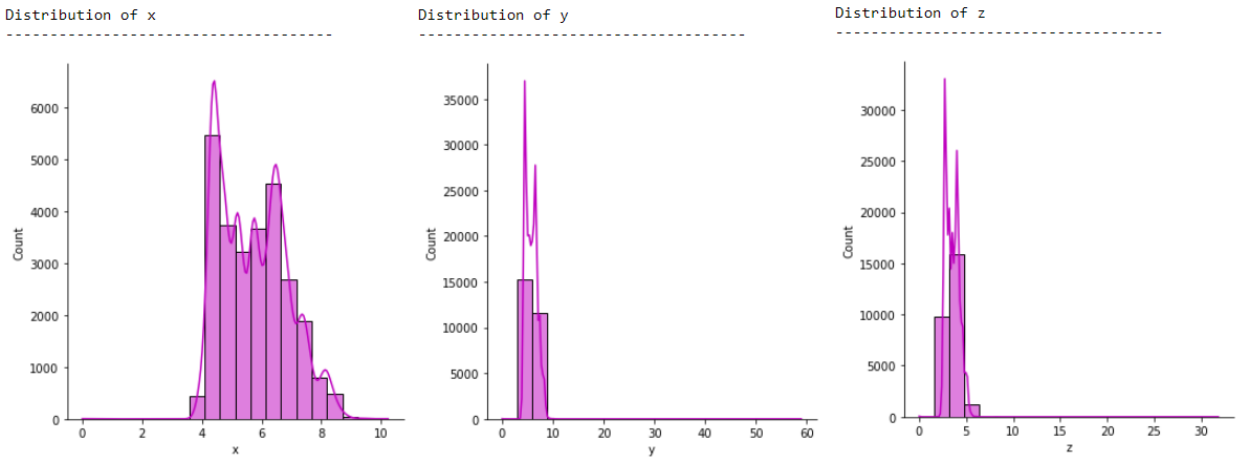


Figure 6: Distribution of features 'x', 'y' and 'z'

(above plots generated with 'displot()' function of 'Seaborn' library on Python)

The distribution for each of the 3 features is jagged, hence it would not be a good idea to impute the zeros with any of the measures of central tendencies. As the number of zeroes are quite less compared to the data size, we can go ahead and just drop them with the '.drop()' function of the PANDAS library in Python.

- To check the possibility of combining the ordinal levels, let us inspect the count and normalised values of classes in the ordinal features:

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

 CUT (normalized)
--------------------
Ideal         0.401083
Premium       0.255831
Very Good     0.223607
Good          0.090518
Fair          0.028961
Name: cut, dtype: float64
```

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

 CLARITY (normalized)
--------------------
SI1      0.243668
VS2      0.226165
SI2      0.169652
VS1      0.151778
VVS2     0.093855
VVS1     0.068194
IF       0.033152
I1       0.013535
Name: clarity, dtype: float64
```

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

 COLOR (normalized)
--------------------
G    0.209923
E    0.182334
F    0.175362
H    0.152112
D    0.124003
I    0.102755
J    0.053510
Name: color, dtype: float64
```

Tables 7: Univariate Analysis of Categorical Data(Problem 1)

(above tables generated with '.value_counts()' function of PANDAS library using Python)

- In the feature 'cut', class 'Fair' is only 2.896% of the total, so we can go ahead and merge class 'Fair' into the nearest class in order, 'Good'.

- ❖ In the feature ‘clarity’, class ‘I1’ is only 1.35% of the total, so we can go ahead and merge class ‘I1’ into the nearest class in order, ‘SI2’. Similarly, class ‘IF’ is only 3.32% of the total, so we can go ahead and merge class ‘IF’ into the nearest class in order, ‘VVS1’.
- ❖ In the feature ‘color’, class ‘J’ is only 5..35% of the total, so we can go ahead and merge class ‘J’ into the nearest class in order, ‘I’.

Let us inspect the count and normalised values of classes in the ordinal features after merging:

```
CUT :  4 unique values
--------------------
Ideal        10816
Premium       6893
Very Good     6030
Good          3219
Name: cut, dtype: int64

 CUT (normalized)
--------------------
Ideal        0.401217
Premium      0.255694
Very Good    0.223681
Good         0.119408
Name: cut, dtype: float64
```

```
CLARITY :  6 unique values
--------------------
SI1     6570
VS2     6098
SI2     4934
VS1     4092
VVS1    2733
VVS2    2531
Name: clarity, dtype: int64

 CLARITY (normalized)
--------------------
SI1     0.243712
VS2     0.226204
SI2     0.183025
VS1     0.151792
VVS1    0.101380
VVS2    0.093887
Name: clarity, dtype: float64
```

```
COLOR :  6 unique values
--------------------
G    5658
E    4917
F    4727
I    4214
H    4098
D    3344
Name: color, dtype: int64

 COLOR (normalized)
--------------------
G    0.209882
E    0.182395
F    0.175347
I    0.156317
H    0.152014
D    0.124045
Name: color, dtype: float64
```

The classes have been merged as desired.

1.3 Encode the data (having string values) for Modelling. Split the data into train and test (70:30). Apply Linear regression using scikit learn. Perform checks for significant variables

using appropriate methods from statsmodel. Create multiple models and check the performance of Predictions on Train and Test sets using R Square, RMSE & Adj R Square. Compare these models and select the best one with appropriate reasoning.

- Encoding has been done with cascaded functions '.str.replace()' in Python. All the classes in the ordinal features have been encoded as per rank or hierarchy:
  - ❖ 'cut' feature:

    Good: 0, Very Good: 1, Premium: 2, Ideal: 3

  - ❖ 'color' feature:

    D: 0, E: 1, F: 2, G: 3, H: 4, I:5

  - ❖ 'clarity' feature:

    'VVS1':0, 'VVS2':1, 'VS1':2, 'VS2':3, 'SI1':4, 'SI2':5

- Further, these category objects were converted into data-type integer(8 bits) with the '.astype()' function of the PANDAS library in Python.

Let us check the general information again after above data preparation:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26958 entries, 0 to 26966
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    26958 non-null  float64
 1   cut      26958 non-null  int8
 2   color    26958 non-null  int8
 3   clarity  26958 non-null  int8
 4   depth    26261 non-null  float64
 5   table    26958 non-null  float64
 6   x        26958 non-null  float64
 7   y        26958 non-null  float64
 8   z        26958 non-null  float64
 9   price    26958 non-null  int64
dtypes: float64(6), int64(1), int8(3)
memory usage: 1.7 MB
```

Table 8: General Information of Prepared Data(Problem1)

(above table generated by '.info' of 'PANDAS' in Python)

- We are not treating the outliers as evident from the EDA's distributions(KDEs and boxplots), we see a significant number of outliers, and treating such a large number of outliers may affect the model's performance in a real production environment.
- 'price' being the continuous target variable is separated from the other predictor features for model building(for that of scikit learn).

- ‘train_test_split’ from ‘sklearn.model_selection’ has been used to split the data in the ratio of 70:30 (train : test), by providing test_size as 0.30. ‘random state’ of 1 has been provided.

Model 1

- We use the ‘LinearRegression’ regressor from the ‘linear_model’ sub-module of ‘sklearn’ module to build our linear regression model.
  - ❖ Following were the coefficients obtained:

    carat : 10981.36

    cut : 114.13

    color : -314.08

    clarity : -437.85

    depth : -111.13

    table : -40.71

    x : -996.55

    y : 10.91

    z : -54.17
  - ❖ The intercept for this model is 12192.55
  - ❖ R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable.

    R^2(train): 0.9012 R^2(test): 0.8940

    90.12% of the variation in the price is explained by the predictors in the model for the train set, while 89.40% of the variation in the price is explained by the predictors in the model for the test set.
  - ❖ The RMSE is the square root of the mean of the squared error values between observations and predictions. In linear regression, the cost function (which tells us what the error is for each combination of coefficients and intercepts) can be visualized as a convex (bowl shaped) shaped function.

    RMSE(train): 1252.91 RMSE(test):1337.45

- To get the other metrics as we get from R language, we can build the model again using ‘statsmodels’.
  - ❖ ‘.concat()’ function of the PANDAS library is being used to concatenate the independent and the target data.

- ‘.ols()’ from ‘statsmodels.formula.api’ is being used to formulate the model.
- ‘.params’ of the model gives us the intercept and coefficients values:

  Intercept 12192.55
  carat 10981.36
  cut 114.13
  color -314.08
  clarity -437.85
  depth -111.13
  table -40.71
  x -996.55
  y 10.91
  z -54.17

- Let us now have a look at the OLS Regression Results given by ‘.summary()’ of the model:

OLS Regression Results

Dep. Variable:	price	R-squared:	0.901
Model:	OLS	Adj. R-squared:	0.901
Method:	Least Squares	F-statistic:	1.911e+04
Date:	Sun, 24 Oct 2021	Prob (F-statistic):	0.00
Time:	11:55:15	Log-Likelihood:	-1.6138e+05
No. Observations:	18870	AIC:	3.228e+05
Df Residuals:	18860	BIC:	3.229e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.219e+04	722.393	16.878	0.000	1.08e+04	1.36e+04
carat	1.098e+04	95.709	114.737	0.000	1.08e+04	1.12e+04
cut	114.1319	10.450	10.921	0.000	93.648	134.616
color	-314.0832	5.982	-52.509	0.000	-325.808	-302.359
clarity	-437.8542	6.182	-70.825	0.000	-449.972	-425.737
depth	-111.1277	8.084	-13.747	0.000	-126.973	-95.283
table	-40.7074	5.124	-7.945	0.000	-50.750	-30.664
x	-996.5514	52.058	-19.143	0.000	-1098.589	-894.514
y	10.9105	24.552	0.444	0.657	-37.214	59.035
z	-54.1747	43.036	-1.259	0.208	-138.530	30.180

Omnibus:	4184.220	Durbin-Watson:	1.979
Prob(Omnibus):	0.000	Jarque-Bera (JB):	202531.402
Skew:	-0.046	Prob(JB):	0.00
Kurtosis:	19.049	Cond. No.	6.74e+03

Table 9: OLS Regression Results - Model 1

(Above table generated with ‘statsmodels’ library in Python)

- R^2 is the coefficient of determination. A statistical measure of how well the regression line approximates the real data points. It is 0.901

- Adjusted R^2 is R^2 value adjusted based on the number of observations and the degrees-of-freedom of the residuals. It excludes the fluke relations. It's also 0.901

Other Statistics

- Prob (F-statistic): The probability that you would get the F-statistic, given the null hypothesis that they are unrelated
- t-statistic value: This is a measure of how statistically significant the coefficient is.
- Durbin-Watson: A test for the presence of autocorrelation (that the errors are not independent.) Often important in time-series analysis
- Cond. No: The t-statistic value. This is a measure of how statistically significant the coefficient is.

Let us have a look at a scatter of the predicted vs. actual prices:

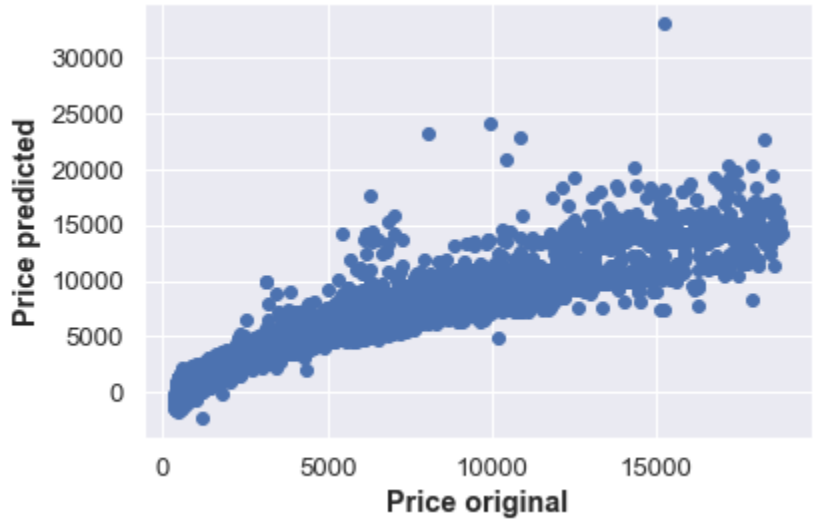


Figure 7: Predicted vs. Actual Plot - Model 1

(above figure generated with 'scatter' function of 'matplotlib' library in Python)

Price = (12192.55) + (10981.36) * carat + (114.13) * cut + (-314.08) * color + (-437.85) * clarity + (-111.13) * depth + (-40.71) * table + (-996.55) * x + (10.91) * y + (-54.17) * z

Assuming a confidence level of 95%, we see the probability values for features 'y' and 'z' are 0.657 and 0.208, which is quite higher than 0.05. So, the interpretability of the model at feature level becomes a bit inconvenient. Also, the condition number is large, 6.74e+03. This might indicate that there are strong multicollinearity or other numerical problems. These have been taken care of in the next model.

To confirm the above, let us also check the variance inflation factors of the features generated using 'variance_inflation_factor' of 'statsmodels.stats.outliers_influence':

carat : 81.267
cut : 5.031
color : 3.911
clarity : 5.840

depth : 550.244
table : 553.160
x : 1132.508
y : 348.218
z : 385.930

A VIF between 5 and 10 indicates high correlation that may be problematic. And if the VIF goes above 10, one can assume that the regression coefficients are poorly estimated due to multicollinearity.

Model 2

Let us revisit the correlation heatmap:

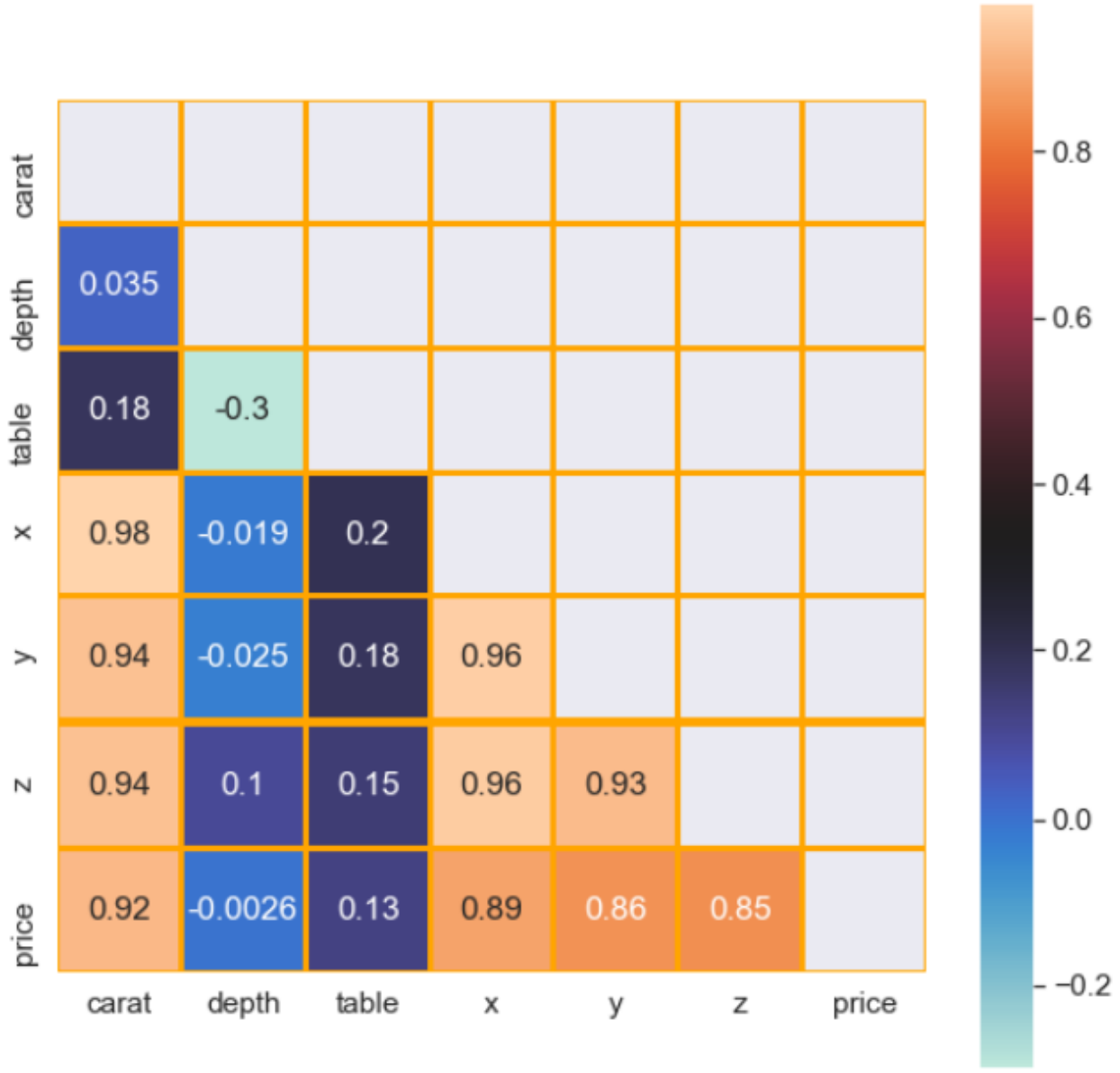


Figure 8: Correlation Heatmap(Problem 1)

(above plot generated using '.heatmap()' function of Seaborn library in Python)

We can see that 'x', 'y', 'z', 'carat' and 'price' are highly correlated to each other(all greater than 0.9). Theoretically also, carat is determined by the dimensions, and accordingly, the price is set. So, we can just take the feature 'carat' and exclude 'x', 'y', and 'z' from the training set. 'price' is the target.

- We use the 'LinearRegression' regressor from the 'linear_model' sub-module of 'sklearn' module to build our linear regression model.
  - Following were the coefficients obtained:

    carat : 8643.74
    cut : 119.25
    color : -309.45
    clarity : -461.36
    depth : -68.7
    table : -39.31

  - The intercept for this model is 5571.31
  - R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable.

    R^2(train): 0.8979 R^2(test): 0.8910

    89.79% of the variation in the price is explained by the predictors in the model for the train set, while 89.10% of the variation in the price is explained by the predictors in the model for the test set.
  - The RMSE is the square root of the mean of the squared error values between observations and predictions. In linear regression, the cost function (which tells us what the error is for each combination of coefficients and intercepts) can be visualized as a convex (bowl shaped) shaped function.

    RMSE(train): 1273.69 RMSE(test):1356.44

- To get the other metrics as we get from R language, we can build the model again using 'statsmodels'.
  - '.concat()' function of the PANDAS library is being used to concatenate the independent and the target data.
  - '.ols()' from 'statsmodels.formula.api' is being used to formulate the model.
  - '.params' of the model gives us the intercept and coefficients values:
    Intercept 5571.31
    carat 8643.74
    cut 119.25

color	-309.45
clarity	-461.36
depth	-68.70
table	-39.31

- Let us now have a look at the OLS Regression Results given by '.summary()' of the model:

OLS Regression Results

Dep. Variable:	price	R-squared:	0.898
Model:	OLS	Adj. R-squared:	0.898
Method:	Least Squares	F-statistic:	2.765e+04
Date:	Sun, 24 Oct 2021	Prob (F-statistic):	0.00
Time:	17:48:30	Log-Likelihood:	-1.6169e+05
No. Observations:	18870	AIC:	3.234e+05
Df Residuals:	18863	BIC:	3.234e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5571.3066	664.851	8.380	0.000	4268.139	6874.475
carat	8643.7419	22.026	392.425	0.000	8600.568	8686.916
cut	119.2512	10.613	11.236	0.000	98.448	140.054
color	-309.4504	6.077	-50.920	0.000	-321.362	-297.538
clarity	-461.3572	6.211	-74.282	0.000	-473.531	-449.183
depth	-68.7031	7.612	-9.026	0.000	-83.623	-53.784
table	-39.3135	5.201	-7.558	0.000	-49.509	-29.118

Omnibus:	3776.315	Durbin-Watson:	1.982
Prob(Omnibus):	0.000	Jarque-Bera (JB):	66671.647
Skew:	0.480	Prob(JB):	0.00
Kurtosis:	12.158	Cond. No.	6.06e+03

Table 10: OLS Regression Results - Model 2

(Above table generated with 'statsmodels' library in Python)

- R^2 is the coefficient of determination. A statistical measure of how well the regression line approximates the real data points. It is 0.898
- Adjusted R^2 is R^2 value adjusted based on the number of observations and the degrees-of-freedom of the residuals. It excludes the fluke relations. It's also 0.898

Other Statistics

- Prob (F-statistic): The probability that you would get the F-statistic, given the null hypothesis that they are unrelated
- t-statistic value: This is a measure of how statistically significant the coefficient is.
- Durbin-Watson: A test for the presence of autocorrelation (that the errors are not independent.) Often important in time-series analysis

❖ Cond. No: The t-statistic value. This is a measure of how statistically significant the coefficient is.

Let us have a look at a scatter of the predicted vs. actual prices:

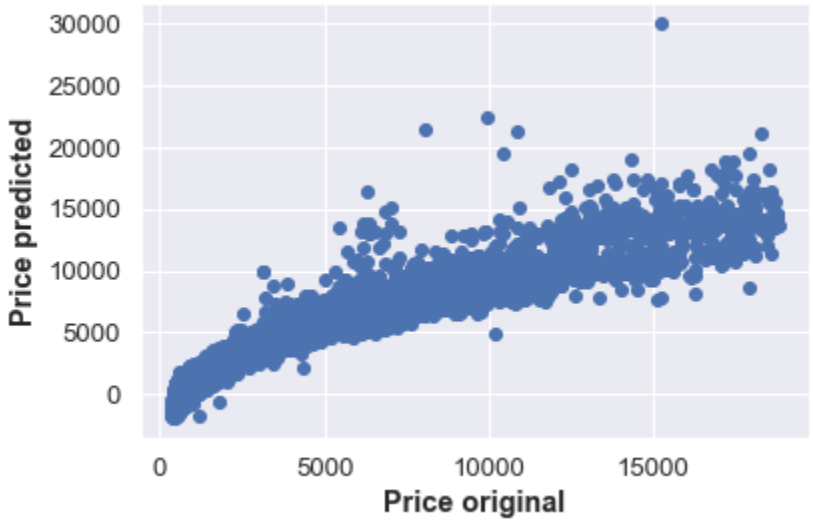


Figure 9: Predicted vs. Actual Plot - Model 2

(above figure generated with 'scatter' function of 'matplotlib' library in Python)

The p-values at individual feature level is 0.00, so the model is good to interpret the coefficients. We can say:

Price = 5571.31 + (8643.74) * carat + (119.25) * cut + (-309.45) * color + (-461.36) * clarity + (-68.7) * depth + (-39.31) * table

To confirm the above, let us also check the variance inflation factors of the features generated using 'variance_inflation_factor' of 'statsmodels.stats.outliers_influence':

carat : 4.814
cut : 4.689
color : 3.907
clarity : 5.736
depth : 443.612
table : 426.035

A VIF between 5 and 10 indicates high correlation that may be problematic. And if the VIF goes above 10, one can assume that the regression coefficients are poorly estimated due to multicollinearity.

We still see VIF values for features 'depth' and 'table' to be high though the Pearson's correlation coefficient for them is low, indicating a non-linear relationship between the features.

<u>Model 3</u>

- We choose to take 'table' feature amongst 'table' and 'depth' features as these two features are correlated some way as evident from the VIF values in above model.

- We use the 'LinearRegression' regressor from the 'linear_model' sub-module of 'sklearn' module to build our linear regression model.
    - ❖ Following were the coefficients obtained:

      carat : 8635.09
      cut : 153.88
      color : -312.25
      clarity : -464.64
      table : -18.93

    - ❖ The intercept for this model is 115.08.
    - ❖ R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable.

      R^2(train): 0.8975 R^2(test): 0.8908

      89.75% of the variation in the price is explained by the predictors in the model for the train set, while 89.08% of the variation in the price is explained by the predictors in the model for the test set.
    - ❖ The RMSE is the square root of the mean of the squared error values between observations and predictions. In linear regression, the cost function (which tells us what the error is for each combination of coefficients and intercepts) can be visualized as a convex (bowl shaped) shaped function.

      RMSE(train): 1276.44 RMSE(test):1357.85

- To get the other metrics as we get from R language, we can build the model again using 'statsmodels'.
    - ❖ '.concat()' function of the PANDAS library is being used to concatenate the independent and the target data.
    - ❖ '.ols()' from 'statsmodels.formula.api' is being used to formulate the model.
    - ❖ '.params' of the model gives us the intercept and coefficients values:
      Intercept 115.08
      carat 8635.09
      cut 153.88
      color -312.25
      clarity -464.64
      table -18.93

- Let us now have a look at the OLS Regression Results given by '.summary()' of the model:

OLS Regression Results

Dep. Variable:	price	R-squared:	0.897
Model:	OLS	Adj. R-squared:	0.897
Method:	Least Squares	F-statistic:	3.302e+04
Date:	Sun, 31 Oct 2021	Prob (F-statistic):	0.00
Time:	15:15:53	Log-Likelihood:	-1.6173e+05
No. Observations:	18870	AIC:	3.235e+05
Df Residuals:	18864	BIC:	3.235e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	115.0802	277.374	0.415	0.678	-428.598	658.759
carat	8635.0915	22.052	391.570	0.000	8591.867	8678.316
cut	153.8775	9.917	15.517	0.000	134.440	173.315
color	-312.2455	6.082	-51.337	0.000	-324.167	-300.324
clarity	-464.6373	6.213	-74.780	0.000	-476.816	-452.458
table	-18.9259	4.695	-4.031	0.000	-28.129	-9.722

Omnibus:	3768.445	Durbin-Watson:	1.981
Prob(Omnibus):	0.000	Jarque-Bera (JB):	68886.547
Skew:	0.464	Prob(JB):	0.00
Kurtosis:	12.314	Cond. No.	1.72e+03

Table 11: OLS Regression Results - Model 3

(Above table generated with 'statsmodels' library in Python)

- R^2 is the coefficient of determination. A statistical measure of how well the regression line approximates the real data points. It is 0.897
- Adjusted R^2 is R^2 value adjusted based on the number of observations and the degrees-of-freedom of the residuals. It excludes the fluke relations. It's also 0.897

Other Statistics

- Prob (F-statistic): The probability that you would get the F-statistic, given the null hypothesis that they are unrelated
- t-statistic value: This is a measure of how statistically significant the coefficient is.
- Durbin-Watson: A test for the presence of autocorrelation (that the errors are not independent.) Often important in time-series analysis
- Cond. No: The t-statistic value. This is a measure of how statistically significant the coefficient is.

Let us have a look at a scatter of the predicted vs. actual prices:

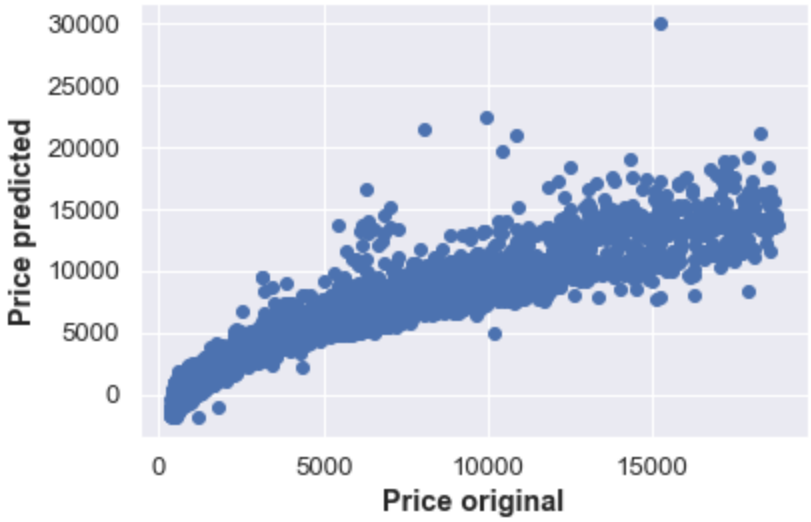


Figure 10: Predicted vs. Actual Plot - Model 3
(above figure generated with 'scatter' function of 'matplotlib' library in Python)

The p-values at individual feature level is 0.00, so the model is good to interpret the coefficients. We can say:
Price = 115.08 + (8635.09)*carat + (153.88)*cut + (-312.25)*color + (-464.64)*clarity + (-18.93)*table

To confirm the above, let us also check the variance inflation factors of the features generated using 'variance_inflation_factor' of 'statsmodels.stats.outliers_influence':
carat : 4.793
cut : 4.230
color : 3.882
clarity : 5.723
table : 12.506

A VIF between 5 and 10 indicates high correlation that may be problematic. And if the VIF goes above 10, one can assume that the regression coefficients are poorly estimated due to multicollinearity.
This model seems to be the best for analysing the coefficients as the VIF values are approximately within limits.

Model 4(Ridge Regularised)

- The same processed data is being used except for the K-Nearest Neighbour Imputation in the feature 'depth'.
- Polynomial features were created using 'PolynomialFeatures' of 'sklearn.preprocessing' with maximum degrees of polynomial features as 2.
- The regularisation parameter, α was set at 0.3.
- Following were the coefficients obtained for the 22 polynomial features:

0. 4376.51840949 131.47878627 -514.07125621 -900.14262287

-35.67471085 -60.79688816 73.53149913 -304.30418471 -508.05614341

-97.48944126 31.36456413 -17.28425548 13.68929271 58.21046778

47.63693777 196.9512232 -6.15528002 -25.44756435 8.92723295

-43.44483984 35.5947649

- R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable.

R^2(train): 0.917 R^2(test): 0.912

Model 5(Lasso Regularised)

- The same processed data is being used except for the K-Nearest Neighbour Imputation in the feature 'depth'.
- Polynomial features were created using 'PolynomialFeatures' of 'sklearn.preprocessing' with maximum degrees of polynomial features as 2.
- The regularisation parameter, α was set at 0.01.
- Following were the coefficients obtained for the 22 polynomial features:

0. 4376.60909627 131.48132917 -514.09183119 -900.17920489
-35.66144189 -60.79936526 73.51015324 -304.32087089 -508.07674458
-97.4976365 31.32908735 -17.25495725 13.68816237 58.20321208
47.62536314 196.94580289 -6.12871434 -25.40899194 8.9123998
-43.43371971 35.58576936

- R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable.

R^2(train): 0.917 R^2(test): 0.912

Conclusion

- If interpretability of the features' coefficients is a concern to us, then looking at Adjusted R^2, R^2, RMSE and VIFs of models 1,2 and 3, we can conclude model 3 is better for interpretability.
- As models 4 and 5 are using feature engineered polynomials too, interpreting the coefficients would be tedious, but either of them(both have similar train and test metrics) can be used for only predicting the price as both have better train and test accuracies than unregularised linear models 1, 2 and 3.

If we were to choose one, then we can go for the Ridge regularised linear regression model(model 4), as **ridge regression is faster compared to lasso**(but then again lasso has the advantage of completely reducing unnecessary parameters in the model).

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

Summary:

- We started with reading the data, general information and general description for numerical and categorical data.
- We did the exploratory data analysis(univariate, bivariate and multivariate) and drew insights.
- For the simple linear models(1 and 2), null values in the feature 'depth' were replaced with the mean values of the same.
- For the regularized models(3 and 4), the null values in the feature 'depth' were replaced by the K-Nearest Neighbour algorithm.
- Zero-valued samples were removed.
- Sub-levels in ordinal variables were merged that were less in number with the nearest sub-level in order.
- Categorical features were encoded.
- Data was split into train and test sets.
- Different models were built and analysed.

As in the conclusion in the previous question 1.3, we conclude:

- If interpretability of the features' coefficients is a concern to us, then looking at Adjusted R^2, R^2 and RMSE of models 1 and 2, we can conclude model 2 is better
- As models 3 and 4 are using feature engineered polynomials, interpreting the coefficients would be tedious, but either of them(both have similar train and test metrics) can be used for only predicting the price as both have better train and test accuracies than unregularised linear models 1 and 2.

Let us have a look at the linear regression equation of the model 2:

Price = 5571.31 + (8643.74) * carat + (119.25) * cut + (-309.45) * color + (-461.36) * clarity + (-68.7) * depth + (-39.31) * table

- From the coefficients we can see the 'carat' feature has the highest positive coefficient(direct relationship). When carat increases by 1 unit, price increases by 8643.74 units, keeping all other predictors constant.
- Also, we can see the 'clarity' feature has the highest negative coefficient(inverse relationship). When clarity increases by 1 unit, price decreases by 461.36 units, keeping all other predictors constant.
- Similarly, we interpret the other coefficients.
- We see the positive coefficient for 'carat'(8643.74) is quite higher than that of 'cut'(119.25), so we can say, in order to increase the price, carat needs to be taken care of. Cut of the stone is less material and doesn't really contribute significantly to the price, when compared with the feature carat.

- We see high negative coefficients for ‘color’ and ‘clarity’, indicating something unexpected generally. It means with increase in color and clarity of stone, the price reduces. It may be because of the reducing carat, which has a far greater coefficient.
- Features ‘depth’ and ‘table’ have very low coefficients indicating the height and width do not really characterise the price of the stone.

For prediction of prices we can use the regularised linear models. Both Lasso and Ridge models have shown similar metrics on both training and testing data.

Problem 2

Executive Summary

A tour and travel agency which deals in selling holiday packages has provided details of 872 employees of a company. Among these employees, some opted for the package and some didn't. The company seeks help in predicting whether an employee will opt for the package or not on the basis of the information given in the data set. Also, important factors need to be listed on the basis of which the company will focus on particular employees to sell their packages.

Introduction

The objective is to use Logistic Regression and Linear Discriminant analysis to predict if the employee will opt for a holiday package or not. We begin with exploratory data analysis to draw insights and provide recommendations too.

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.

Let us have a look at first and last few records of data:

	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

Table 12: Sample of Data(Problem 2) - 1

(above table generated with .head() function of PANDAS library in Python)

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
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

Table 13: Sample of Data(Problem 2) - 2

(above table generated with .tail() function of PANDAS library in Python)

Let us drop the 'Unnamed: 0' feature which seems to be just indices and proceed further.

Data Dictionary:

Variable Name	Description
Holliday_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

Exploratory Data Analysis

Let us check the data types of variables and the missing values(if any):

```
<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    int64
 2   age                872 non-null    int64
 3   educ               872 non-null    int64
 4   no_young_children  872 non-null    int64
 5   no_older_children  872 non-null    int64
 6   foreign            872 non-null    object
dtypes: int64(5), object(2)
memory usage: 47.8+ KB
```

Table 14: General Information of Data(Problem2)

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- shape of data is (872x7), after dropping 'Unnamed: 0' feature
- there are total 872 entries(rows) of stone samples in data indexed from 0 to 871
- there are 7 columns in total(Holliday_Package, Salary, age, educ, no_young_children, no_older_children, foreign)
- except for 'Holliday_Package' and 'foreign' which are having object data, all other variables are of data-type integer(64 bits)
- the memory usage by data is 47.8+ kb
- there is no null value in data set as all columns have 872 non null objects

'.isnull().sum().sum()' also confirms the null value count in the data set, yielding an output 0.

There are no duplicate entries that have been confirmed by '.duplicated()'.

(Heatmap and Pairplot provided after Univariate Analysis)

Let us now move to the general description of numeric data(minimum values, maximum values, measures of central tendencies like mean, median, mode etc.):

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode
Salary	47729.17	23418.67	1322.0	35324.0	41903.5	53469.5	236961.0	235639.0	18145.5	0.49	32197
age	39.96	10.55	20.0	32.0	39.0	48.0	62.0	42.0	16.0	0.26	44.0
educ	9.31	3.04	1.0	8.0	9.0	12.0	21.0	20.0	4.0	0.33	8.0
no_young_children	0.31	0.61	0.0	0.0	0.0	0.0	3.0	3.0	0.0	1.96	0.0
no_older_children	0.98	1.09	0.0	0.0	1.0	2.0	6.0	6.0	2.0	1.11	0.0

Table 15: General Description of Numeric Data(Problem 2)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- 'mean' and 'median' values are nearly equal for all features, indicating no wrong entries prima facie.
- '75%' and 'max' values seem to be optimally distant, again indicating no wrong entries prima facie.
- the coefficient of variation is quite high for 'no_older_children' (1.96)

Let us now move to the general description of 'object' data:

	count	unique	top	freq
Holliday_Package	872	2	no	471
foreign	872	2	no	656

Table 16: General Description of 'Object' type data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

The object data seems to be fine.

Let us now go for the Univariate analysis of numeric features in the data set. We will go for general descriptions, distribution plots with kernel density estimates and boxplots for all the features one by one:

(zoom to at least 125% for better visibility)

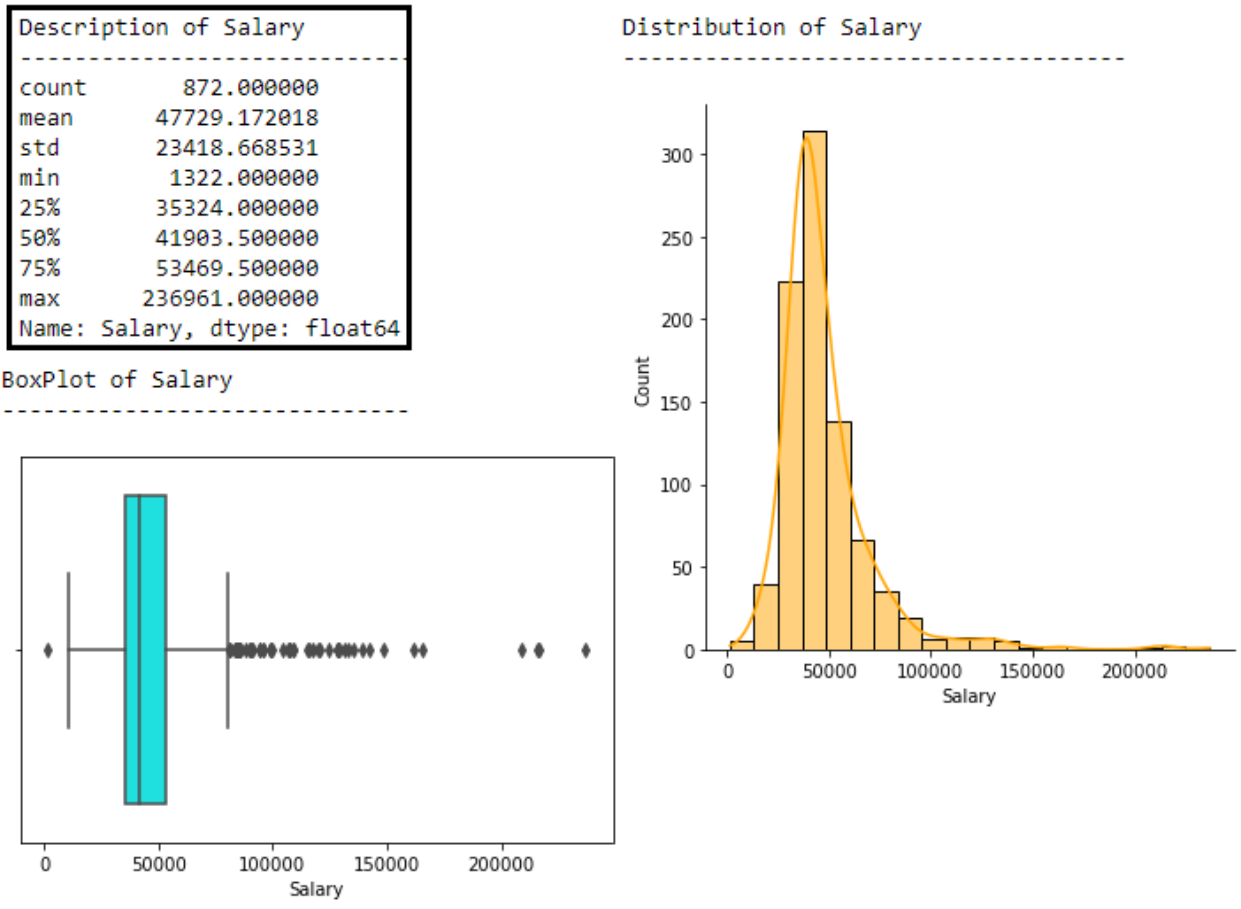


Figure 11: Univariate Analysis of 'Salary'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- the mean and median values are very close to each other in the feature 'Salary'
- the distribution resembles a bell shape with positive skewness
- there are more outliers to the right
- Due to positive skewness: MEAN>MEDIAN>MODE

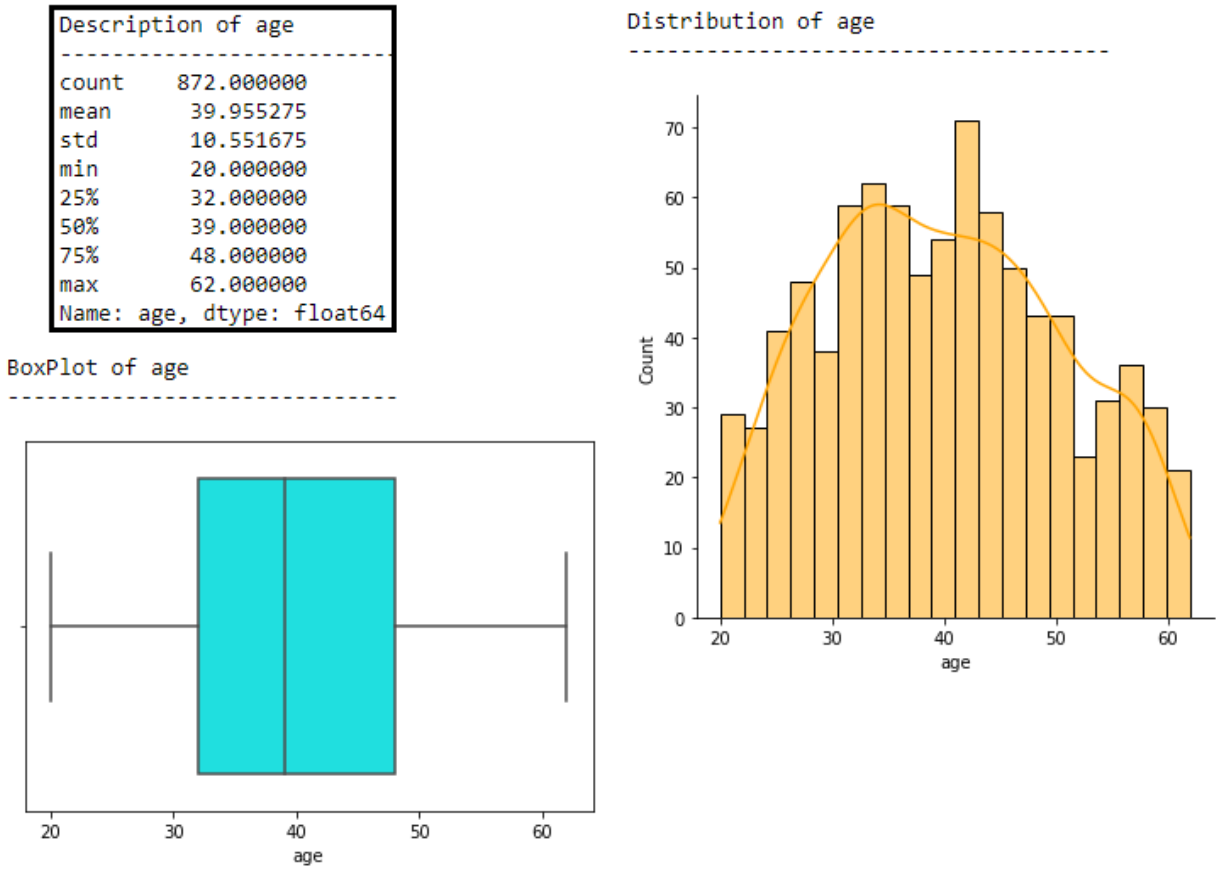


Figure 12: Univariate Analysis of 'age'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- the mean and median are different by 1 decimal place
- the distribution aint Gaussian

- the boxplot doesn't show any outlier, indicating no anomaly

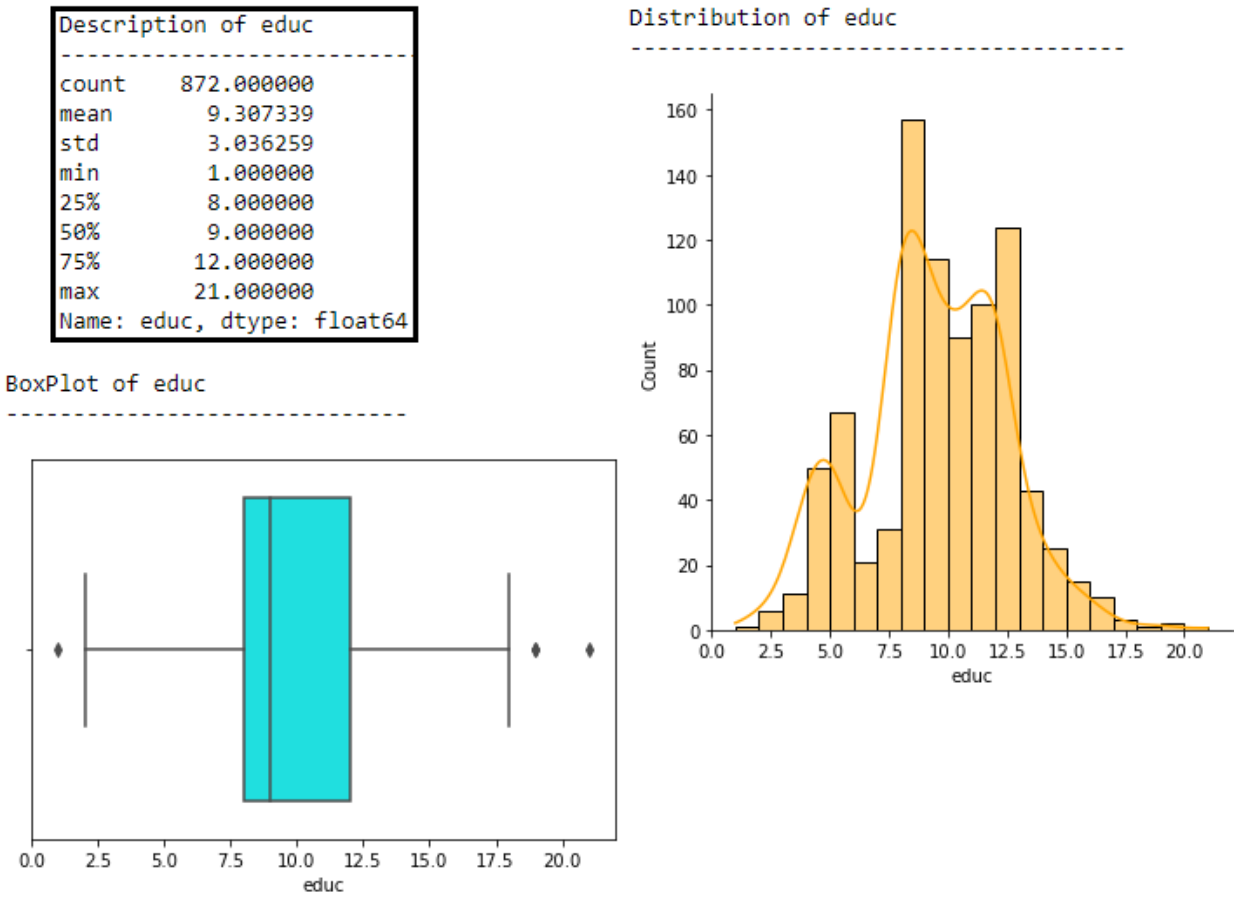


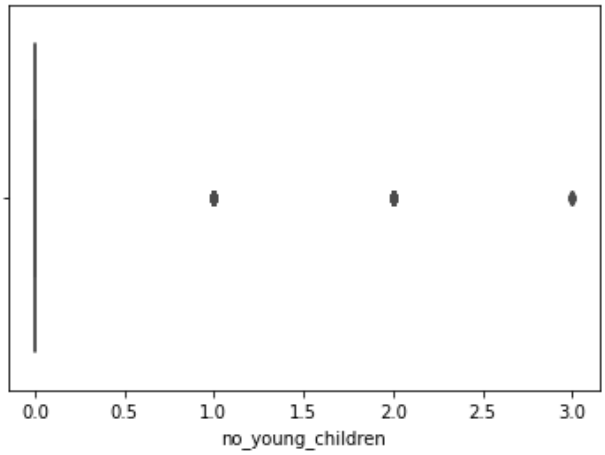
Figure 13: Univariate Analysis of 'educ'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- the mean and median values differ by 1 decimal place only.
- the distribution is not Gaussian
- the boxplot shows just 1 outlier on the left side, and 2 to the right

```
Description of no_young_children
---------------------------------
count    872.000000
mean       0.311927
std        0.612870
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        3.000000
Name: no_young_children, dtype: float64
```

BoxPlot of no_young_children



Distribution of no_young_children

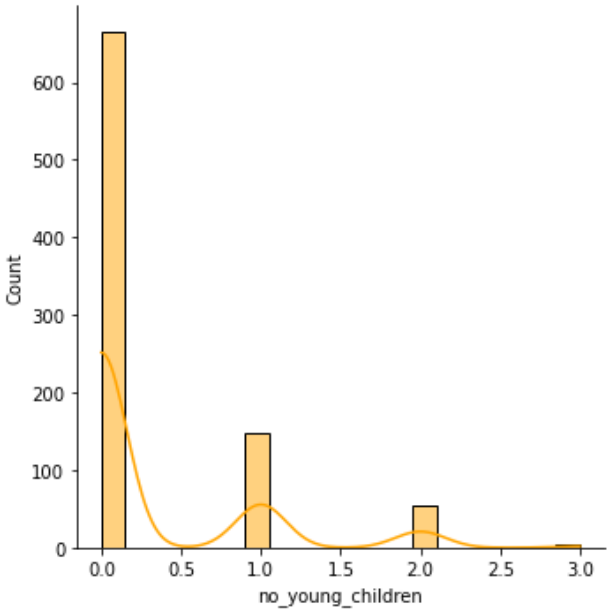


Figure 14: Univariate Analysis of 'no_young_children'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- the median, 25th as well as 75th percentile values are 0, indicating at least 75% of the observations to be zero.
- the distribution shows the feature to be actually having classes of numbers 0, 1, 2 and 3
- the boxplot shows the number of children 1, 2, or 3 to be outliers

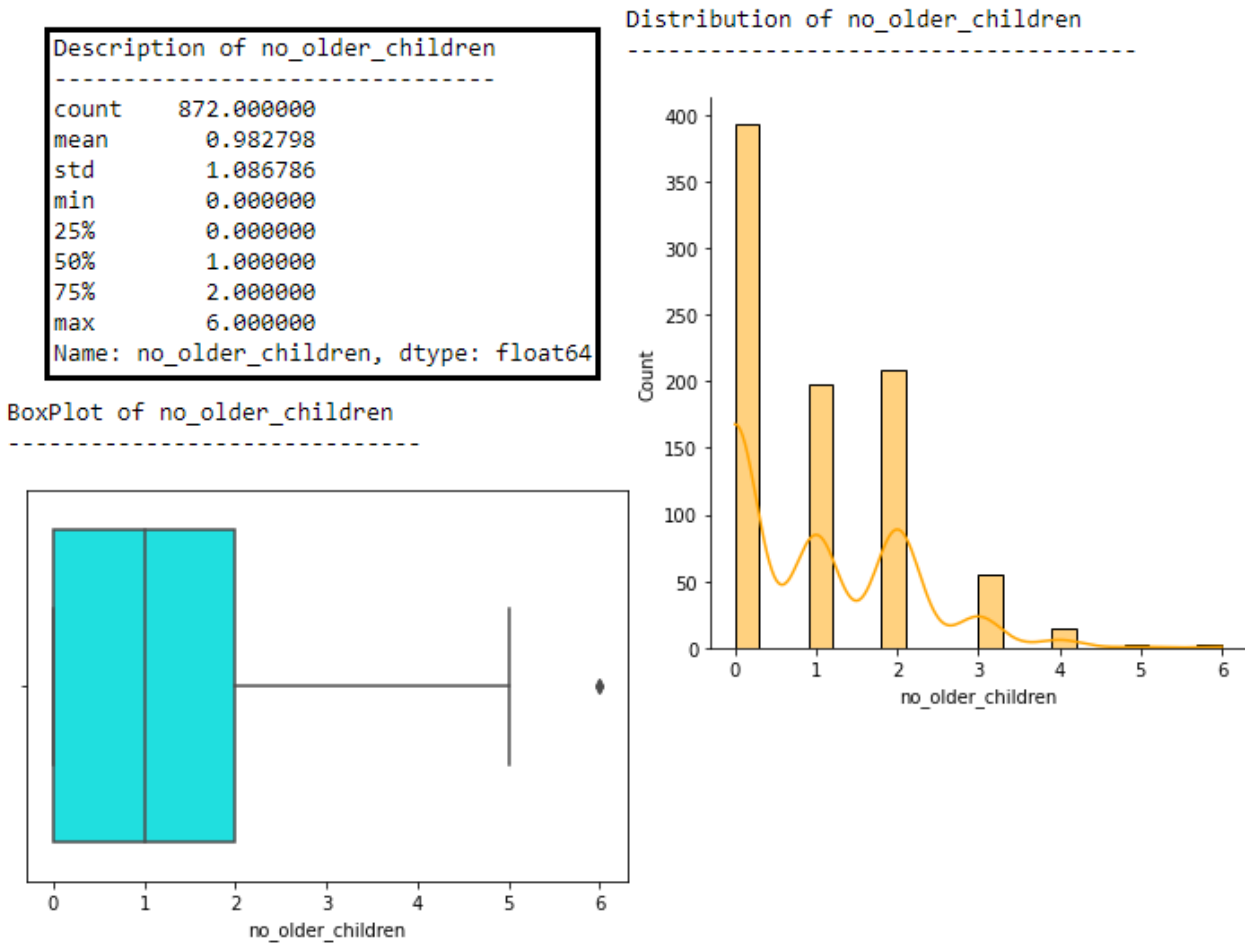


Figure 15: Univariate Analysis of 'no_older_children'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- the 25th percentile, median and 75th percentile are 0, 1 and 2 respectively.
- the distribution shows the feature to be actually having classes of numbers 0, 1, 2, 3, 4, 5 and 6.
- the boxplot shows the number of children 6 to be outlier

Let us now go for the Univariate analysis of 'object' data features in the data set. We will check the unique values, count of unique values and proportion(i.e, normalized values) of unique values in each feature:

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

 HOLLIDAY_PACKAGE (normalized)
--------------------
no     0.540138
yes    0.459862
Name: Holliday_Package, dtype: float64
```

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

 FOREIGN (normalized)
--------------------
no     0.752294
yes    0.247706
Name: foreign, dtype: float64
```

Tables 17: Univariate Analysis of Categorical Data(Problem 2)

(above tables generated with ‘.value_counts()’ function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

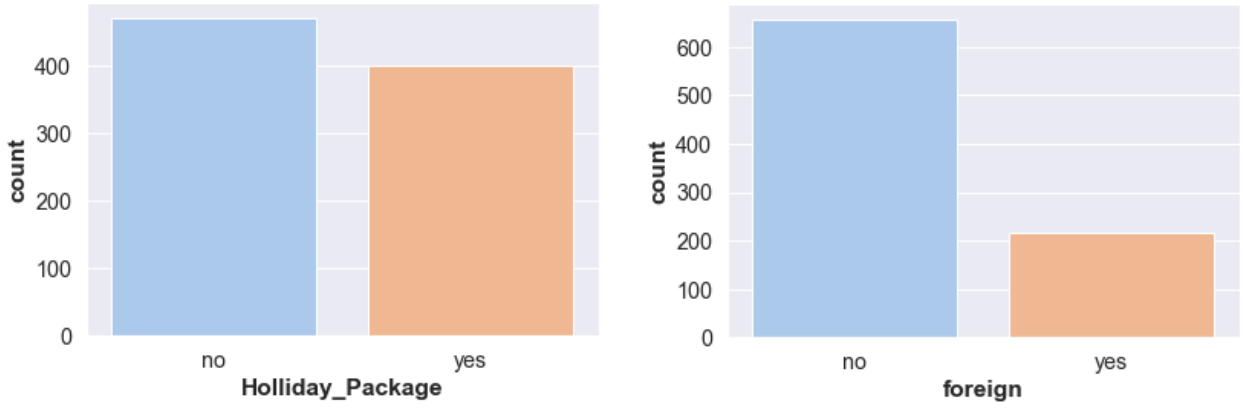


Figure 16: Count plots of Categorical Data(Problem 2)

(above plots generated using ‘.countplot()’ function of Seaborn library in Python)

We can see:

- the target feature, ‘Holliday_Package’, has class proportions 54.01% ‘no’ and 45.99% ‘yes’
- the feature ‘foreign’ has imbalanced classes of 75.23% ‘no’ and 24.77% ‘yes’, so we should avoid having conclusions biased to the class ‘no’.

Let us go for bivariate and multivariate analysis now.

To check the correlations amongst numerical features, let us have a look at the Pearson’s Correlation coefficients in the form of a heatmap:

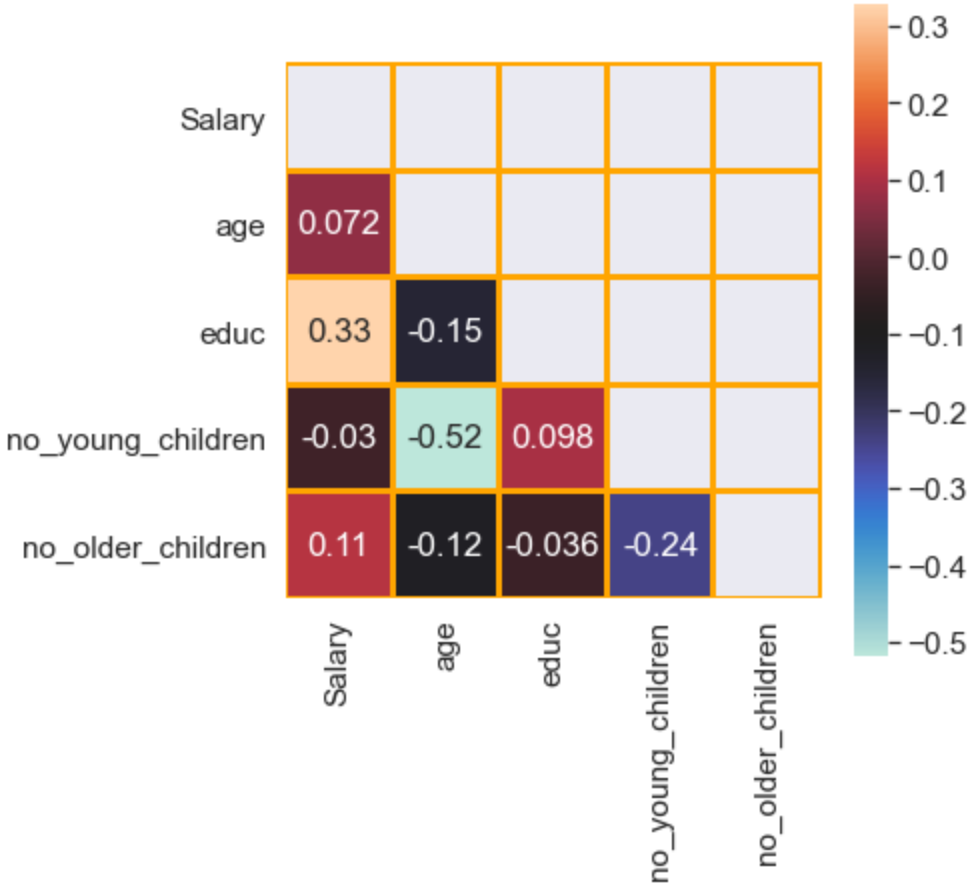


Figure 17: Correlation Heatmap(Problem 2)

(above plot generated using ‘.heatmap()’ function of Seaborn library in Python)

- Pearson’s Correlation coefficients near to 1 or -1 are highly positively correlated and highly negatively correlated respectively. Correlation values near to 0 are not or minimally correlated to each other.
- We see very weak correlations amongst the features, most of them close to 0.

Now, let us view the pairplot of the numeric features:

(zoom to at least 150% for better visibility)

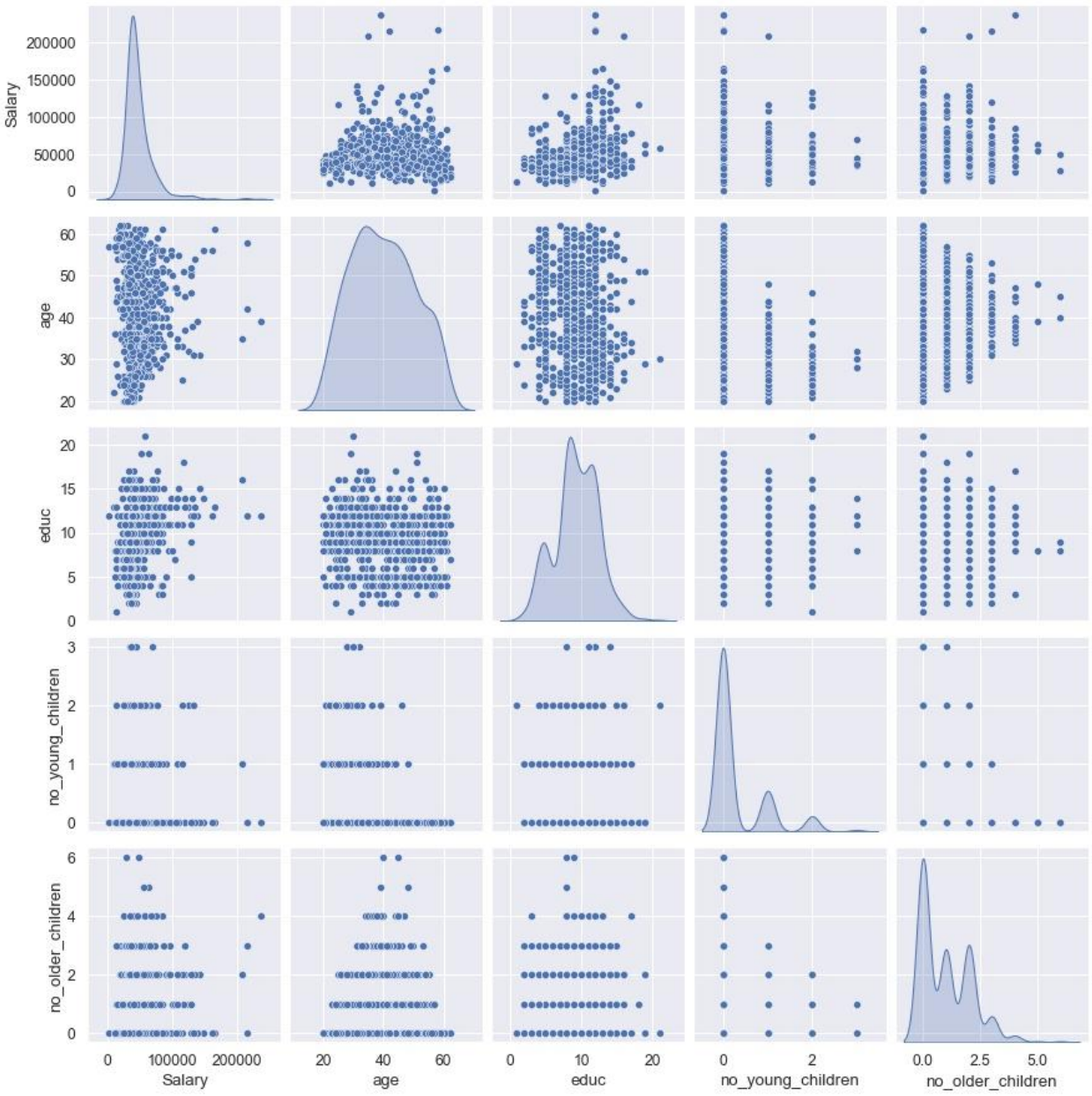


Figure 18: Pairplot(Problem 2)

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Pairplot shows the relationship between the numeric variables in the form of scatterplot and the distribution of the variables in the form of histogram.

- The pattern of clouds of points depict very weak correlations.
- Features 'no_young_children' and 'no_older_children' are having classes of numbers rather than continuous numerical data.

Let us visualise how the target feature 'Holliday_Package' is distributed over other categorical variable, 'foreign':

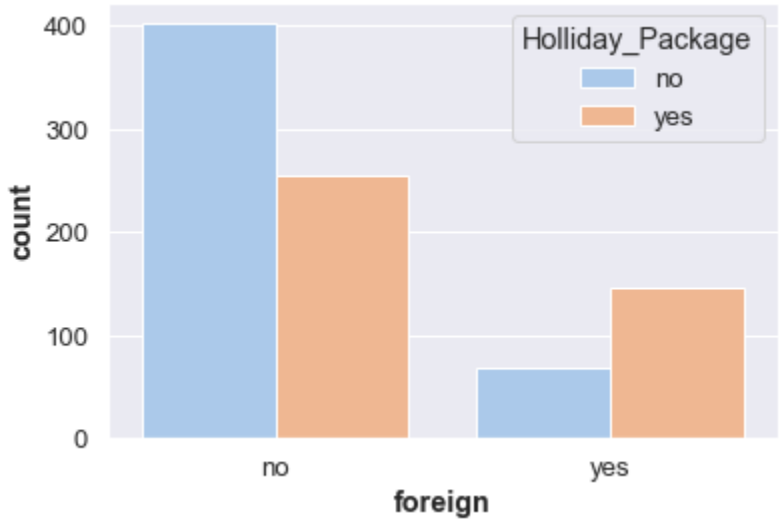


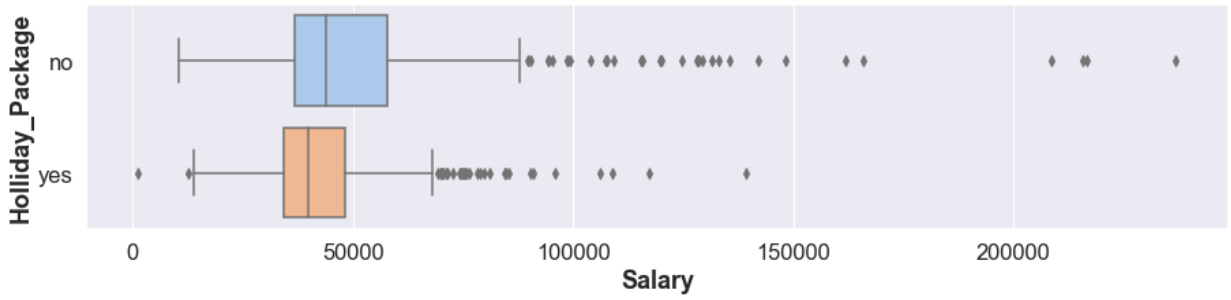
Figure 19: Distribution of Target over 'foreign'(Problem 2)

(above figure generated with '.countplot()' function of Seaborn library in Python)

We can see:

- majority of the foreigners opt for holiday packages.
- majority of the native citizens do not opt for holiday packages.
- the no. of customers of native citizens are more than foreigners.

Let us now check how the target 'Holliday_Package' is distributed over the numeric features of the data:



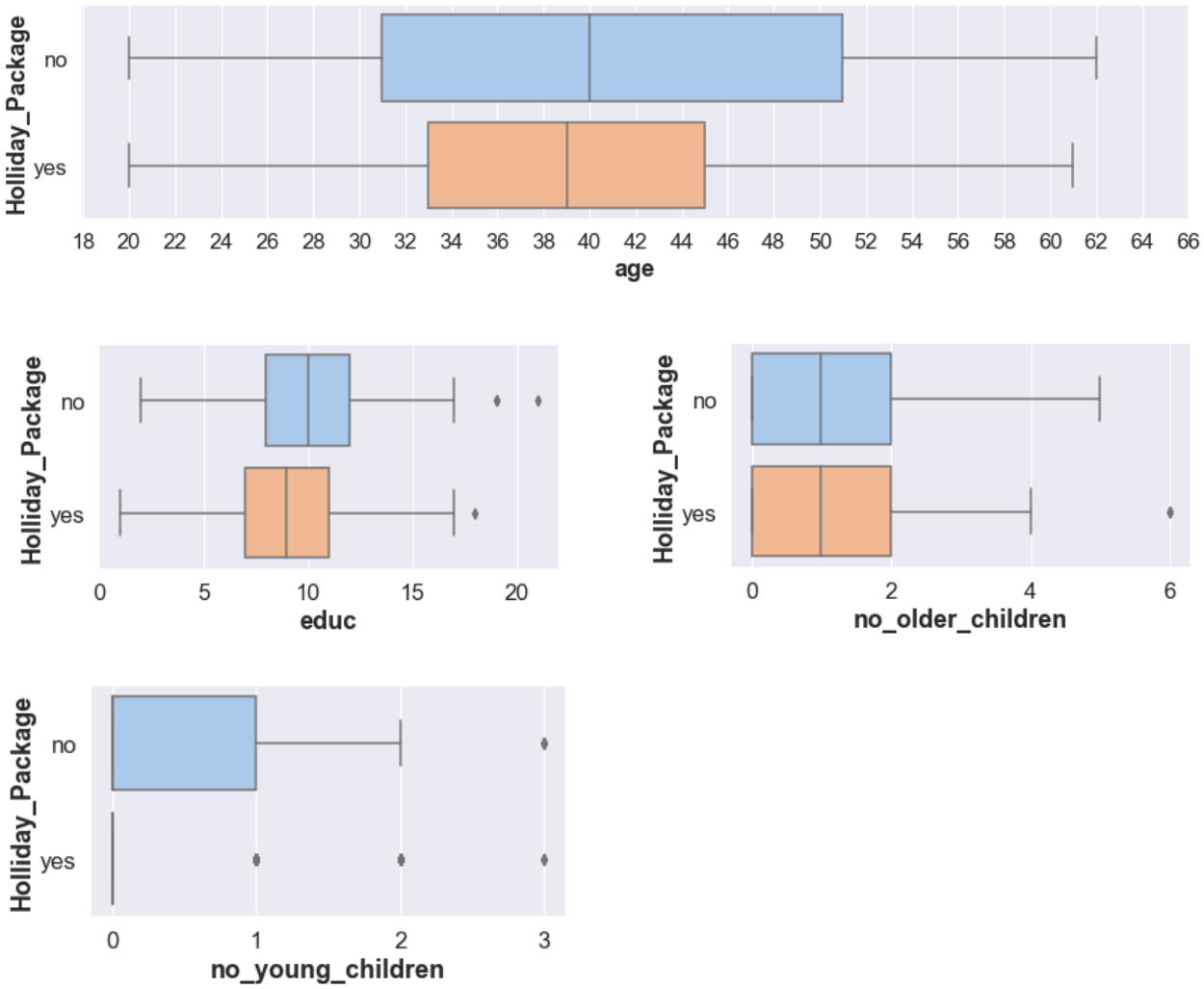


Figure 20: Distribution of Target over Numerical features(Problem 2)

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- Most customers with high salary values do not opt for holiday packages
- Customers in age group 34-45 opt for holiday packages the most(as evident from IQR)
- Spread across 'educ' seems to be almost equal except for an outlier in 'no' class
- 'no_older_children' have equal spread of IQRs

Below we can see the feature 'Salary' versus target feature 'Holliday_Package' hued over feature 'foreign' and 'Destination' respectively:

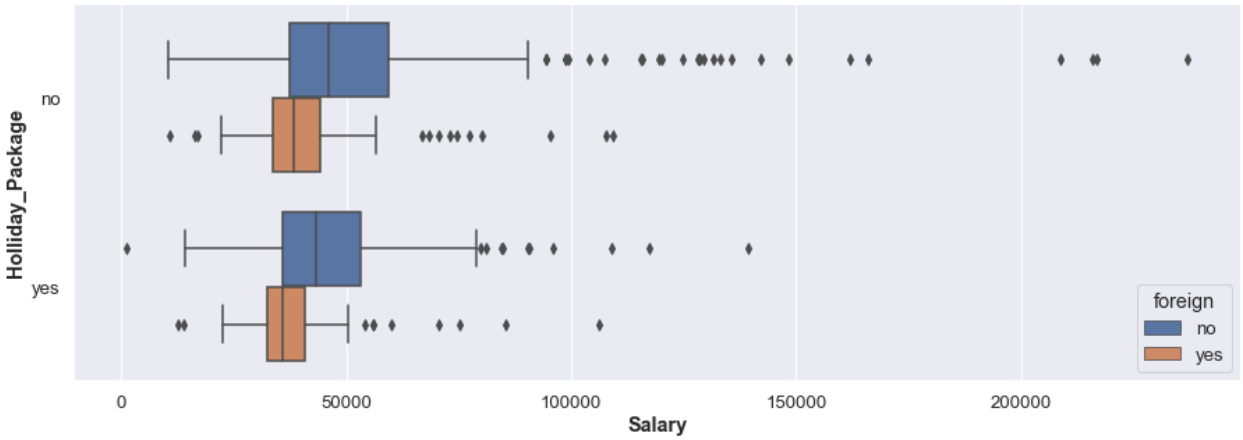


Figure 21: Multivariate Analysis(Problem 2) - 1

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- the majority of native citizens do not opt for holiday packages that are with high salaries.
- foreigners are equally likely to opt for holiday packages.

Below we can see the feature 'age' versus target feature 'Holliday_Package' hued over feature 'foreign' and 'Destination' respectively:

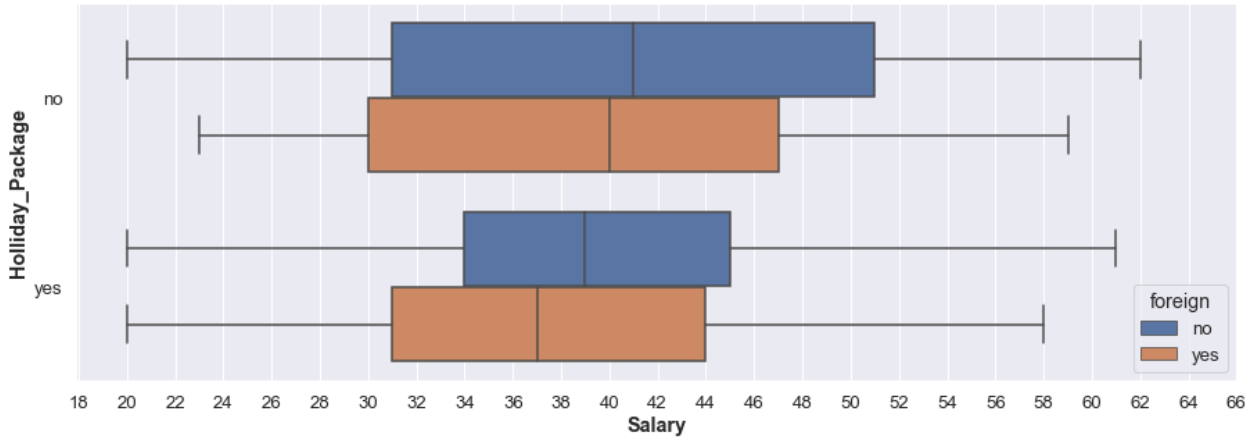


Figure 22: Multivariate Analysis(Problem 2) - 2

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- The IQR spread of native citizens is more than that of foreigners.

Now, let us view the pairplot of the numeric features hued over target feature 'Holliday_Package':

(zoom to at least 150% for better visibility)

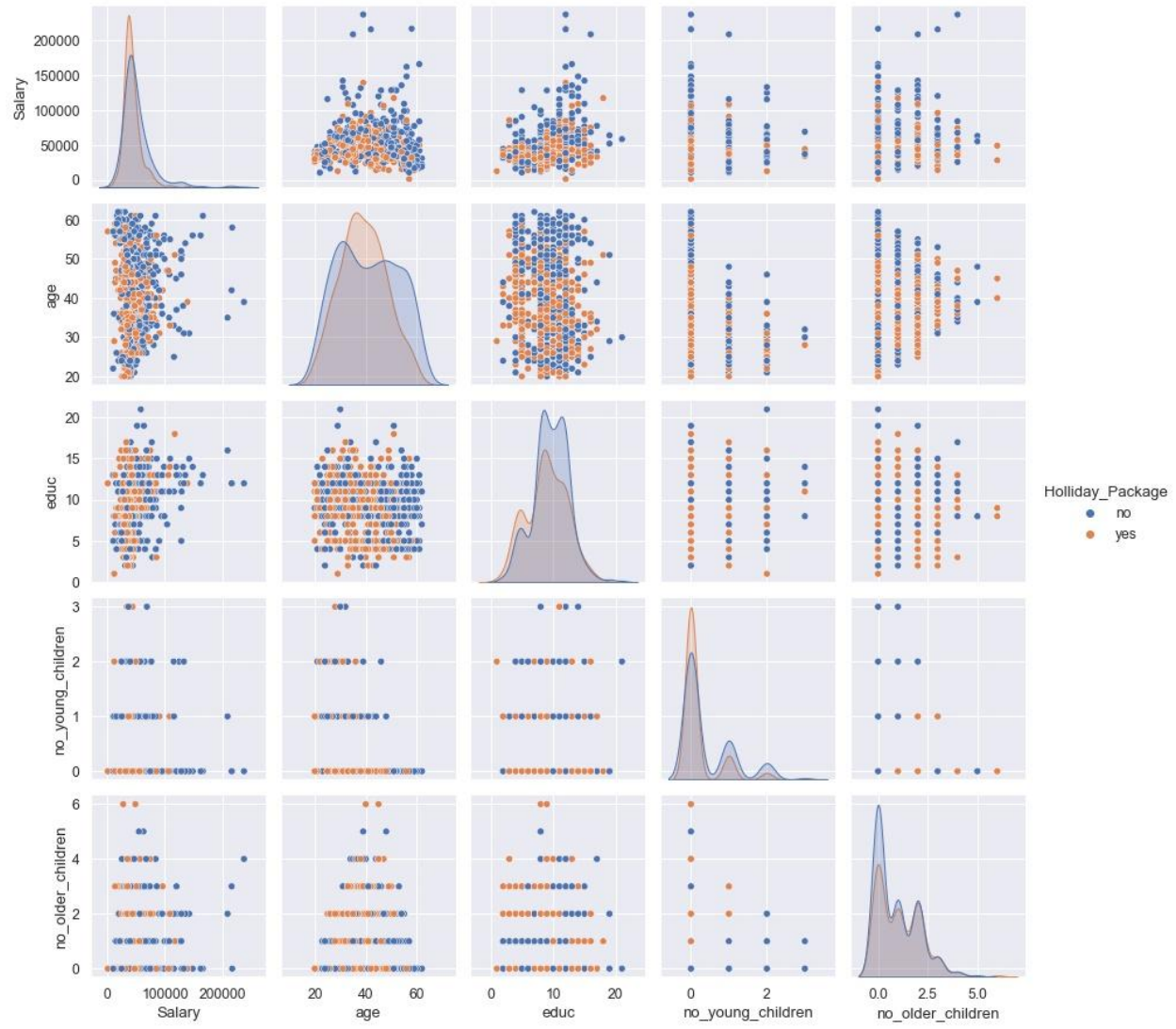


Figure 23: Pairplot hued over target feature 'Holliday_Package'

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Now, let us view the pairplot of the numeric features hued over feature 'foreign':

(zoom to at least 150% for better visibility)

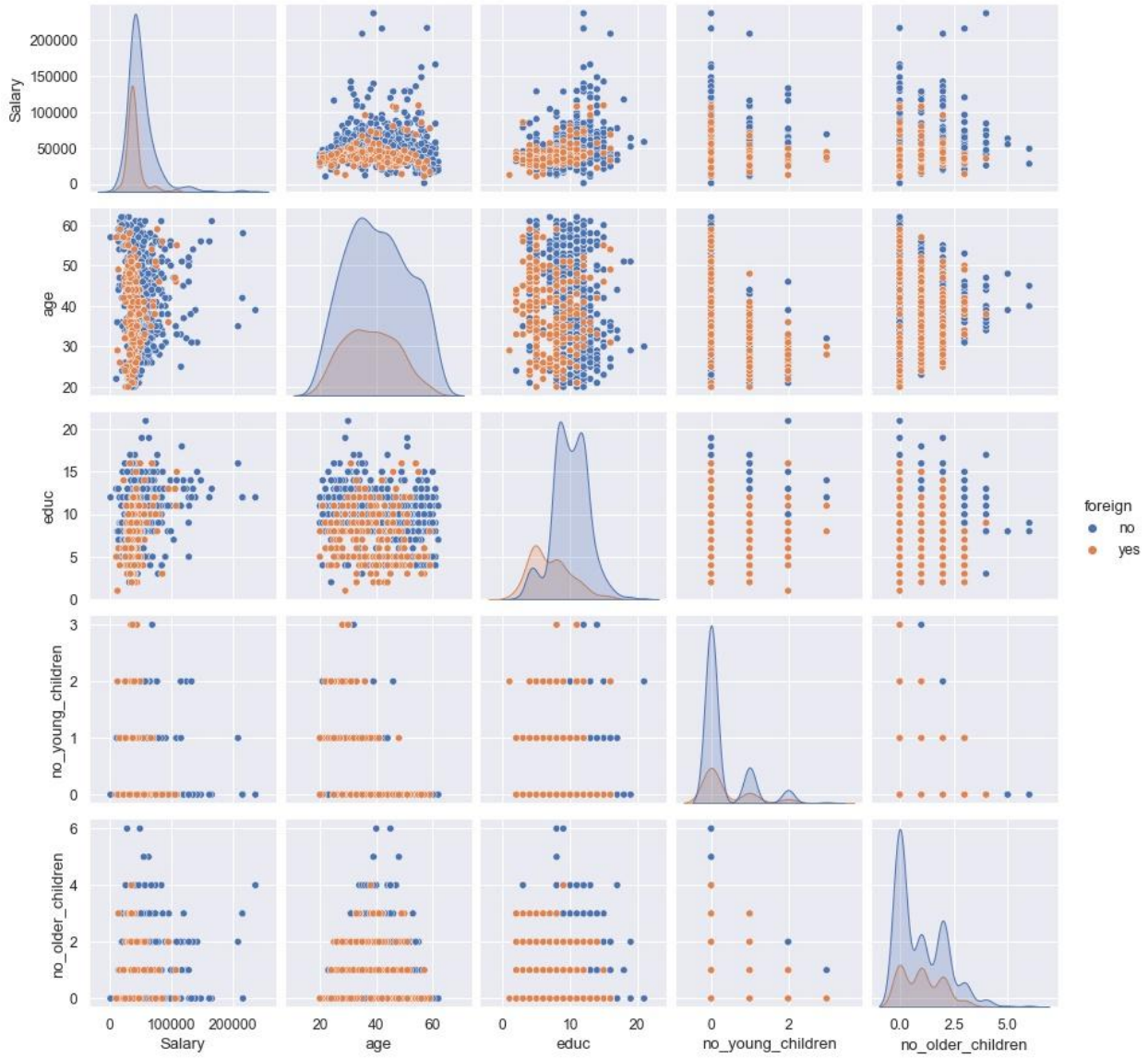


Figure 24: Pairplot hued over feature 'foreign'

(above figure generated with '.pairplot()' function of Seaborn library in Python)

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).

- Features 'Holliday_Package' and 'foreign' are object type and need to be encoded.
- 'np.where()' has been used for the same. The 'yes' in both the features have been replaced with '1' and the 'no' have been replaced with '0'.
- '.astype()' was used to convert the above 2 features into data-type integer(8 bits).

Let us check the general information again after above data preparation:

```
<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    int8
 1   Salary             872 non-null    int64
 2   age                872 non-null    int64
 3   educ               872 non-null    int64
 4   no_young_children  872 non-null    int64
 5   no_older_children  872 non-null    int64
 6   foreign            872 non-null    int8
dtypes: int64(5), int8(2)
memory usage: 35.9 KB
```

Table 18: General Information of Prepared Data(Problem 2)

(above table generated by '.info' of 'PANDAS' in Python)

- We are not treating the outliers as evident from the EDA's distributions(KDEs and boxplots), we see a significant number of outliers, and treating such a large number of outliers may affect the model's performance in a real production environment.
- 'Holliday_Package' being the continuous target variable is separated from the other predictor features for model building.
- 'train_test_split' from 'sklearn.model_selection' has been used to split the data in the ratio of 70:30 (train : test), by providing test_size as 0.30. 'random state' of 1 has been provided.

For both Logistic Regression and LDA, we will see and compare 4 models each:

1. General fitted model
2. Model after probability threshold optimisation
3. General fitted model with polynomial features
4. Model with polynomial features after probability threshold optimisation

Logistic Regression(LR)

1. **LR General fitted model**

- A grid search operation was carried out using 'GridSearchCV' of 'sklearn.model_selection'. Following was the grid of parameters:

'penalty':['none', 'l2', 'elasticnet', 'l1'],

'solver':['newton-cg', 'lbfgs', 'sag', 'saga', 'liblinear'],

'tol':[0.0001,0.00001,0.001]}

- 'GridSearchCV' also took the following parameters:

- ❖ cv = 3, indicating a 3 fold cross-validation
- ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
- ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score

(Note that not the entire grid be passed in one go as 'newton-cg' solver supports only 'l2' and 'none' penalties, and only 'saga' solver supports 'elasticnet'. So, the grid search needs to done in more than one go)

- 'f1-score' is being used as the 'scoring policy' in the grid search operation.
- Following were the best parameters from the grid search operation:

  'penalty': 'none', 'solver': 'newton-cg', 'tol': 0.0001
- Model was built using 'LogisticRegression' of 'sklearn.linear_model'.
- Model was fit on the train set.
- Predictions were made both on the train and test sets with default probability threshold of 0.5, and performance metrics were calculated.
- The model intercept was found out to be 2.46
- Following were the coefficients found for the various features:

  Salary : -1.6461420085214953e-05

  age : -0.057072537177018094

  educ : 0.06034733146849842

  no_young_children : -1.348834795784143

  no_older_children : -0.0489436595505914

  foreign : 1.2664792025982512

2. <u>LR Model after probability threshold optimisation</u>

- Same grid search parameters' results obtained in model 1 were used as the data set is the same.
- Model was built using 'LogisticRegression' of 'sklearn.linear_model'.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:

	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.485246	0.640732	0.996441	0.472175
1	0.2	0.518033	0.646635	0.957295	0.488203
2	0.3	0.591803	0.668442	0.893238	0.534043
3	0.4	0.675410	0.684713	0.765125	0.619597
4	0.5	0.667213	0.616257	0.580071	0.657258
5	0.6	0.659016	0.535714	0.427046	0.718563
6	0.7	0.649180	0.433862	0.291815	0.845361
7	0.8	0.588525	0.198083	0.110320	0.968750
8	0.9	0.542623	0.014134	0.007117	1.000000

Table 19: Metrics for different thresholds for LR model 2

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.4 was chosen which gave the best f1-score as well as accuracy.

- Predictions on train and test were changed according to the new threshold and metrics were calculated.

3. <u>LR General fitted model with polynomial features</u>

- Same grid search parameters' results obtained in model 1 were used as the data set is the same.
- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 4. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- Model was built using 'LogisticRegression' of 'sklearn.linear_model'.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Following were the coefficients found:

  3.41802197e-03, 9.93110568e-05, -1.32213958e-02,

  2.73206430e-02, -1.36689440e-02, 1.05449281e-02,

  1.14258253e-03, -2.40248982e-06, -9.41060101e-07,

  3.64118296e-05, -6.10661485e-05, 2.10790258e-04,

  1.94783791e-03, -2.34673787e-01, 2.59489606e-02,

-3.97181254e-02, -7.64523388e-02, 8.69626111e-02,

-2.06933694e-02, -1.82536996e-03, 8.71792801e-04,

 1.11430828e-02, 1.19247999e-08, 6.14983186e-07,

 8.61425717e-07, -1.87490289e-06, -1.39196463e-05,

-5.60007034e-07, -2.64838905e-05, 9.03401126e-05,

 8.14905655e-05, -1.63385969e-06, 2.06330332e-02,

-4.68736306e-03, 9.61561310e-03, 1.75042692e-02,

 2.23471279e-02, 9.44881512e-02, -2.12501034e-02,

-8.87837835e-03, 4.66643700e-02, 3.11648943e-03,

 2.09024502e-07, 6.31120749e-08, 2.60406264e-07,

-2.72198082e-06, -3.56902611e-06, -2.80503569e-06,

 1.51318716e-06, 4.88013071e-06, 1.24996998e-05,

-1.90479998e-04, -1.99551619e-03, 3.08389392e-04,

-1.21871745e-02, 2.35287567e-01, 1.20403678e-02

4. <u>LR Model with polynomial features after probability threshold optimisation</u>

- Model 3 was used for proceeding further with probability threshold optimization.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:

	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.531148	0.663529	0.992958	0.498233
1	0.2	0.578689	0.683107	0.975352	0.525617
2	0.3	0.637705	0.700136	0.908451	0.569536
3	0.4	0.685246	0.703704	0.802817	0.626374
4	0.5	0.696721	0.647619	0.598592	0.705394
5	0.6	0.690164	0.571429	0.443662	0.802548
6	0.7	0.668852	0.507317	0.366197	0.825397
7	0.8	0.637705	0.404313	0.264085	0.862069
8	0.9	0.581967	0.190476	0.105634	0.967742

Table 20: Metrics for different thresholds for LR model 4

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.4 was chosen which gave the best f1-score, and decent accuracy.

- Predictions on train and test were changed according to the new threshold and metrics were calculated.

Linear Discriminant Analysis

1. **LDA General fitted model**

- 'LinearDiscriminantAnalysis' from 'sklearn.discriminant_analysis' is being used to build the model.
- Model was fit on the train set.
- Predictions were made both on the train and test sets with default probability threshold of 0.5, and performance metrics were calculated.

2. **LDA Model after probability threshold optimisation**

- 'LinearDiscriminantAnalysis' from 'sklearn.discriminant_analysis' is being used to build the model.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:

	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.481967	0.639269	0.996441	0.470588
1	0.2	0.521311	0.649880	0.964413	0.490054
2	0.3	0.593443	0.669333	0.893238	0.535181
3	0.4	0.665574	0.676190	0.758007	0.610315
4	0.5	0.663934	0.612476	0.576512	0.653226
5	0.6	0.659016	0.533632	0.423488	0.721212
6	0.7	0.649180	0.439791	0.298932	0.831683
7	0.8	0.588525	0.198083	0.110320	0.968750
8	0.9	0.542623	0.014134	0.007117	1.000000

Table 21: Metrics for different thresholds for LDA model 2

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.4 was chosen which gave the best f1-score as well a decent accuracy.

- Predictions on train and test were changed according to the new threshold and metrics were calculated.

3. LDA General fitted model with polynomial features

- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 4. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- 'LinearDiscriminantAnalysis' from 'sklearn.discriminant_analysis' is being used to build the model.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.

4. LDA Model with polynomial features after probability threshold optimisation

- Model 3 was used for proceeding further with probability threshold optimization.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:

	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.467213	0.634421	0.992958	0.466116
1	0.2	0.473770	0.628042	0.954225	0.468048
2	0.3	0.483607	0.619105	0.901408	0.471455
3	0.4	0.536066	0.596291	0.735915	0.501199
4	0.5	0.596721	0.467532	0.380282	0.606742
5	0.6	0.563934	0.288770	0.190141	0.600000
6	0.7	0.540984	0.166667	0.098592	0.538462
7	0.8	0.529508	0.059016	0.031690	0.428571
8	0.9	0.527869	0.006897	0.003521	0.166667

Table 22: Metrics for different thresholds for LDA model 4

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

After trying for nearby values, threshold of 0.4 was chosen which is the best trade-off between threshold 0.1(with best f1-score) and threshold 0.5(with best accuracy). Also, for thresholds 0.1, 0.2 and 0.3, the model seems to overfit with polynomial features of degree 4.

- Predictions on train and test were changed according to the new threshold and metrics were calculated.

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.

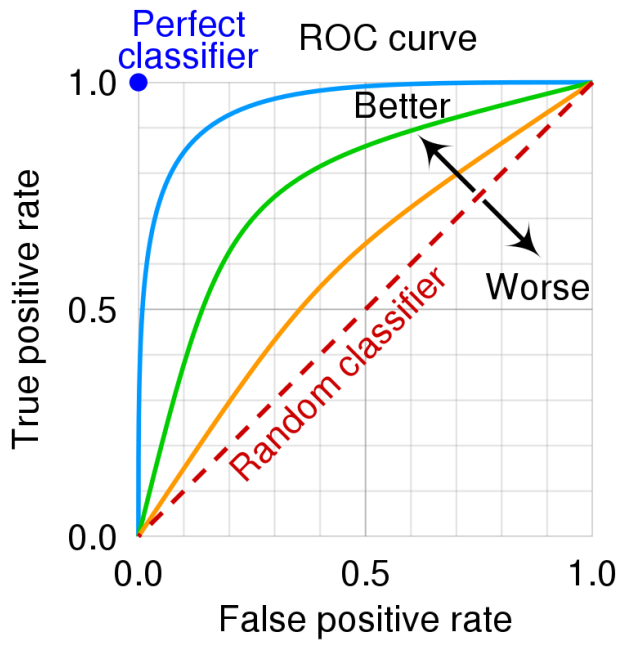
- Classification accuracy is the ratio of correct predictions to total predictions made.
- A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive **+**	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

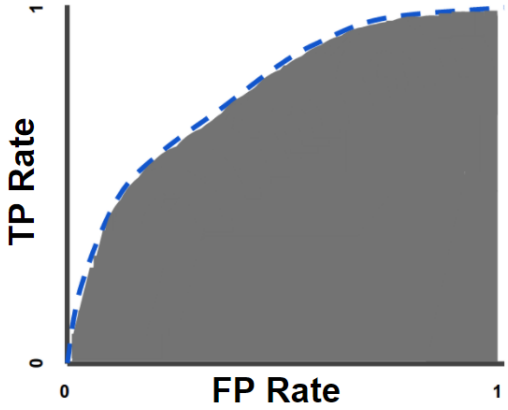
- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
  - ❖ True Positive Rate(TPR)
  - ❖ False Positive Rate(FPR)

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

- An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows few ROC curves:

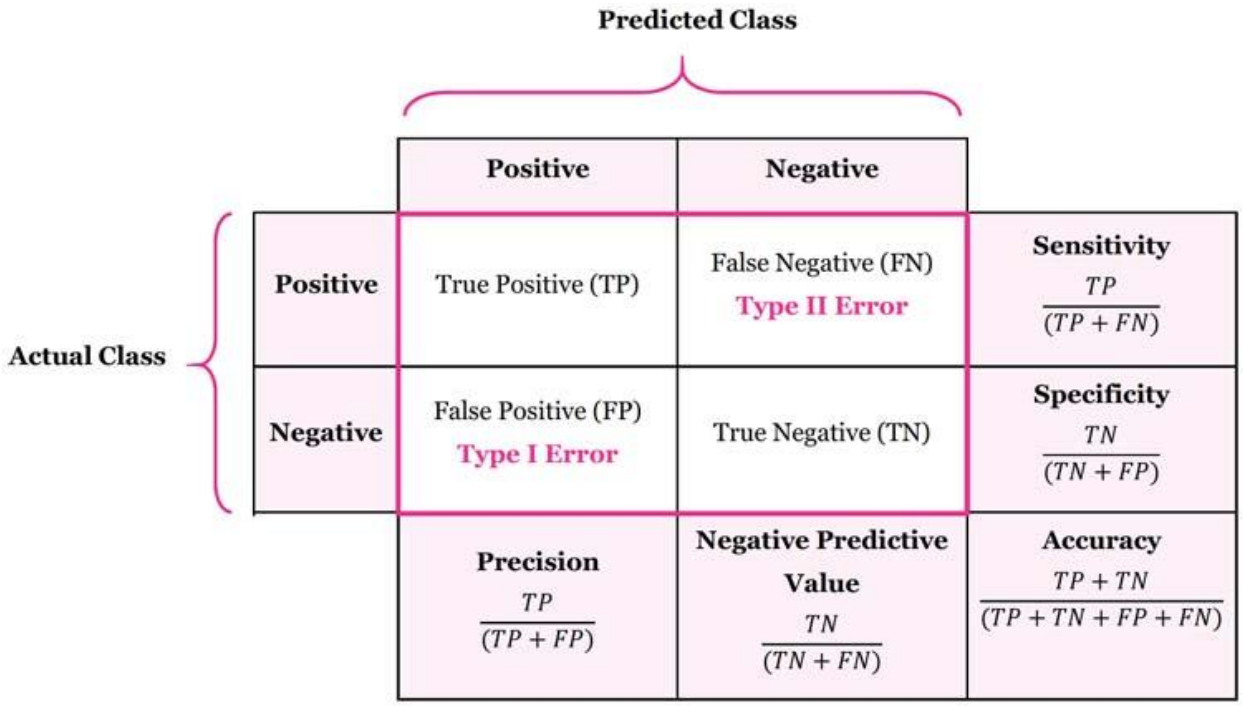


- AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). The grey shaded portion in the below ROC curve represents the AUC for it:



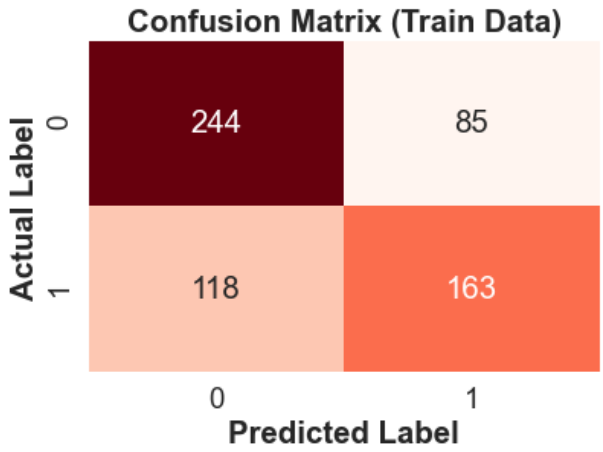
- A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of any trained classification model.It provides a better understanding of the overall performance of our trained model. To understand the classification report of a machine learning model, following are the metrics displayed in the report:
    - ❖ **Precision:** Precision is defined as the ratio of true positives to the sum of true and false positives. A model that produces no false positives has a precision of 1.0.

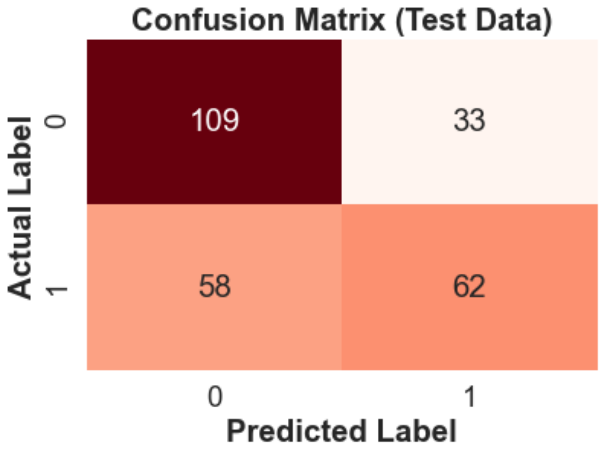
- **Recall(or Sensitivity):** Recall is defined as the ratio of true positives to the sum of true positives and false negatives. A model that produces no false negatives has a recall of 1.0.
- **F1-score:** The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
- **Support:** Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.



Logistic Regression(LR)

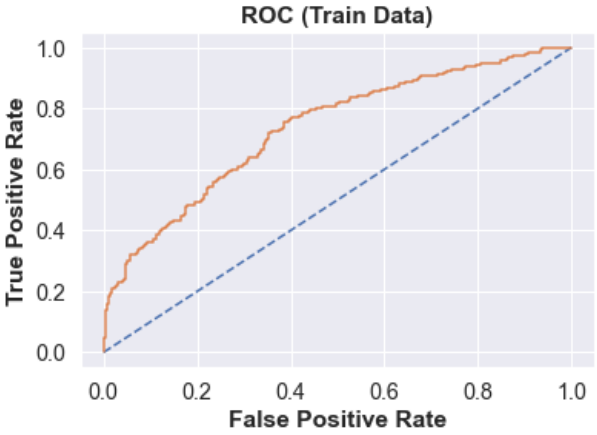
1. **LR General fitted model**





Train Accuracy: 66.72%

Test Accuracy: 65.27%



Train AUC: 0.735

Test AUC: 0.717

Training Classification Report:

	precision	recall	f1-score	support
0	0.67	0.74	0.71	329
1	0.66	0.58	0.62	281
accuracy			0.67	610
macro avg	0.67	0.66	0.66	610
weighted avg	0.67	0.67	0.66	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.65	0.77	0.71	142
1	0.65	0.52	0.58	120
accuracy			0.65	262
macro avg	0.65	0.64	0.64	262
weighted avg	0.65	0.65	0.65	262

Train Data:

AUC: 73.5%

Accuracy: 66.72%

Precision(positive class): 66%

Test Data:

AUC: 71.7%

Accuracy:65.27%

Precision(positive class): 65%

Recall(positive class): 58%
f1-Score(positive class): 62%

Recall(positive class): 52%
f1-Score(positive class): 58%

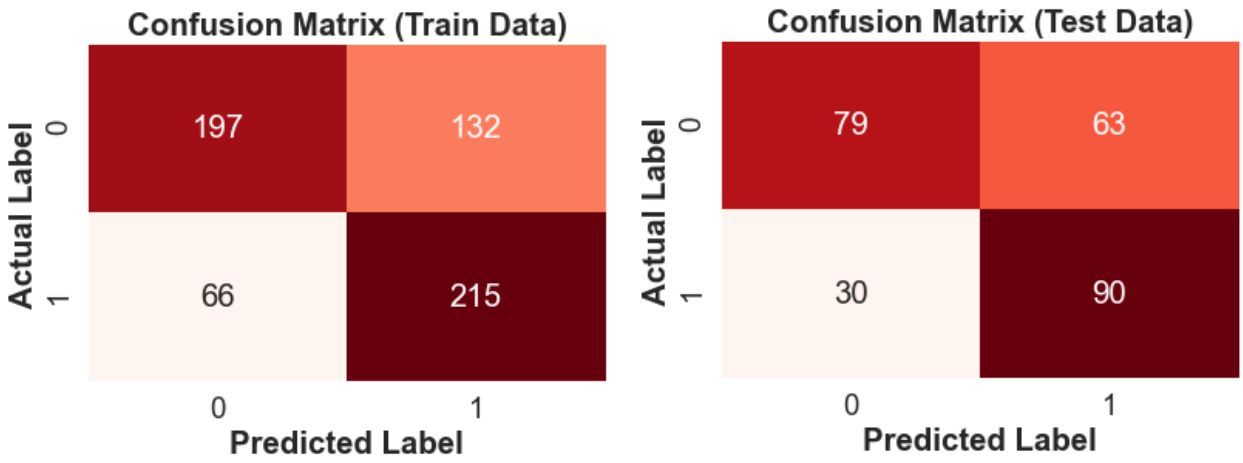
Figure 25: LR Model 1 Metrics

(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

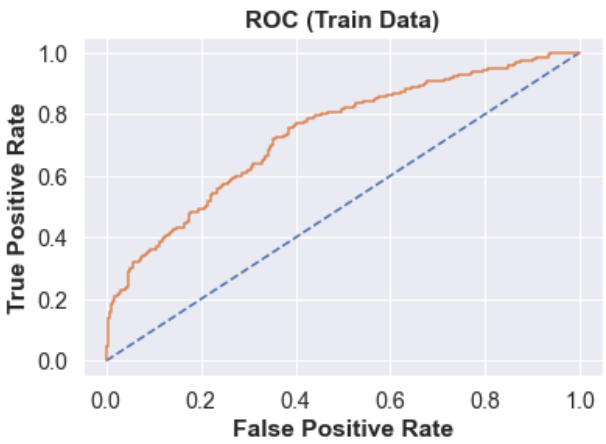
- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- As the target feature has low samples in class 'Yes'(1), we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 34%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 35% of the time(1-Precision).
- The false negatives during training were 42%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 48%(1-Recall) of the time.
- The model is an average model.

2. <u>LR Model after probability threshold optimisation</u>



Train Accuracy: 67.54%

Test Accuracy: 64.50%



Train AUC: 0.735

Test AUC: 0.717

Training Classification Report:

	precision	recall	f1-score	support
0	0.75	0.60	0.67	329
1	0.62	0.77	0.68	281
accuracy			0.68	610
macro avg	0.68	0.68	0.68	610
weighted avg	0.69	0.68	0.67	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.72	0.56	0.63	142
1	0.59	0.75	0.66	120
accuracy			0.65	262
macro avg	0.66	0.65	0.64	262
weighted avg	0.66	0.65	0.64	262

Train Data:

AUC: 73.5%
Accuracy: 67.54%
Precision(positive class): 62%
Recall(positive class): 77%
f1-Score(positive class): 68%

Test Data:

AUC: 71.7%
Accuracy: 64.50%
Precision(positive class): 59%
Recall(positive class): 75%
f1-Score(positive class): 66%

Figure 26: LR Model 2 Metrics

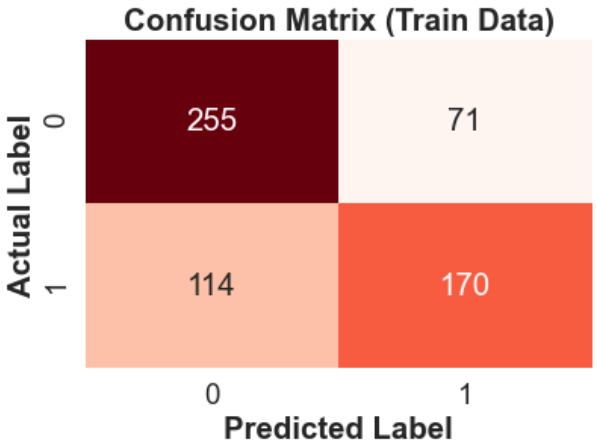
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

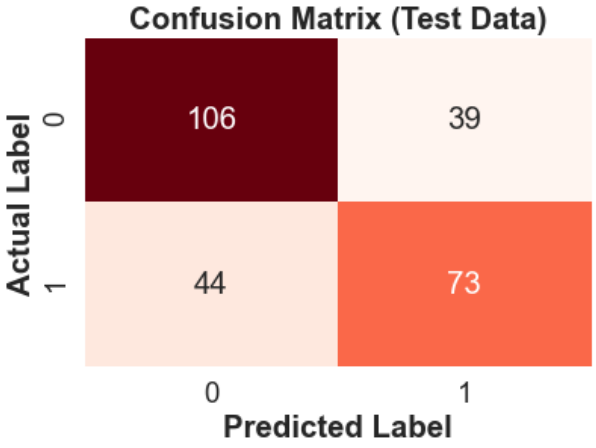
Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.

- After probability threshold optimization, we see the metrics for the positive class and negative class to be similar.
- The false positives during training were 38%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 41% of the time(1-Precision).
- The false negatives during training were 23%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 25%(1-Recall) of the time.
- The model is an average model.

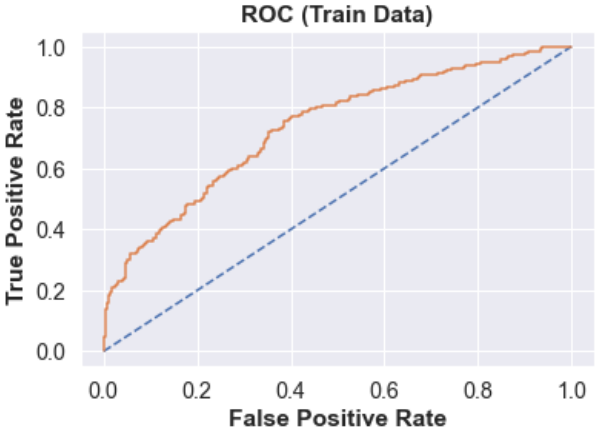
3. LR General fitted model with polynomial features

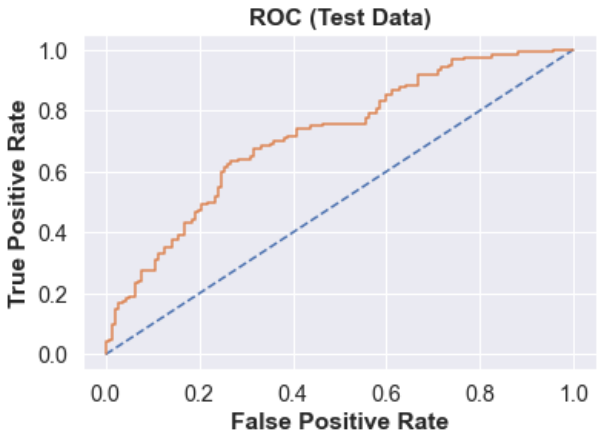




Train Accuracy: 69.67%

Test Accuracy: 68.32%





Train AUC: 0.782

Test AUC: 0.691

Training Classification Report:

Testing Classification Report:

	precision	recall	f1-score	support
0	0.69	0.78	0.73	326
1	0.71	0.60	0.65	284
accuracy			0.70	610
macro avg	0.70	0.69	0.69	610
weighted avg	0.70	0.70	0.69	610

	precision	recall	f1-score	support
0	0.71	0.73	0.72	145
1	0.65	0.62	0.64	117
accuracy			0.68	262
macro avg	0.68	0.68	0.68	262
weighted avg	0.68	0.68	0.68	262

Train Data:
AUC: 78.2%
Accuracy: 69.67%
Precision(positive class): 71%
Recall(positive class): 60%
f1-Score(positive class): 65%

Test Data:
AUC: 69.1%
Accuracy: 68.32%
Precision(positive class): 65%
Recall(positive class): 62%
f1-Score(positive class): 64%

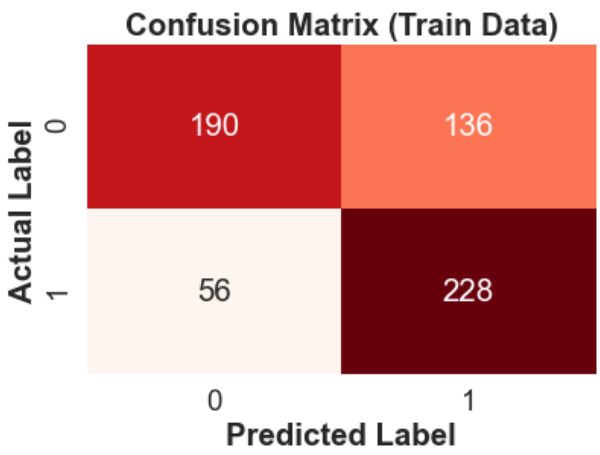
Figure 27: LR Model 3 Metrics

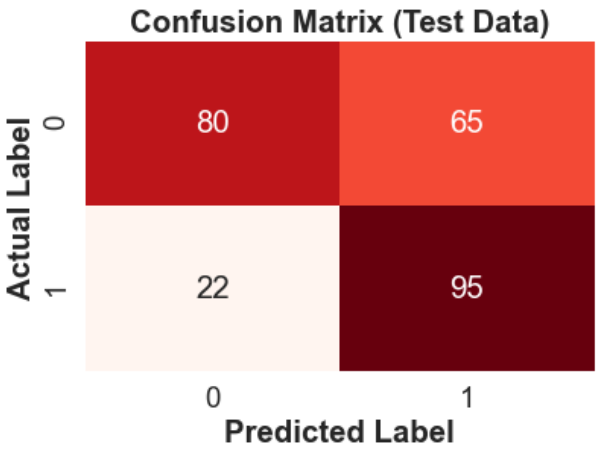
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- As the target feature has low samples in class 'Yes'(1), we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 29%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 35% of the time(1-Precision).
- The false negatives during training were 40%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 38%(1-Recall) of the time.
- The model is an average model.

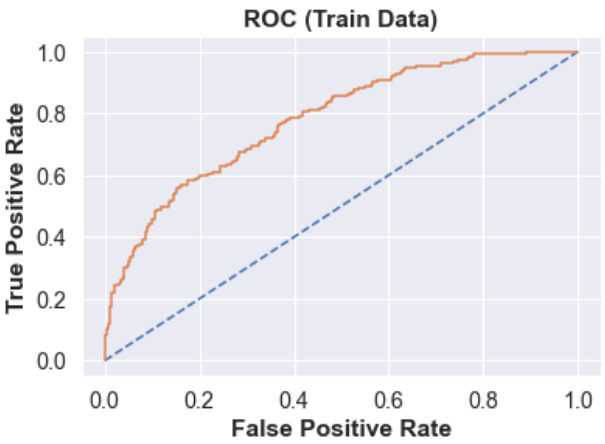
4. LR Model with polynomial features after probability threshold optimisation

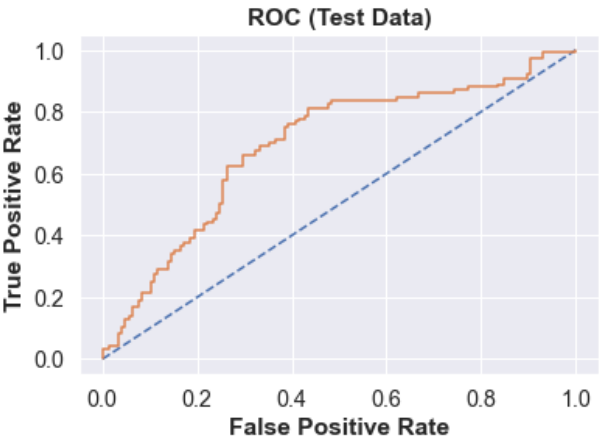




Train Accuracy: 68.52%

Test Accuracy: 66.79%





Train AUC: 0.782

Test AUC: 0.691

Training Classification Report:

	precision	recall	f1-score	support
0	0.77	0.58	0.66	326
1	0.63	0.80	0.70	284
accuracy			0.69	610
macro avg	0.70	0.69	0.68	610
weighted avg	0.70	0.69	0.68	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.78	0.55	0.65	145
1	0.59	0.81	0.69	117
accuracy			0.67	262
macro avg	0.69	0.68	0.67	262
weighted avg	0.70	0.67	0.66	262

Train Data:

AUC: 78.2%

Accuracy: 68.52%

Precision(positive class): 63%

Test Data:

AUC: 69.1%

Accuracy: 66.79%

Precision(positive class): 59%

Recall(positive class): 80%
f1-Score(positive class): 70%

Recall(positive class): 81%
f1-Score(positive class): 69%

Figure 28: LR Model 4 Metrics

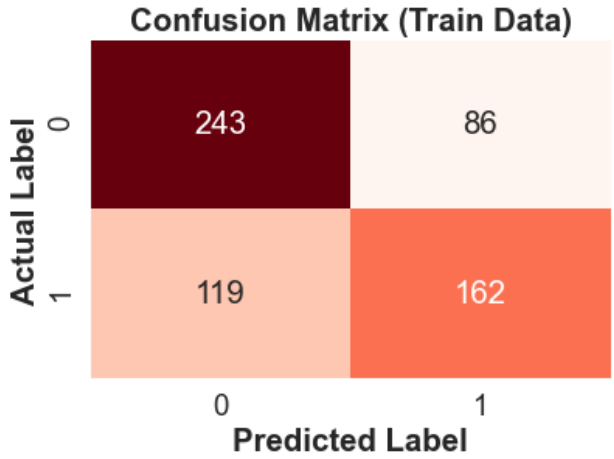
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

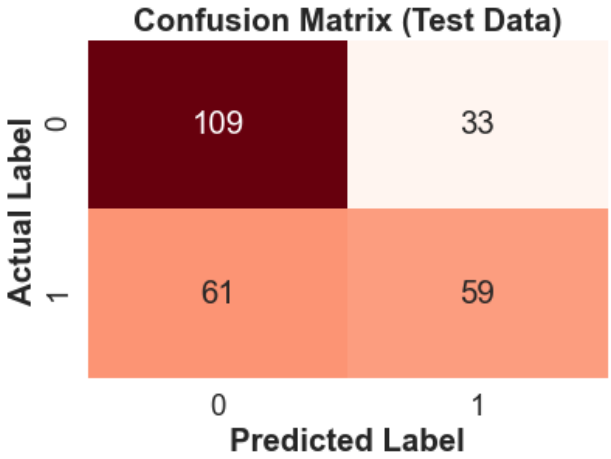
Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- After probability threshold optimization, we see the metrics for the positive class and negative class to be similar.
- The false positives during training were 37%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 41% of the time(1-Precision).
- The false negatives during training were 20%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 19%(1-Recall) of the time.
- The model is an average model.

Linear Discriminant Analysis(LDA)

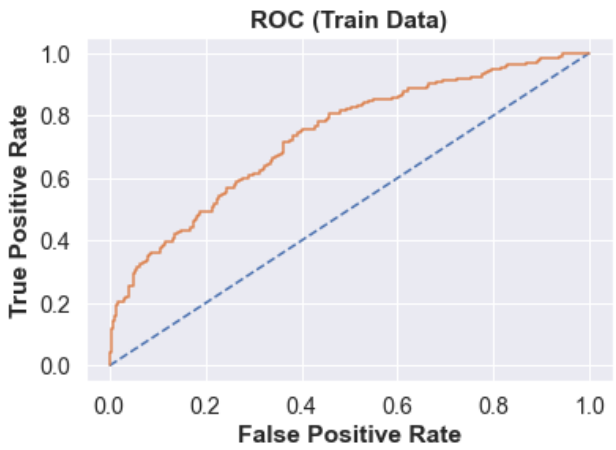
1. LDA General fitted model





Train Accuracy: 66.39%

Test Accuracy: 64.12%



Train AUC: 0.733

Test AUC: 0.714

Training Classification Report:

	precision	recall	f1-score	support
0	0.67	0.74	0.70	329
1	0.65	0.58	0.61	281
accuracy			0.66	610
macro avg	0.66	0.66	0.66	610
weighted avg	0.66	0.66	0.66	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.64	0.77	0.70	142
1	0.64	0.49	0.56	120
accuracy			0.64	262
macro avg	0.64	0.63	0.63	262
weighted avg	0.64	0.64	0.63	262

Train Data:

AUC: 73.3%
Accuracy: 66.39%
Precision(positive class): 65%
Recall(positive class): 58%
f1-Score(positive class): 61%

Test Data:

AUC: 71.4%
Accuracy:64.12%
Precision(positive class): 64%
Recall(positive class): 49%
f1-Score(positive class): 56%

Figure 29: LDA Model 1 Metrics

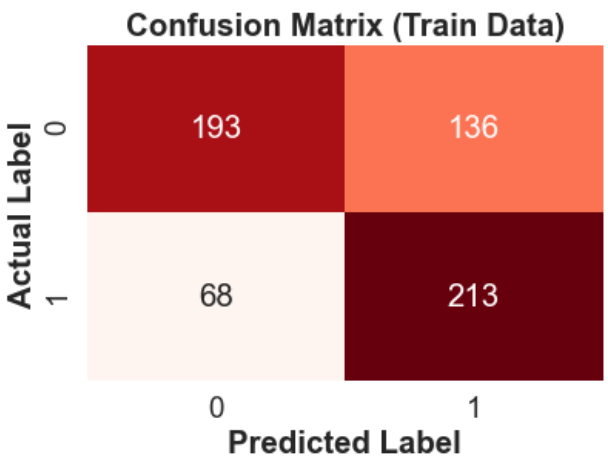
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

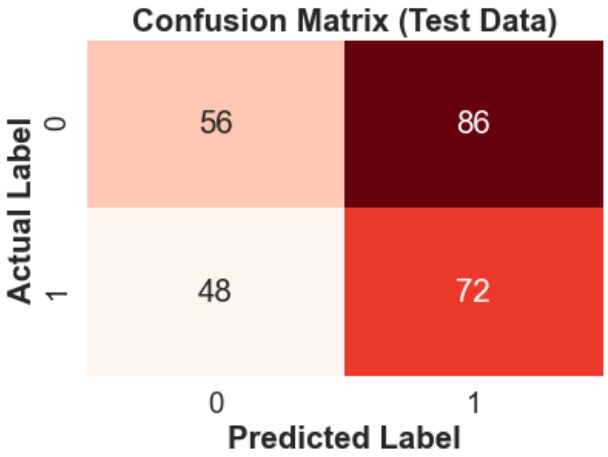
Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.

- As the target feature has low samples in class 'Yes'(1), we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 35%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 36% of the time(1-Precision).
- The false negatives during training were 42%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 51%(1-Recall) of the time.
- The model is an average model.

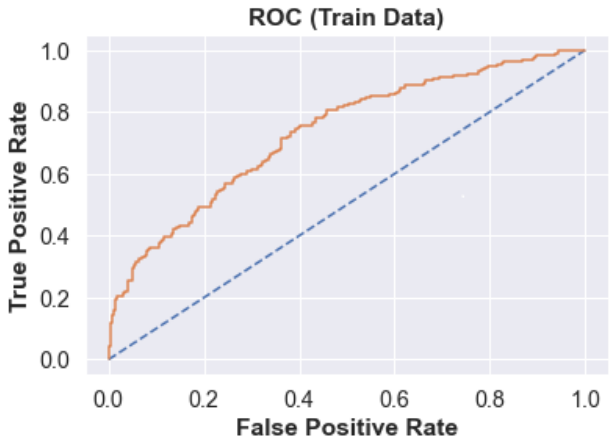
2. LDA Model after probability threshold optimisation





Train Accuracy: 66.56%

Test Accuracy: 48.85%



Train AUC: 0.733

Test AUC: 0.714

Training Classification Report:

	precision	recall	f1-score	support
0	0.74	0.59	0.65	329
1	0.61	0.76	0.68	281
accuracy			0.67	610
macro avg	0.67	0.67	0.67	610
weighted avg	0.68	0.67	0.66	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.54	0.39	0.46	142
1	0.46	0.60	0.52	120
accuracy			0.49	262
macro avg	0.50	0.50	0.49	262
weighted avg	0.50	0.49	0.48	262

Train Data:
AUC: 73.3%
Accuracy: 66.56%
Precision(positive class): 61%
Recall(positive class): 76%
f1-Score(positive class): 68%

Test Data:
AUC: 71.4%
Accuracy: 48.85%
Precision(positive class): 46%
Recall(positive class): 60%
f1-Score(positive class): 52%

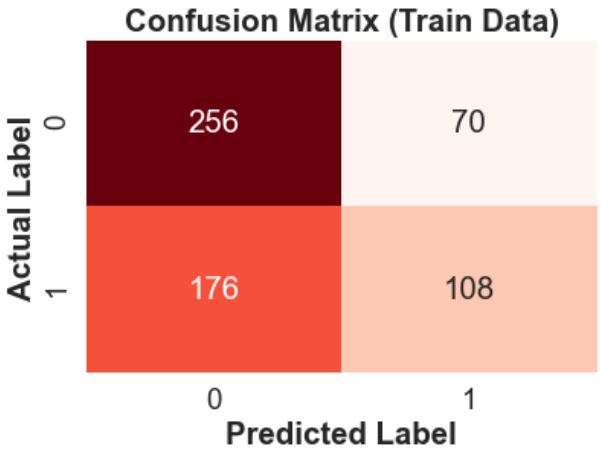
Figure 30: LDA Model 2 Metrics

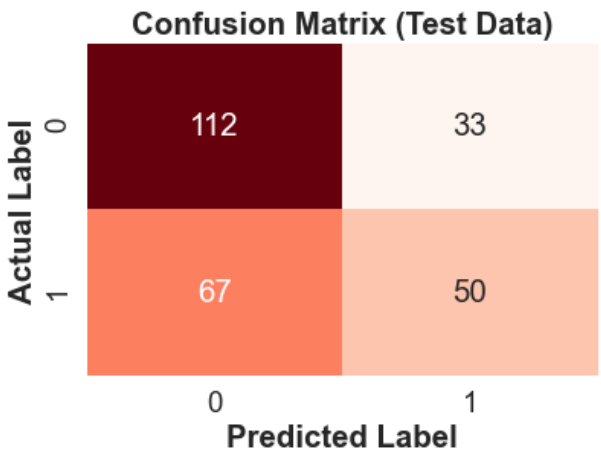
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- After probability threshold optimization, we see the metrics for the positive class and negative class to be similar.
- The false positives during training were 39%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 54% of the time(1-Precision).
- The false negatives during training were 24%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 40%(1-Recall) of the time.
- The model is an average model.

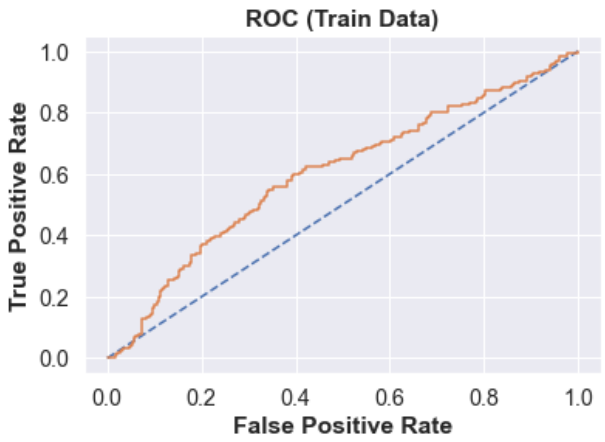
3. LDA General fitted model with polynomial features





Train Accuracy: 63.11%

Test Accuracy: 46.18%



Train AUC: 0.601

Test AUC: 0.639

Training Classification Report:

	precision	recall	f1-score	support
0	0.59	0.79	0.68	326
1	0.61	0.38	0.47	284
accuracy			0.60	610
macro avg	0.60	0.58	0.57	610
weighted avg	0.60	0.60	0.58	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.63	0.77	0.69	145
1	0.60	0.43	0.50	117
accuracy			0.62	262
macro avg	0.61	0.60	0.60	262
weighted avg	0.62	0.62	0.61	262

Train Data:

AUC: 60.1%

Accuracy: 63.11%

Precision(positive class): 61%

Test Data:

AUC: 63.9%

Accuracy: 46.18%

Precision(positive class): 60%

Recall(positive class): 38%
f1-Score(positive class): 47%

Recall(positive class): 43%
f1-Score(positive class): 50%

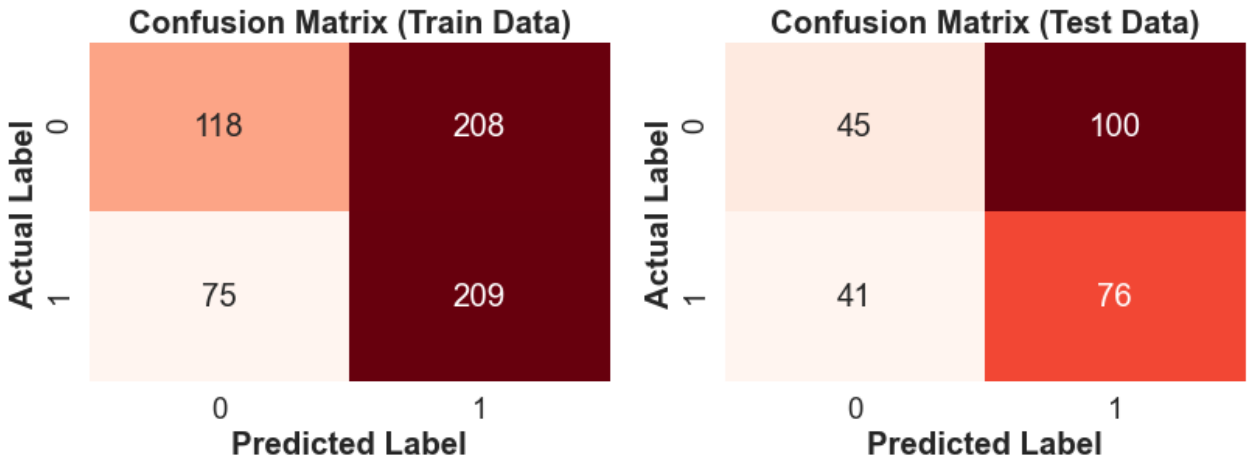
Figure 31: LDA Model 3 Metrics

(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

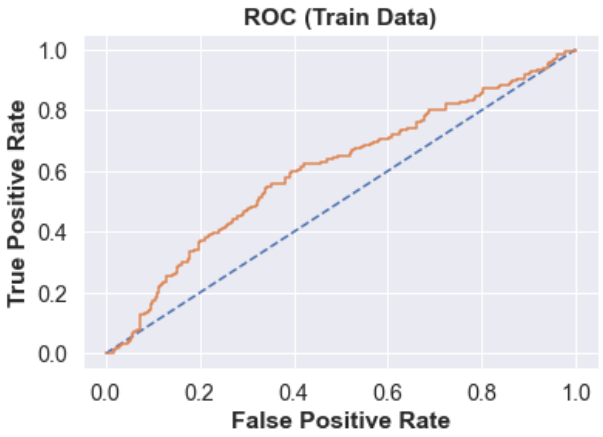
- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- As the target feature has low samples in class 'Yes'(1), we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 39%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 40% of the time(1-Precision).
- The false negatives during training were 62%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 57%(1-Recall) of the time.
- The model is an average model.

4. <u>LDA Model with polynomial features after probability threshold optimisation</u>



Train Accuracy: 53.61%

Test Accuracy: 46.18%



Train AUC: 0.601

Test AUC: 0.639

Training Classification Report:

	precision	recall	f1-score	support
0	0.61	0.36	0.45	326
1	0.50	0.74	0.60	284
accuracy			0.54	610
macro avg	0.56	0.55	0.53	610
weighted avg	0.56	0.54	0.52	610

Testing Classification Report:

	precision	recall	f1-score	support
0	0.52	0.31	0.39	145
1	0.43	0.65	0.52	117
accuracy			0.46	262
macro avg	0.48	0.48	0.45	262
weighted avg	0.48	0.46	0.45	262

Train Data:

AUC: 60.1%
Accuracy: 53.61%
Precision(positive class): 50%
Recall(positive class): 74%
f1-Score(positive class): 60%

Test Data:

AUC: 63.9%
Accuracy: 46.18%
Precision(positive class): 43%
Recall(positive class): 65%
f1-Score(positive class): 52%

Figure 32: LDA Model 4 Metrics

(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- After probability threshold optimization, we see the metrics for the positive class and negative class to be similar.

- The false positives during training were 50%(1-Precision) of the sum of true positives and false positives, while during testing there were false positives 57% of the time(1-Precision).
- The false negatives during training were 26%(1-Recall) of the sum of true negatives and false positives, while during testing, false negatives occurred 35%(1-Recall) of the time.
- The model is an average model.

Comparison

For both Logistic Regression and Linear Discriminant Analysis, following are the keys used in comparison tables:

Model 1: General fitted model

Model 2: Model after probability threshold optimisation

Model 3: General fitted model with polynomial features

Model 4: Model with polynomial features after probability threshold optimisation

Logistic Regression Models

	Accuracy	AUC	Recall	Precision	F1-score
Model 1 Train	66.72	73.5	58	66	62
Model 1 Test	65.27	71.7	52	65	58
Model 2 Train	67.54	73.5	77	62	68
Model 2 Test	64.50	71.7	75	59	66
Model 3 Train	69.67	78.2	60	71	65
Model 3 Test	68.32	69.1	62	65	64
Model 4 Train	68.52	78.2	80	63	70
Model 4 Test	66.79	69.1	81	59	69

Table 23: LR Models Metrics Comparison

Linear Discriminant Analysis Models

	Accuracy	AUC	Recall	Precision	F1-score
Model 1 Train	66.39	73.3	58	65	61
Model 1 Test	64.12	71.4	49	64	56
Model 2 Train	66.56	73.3	76	61	68
Model 2 Test	48.85	71.4	60	46	52
Model 3 Train	63.11	60.1	38	61	47
Model 3 Test	46.18	63.9	43	60	50
Model 4 Train	53.61	60.1	74	50	60
Model 4 Test	46.18	63.9	65	43	52

Table 24: LDA Models Metrics Comparison

- Accuracy speaks about the proportion of correct mapping made by the prediction model. Even more so, a test can have a high accuracy but actually perform worse than a test with a lower accuracy.
- Whilst accuracy score is fine for balanced classes,it can be very misleading for unbalanced classes.
- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.
- Recall tells what proportion of actual positives was identified correctly.
- A model that produces no false negatives has a recall of 1.0.
- Precision tells what proportion of positive identifications was actually correct.
- A model that produces no false positives has a precision of 1.0.
- It is difficult to compare two models with different Precision and Recall. So to make them comparable, we use F1-Score.

Choosing the Final Model(s)…

If we were to choose between Logistic Regression and Linear Discriminant Analysis Models, we should go for Logistic Regression as:

- The performance metrics for Logistic Regression models are better than that of Linear Discriminant Analysis models.

- Logistic Regression works better than LDA when classes are not well separated. In our case there are overlaps between classes in the general pairplot and hued pairplots. So, Logistic Regression models learned better.
- LDA works better when the data set is too small to capture all permutations and combinations of the features. In our case, there are sufficient, though not large numbers of, data points for Logistic Regression models to learn.

If we intend to reduce the false negatives, which generally should be the business requirement in this context, we can go for the Model 4 of Logistic Regression that gives the best recalls for train and test sets. If, in some context, we intend to reduce the false positives, we can go for Model 3 which gives the best precisions in both train and test sets.

If we were to choose one model, we should go for the Model 4 of Logistic Regression, which has all the performance metrics optimally high.

2.4 Inference: Based 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.

Summary:

- We started with reading the data, general information and general description for numerical and categorical data.
- We did the exploratory data analysis(univariate, bivariate and multivariate) and drew insights.
- Categorical Data were encoded.
- Data was split into train and test sets.
- Different models were built and analysed for both Logistic Regression and Linear Discriminant Analysis.
- Probability threshold optimization was done.
- Polynomial features of maximum degree 4 were added to analyse further.
- Best model was chosen based on the performance metrics.

From the logistic regression model 1 coefficients and intercept we can write the discriminant score, DS as:

DS = 2.462 + (Salary)(-1.646e-05) + (age)(-0.057) + (educ)(0.060) + (no_young_children)(-1.349) + (no_older_children)(-0.049) + (foreign)(1.266)

From the coefficients, we can conclude about the important factors:

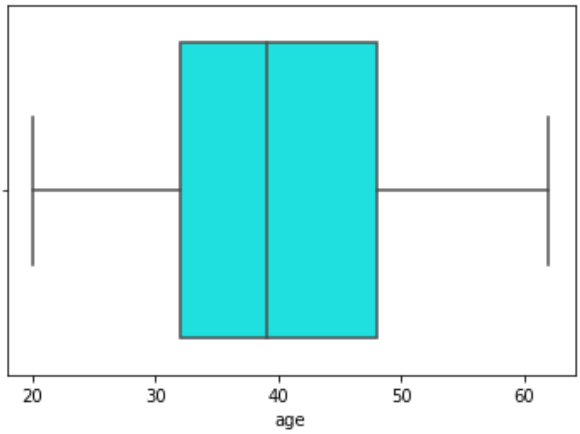
- ‘foreign’ feature has the highest positive coefficient, i.e, if a person is a foreigner, more are that one’s chances of availing a holiday package.

- feature ‘educ’ has a relatively low positive coefficient, indicating slight increase in chances of a person availing a holiday package if that one has more years of formal education.
- ‘no_young_children’ feature has the highest negative coefficient of -1.349, i.e, lesser are the chances of a person availing a holiday package if that one has more number of young children(below 7 years).
- features ‘no_older_children’ and ‘age’ have relatively lower negative coefficients, indicating lesser are the chances of a person availing a holiday package if that one has more no. of older children or has more age or both.
- ‘Salary’ feature has a very low coefficient indicating the relation between ‘Salary’ and ‘Holliday_Package’ to be not linear. Hence, nothing can be inferred about this feature from Logistic Regression.

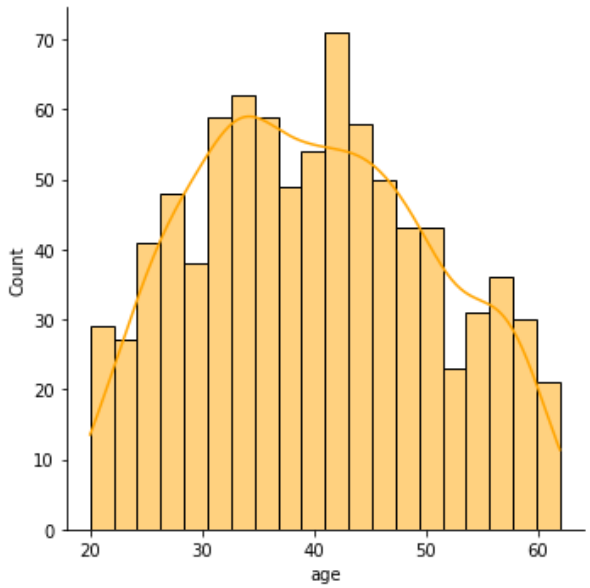
Considering above conclusions, let us try to infer more:

```
Description of age
--------------------------
count    872.000000
mean      39.955275
std       10.551675
min       20.000000
25%       32.000000
50%       39.000000
75%       48.000000
max       62.000000
Name: age, dtype: float64
```

BoxPlot of age



Distribution of age



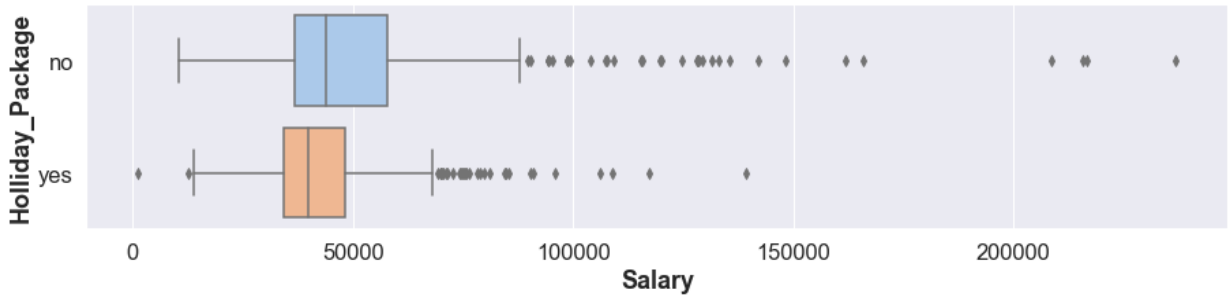


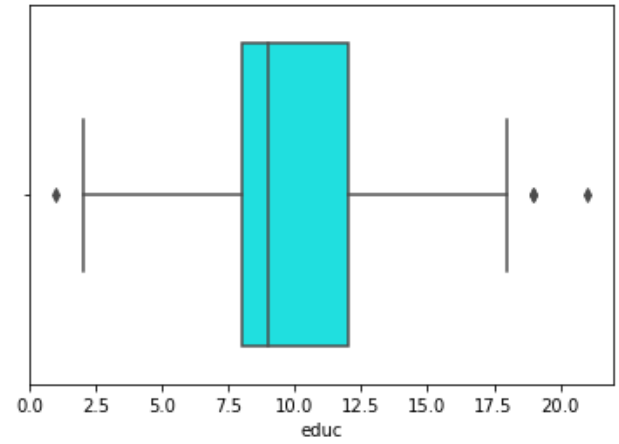
Figure 33: Univariate Analysis of 'age'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

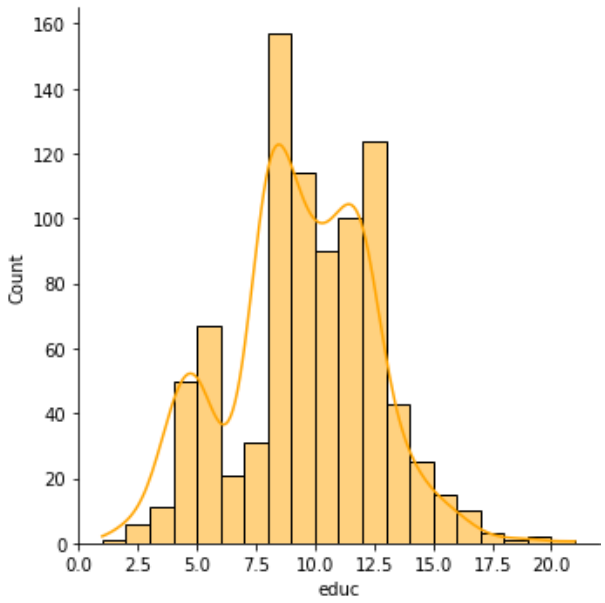
- 'age' feature had a coefficient of -0.057, indicating less the age, more the chances of availing a holiday package.
- from above conclusion, we desire our distribution of 'age' to be right skewed, i.e, more observations to be on the left side where the low age group is there.
- suitable business steps may be taken for the same.

```
Description of educ
---------------------------
count    872.000000
mean       9.307339
std        3.036259
min        1.000000
25%        8.000000
50%        9.000000
75%       12.000000
max       21.000000
Name: educ, dtype: float64
```

BoxPlot of educ



Distribution of educ



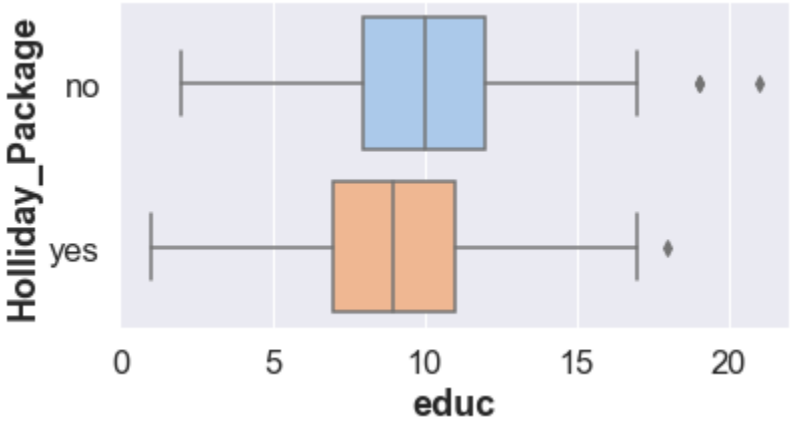
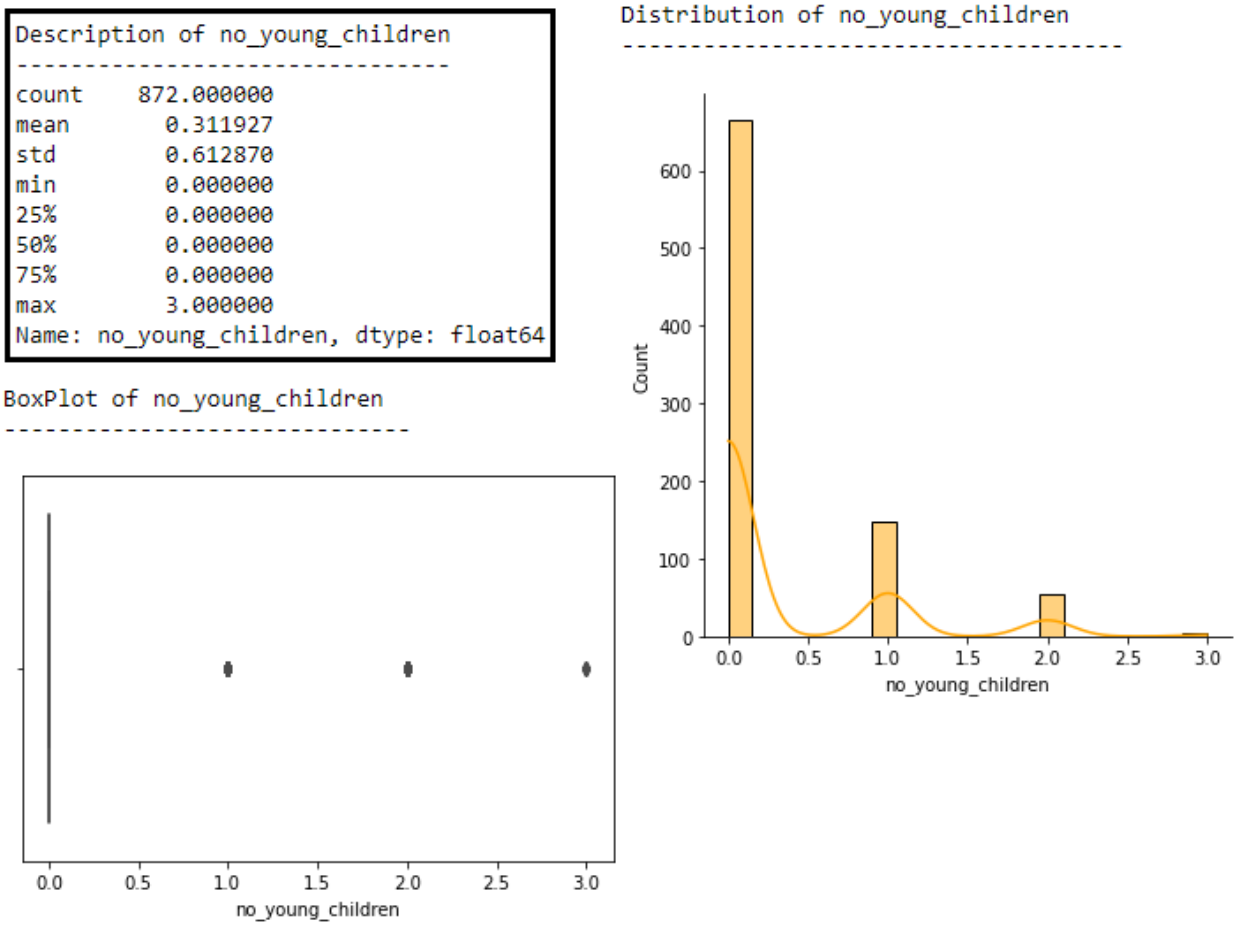


Figure 34: Univariate Analysis of 'educ'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- ‘educ’ feature had a coefficient of 0.060, indicating more the number of formal education years, more the chances of availing a holiday package.
- from above conclusion, we desire our distribution of ‘educ’ to be left skewed, i.e, more observations to be on the right side.
- suitable business steps may be taken for the same.

```
Description of no_young_children
---------------------------------
count    872.000000
mean       0.311927
std        0.612870
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        3.000000
Name: no_young_children, dtype: float64
```



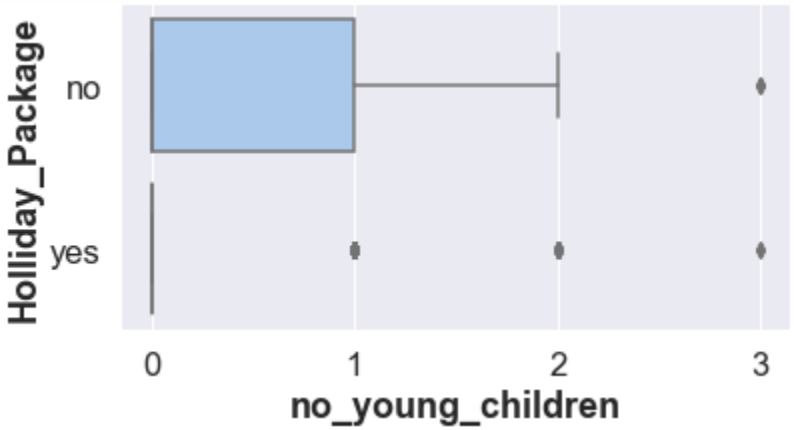


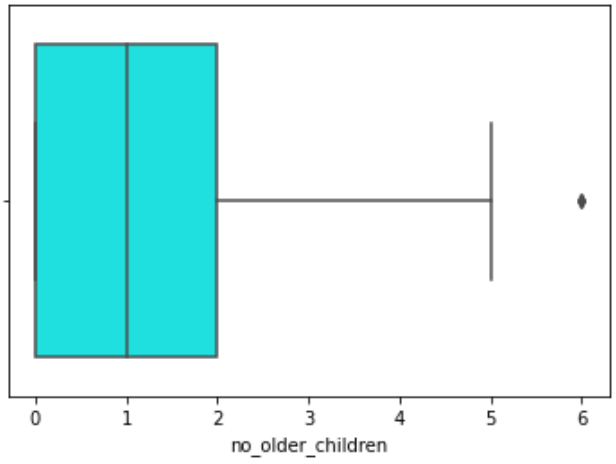
Figure 35: Univariate Analysis of ‘no_young_children’

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

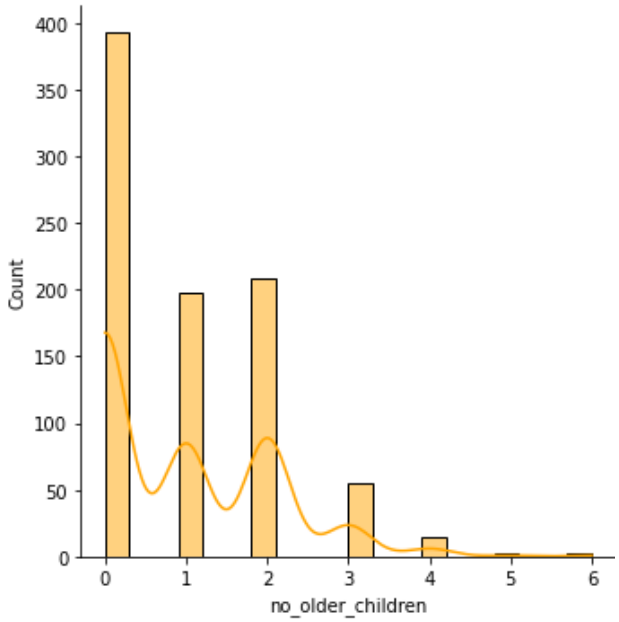
- 'no_young_children' feature has the highest negative coefficient of -1.349, i.e, lesser are the chances of a person availing a holiday package if that one has more number of young children(below 7 years).
- we desire a highly left skewed distribution, and we got a highly right skewed distribution.
- suitable business steps may be taken for the same.
- the distribution shows the feature to be actually having classes of numbers 0, 1, 2 and 3

```
Description of no_older_children
---------------------------------
count    872.000000
mean       0.982798
std        1.086786
min        0.000000
25%        0.000000
50%        1.000000
75%        2.000000
max        6.000000
Name: no_older_children, dtype: float64
```

```
BoxPlot of no_older_children
-----------------------------
```




```
Distribution of no_older_children
-----------------------------------
```



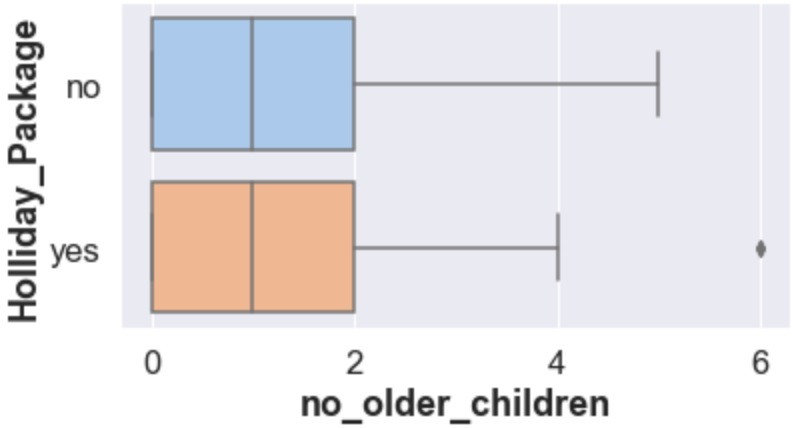


Figure 36: Univariate Analysis of 'no_older_children'

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- 'no_older_children' feature had a coefficient of -0.049, indicating less the age, more the chances of availing a holiday package.
- from above conclusion, we desire our distribution of 'no_older_children' to be right skewed, i.e, more observations to be on the left side, and we got one.

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

 HOLLIDAY_PACKAGE (normalized)
--------------------
no     0.540138
yes    0.459862
Name: Holliday_Package, dtype: float64
```

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

 FOREIGN (normalized)
--------------------
no     0.752294
yes    0.247706
Name: foreign, dtype: float64
```

Tables 25: Univariate Analysis of Categorical Data(Problem 2)

(above tables generated with '.value_counts()' function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

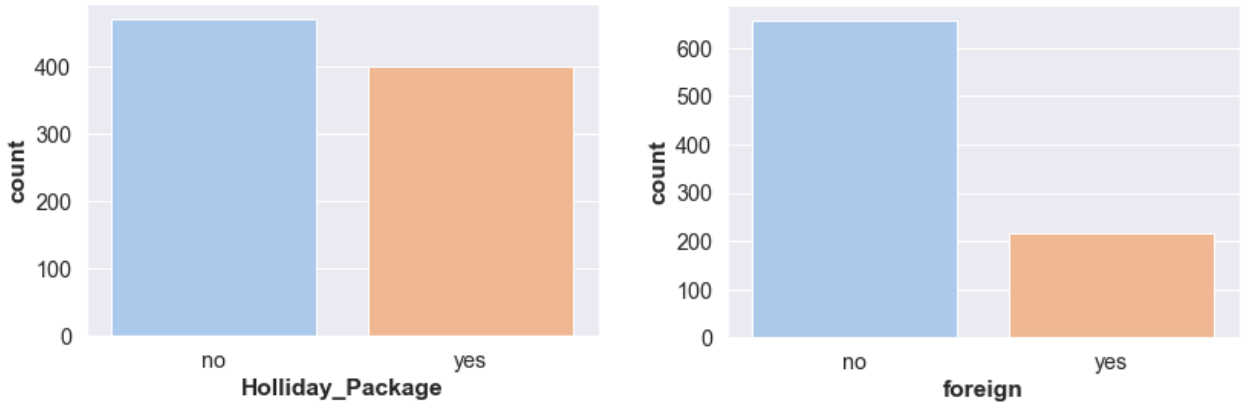


Figure 37: Count plots of Categorical Data(Problem 2)

(above plots generated using '.countplot()' function of Seaborn library in Python)

We saw:

- 'foreign' feature had the highest positive coefficient, i.e, if a person is a foreigner, more are that one's chances of availing a holiday package.
- the feature 'foreign' has imbalanced classes of 75.23% 'no' and 24.77% 'yes', yet the target feature 'Holliday_Pacakge' has approximately a 54:45 ratio of 'no' to 'yes'.

Let us visualise how the target feature 'Holliday_Package' is distributed over other categorical variable, 'foreign':

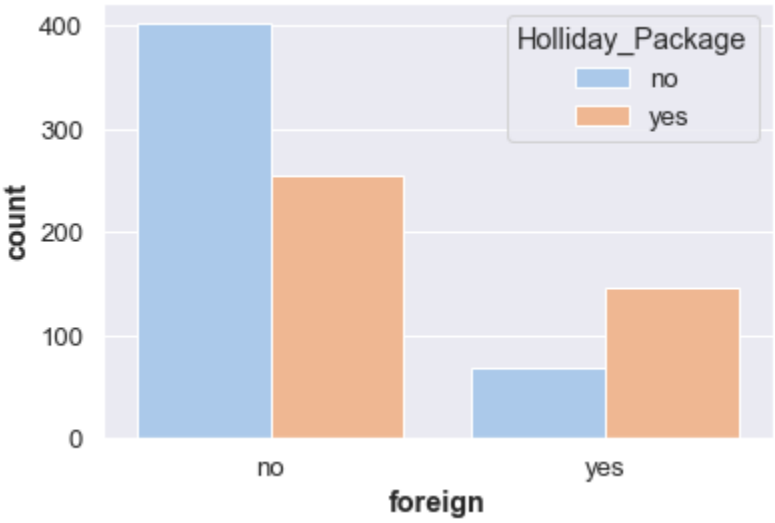


Figure 38: Distribution of Target over 'foreign'(Problem 2)

(above figure generated with '.countplot()' function of Seaborn library in Python)

We can see:

- majority of the foreigners opt for holiday packages.
- majority of the native citizens do not opt for holiday packages.

References

- The logos for ‘Great Learning’, ‘Great Lakes’ and ‘UT Austin Texas McCombs’ School of Business’ have been taken from ‘https://olympus.greatlearning.in/’
- Author grateful to creators of:
  - https://towardsdatascience.com/
  - https://medium.com
  - https://www.geeksforgeeks.org/
  - https://www.analyticsvidhya.com