

## PROJECT - PREDICTIVE MODELLING

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

### Read the data:

To start with import the libraries and load the dataset.

- Import pandas library, *import pandas as pd*
- Use read\_csv() method to read the raw data in the CSV file into a data frame, df
- Use head() method of the data frame to show the first five rows of the data.

```
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# reading the CSV file into pandas dataframe
df = pd.read_csv("cubic_zirconia.csv")
```

```
df.head()
```

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

### Exploratory Data Analysis:

Exploratory data analysis is the first step in the data analysis process to describe data that are used in future model building. It is an approach to describe and summarize the characteristics of the data especially with graphical representations.

#### **INFO () & SHAPE**

- “.info()” function gives information there are **26967 records and 11 variables in the dataset**. Hence there are 26967 cubic zirconia data with respect to 11 paramters.
- “.shape” function gives the total number of rows and columns in the data frame. i.e 26967 rows and 11 columns
- The variable ‘Unanmed: 0’ is similar to index column and it has no specific use, hence we can drop that varaible.

### Before dropping Unnamed: 0

```
df.info()
```

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

### After dropping Unnamed: 0

```
df.drop('Unnamed: 0',axis=1,inplace = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  -
0   carat        26967 non-null  float64
1   cut          26967 non-null  object
2   color        26967 non-null  object
3   clarity      26967 non-null  object
4   depth        26270 non-null  float64
5   table        26967 non-null  float64
6   x            26967 non-null  float64
7   y            26967 non-null  float64
8   z            26967 non-null  float64
9   price        26967 non-null  int64
dtypes: float64(6), int64(1), object(3)
memory usage: 2.1+ MB
```

```
df.shape
```

```
(26967, 11)
```

```
df.shape
```

```
(26967, 10)
```

## DATA TYPES

- “.info()” function also gives information about the data types of the variables present in the dataset along with the no. of entries in each variables
- .dtypes( ) method could also be used to list the data types of all the variables.

```
df.dtypes
```

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

- Out of the 10 variables in the dataset,
  - ‘carat’, ‘depth’, ‘table’, ‘x’, ‘y’, ‘z’ are of **float64 data type**
  - ‘cut’, ‘color’, ‘clarity’ are of **object data type**
  - ‘price’ belongs to **int64 data type**.

## NULL & DUPLICATE VALUES

- .info( ) method also shows that there are **some null values** present in the ‘depth’ variable of the dataset.
- .isnull( ) method gives detailed information about how many null values are present in each variable of the dataset.
- This method shows that there are **697 null values in the ‘depth’ variable** and no null values in other variables of the dataset.
- .duplicated( ) method shows the no. of duplicate rows in the dataset which could be removed.

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

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

```
# Are there any duplicates ?
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
#df[dups]
```

Number of duplicate rows = 34

## DESCRIBE ()

```
df.describe(include="all")
```

	carat	cut	color	clarity	depth	table	x	y	z	price
count	26967.000000	26967	26967	26967	26270.000000	26967.000000	26967.000000	26967.000000	26967.000000	26967.000000
unique	NaN	5	7	8	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	Ideal	G	SI1	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	10816	5661	6571	NaN	NaN	NaN	NaN	NaN	NaN
mean	0.798375	NaN	NaN	NaN	61.745147	57.456080	5.729854	5.733569	3.538057	3939.518115
std	0.477745	NaN	NaN	NaN	1.412860	2.232068	1.128516	1.166058	0.720624	<a href="#">4024.864666</a>
min	0.200000	NaN	NaN	NaN	50.800000	49.000000	0.000000	0.000000	0.000000	326.000000
25%	0.400000	NaN	NaN	NaN	61.000000	56.000000	4.710000	4.710000	2.900000	945.000000
50%	0.700000	NaN	NaN	NaN	61.800000	57.000000	5.690000	5.710000	3.520000	<a href="#">2375.000000</a>
75%	1.050000	NaN	NaN	NaN	62.500000	59.000000	6.550000	6.540000	4.040000	5360.000000
max	4.500000	NaN	NaN	NaN	73.600000	79.000000	10.230000	58.900000	31.800000	18818.000000

- “.describe()” function gives the summary of the data and returns the **count, mean, standard deviation, minimum, maximum and quantiles** (25%, 50% and 75%) of the data.
- This function gives the **Five Point Summary**
  - **Min**
  - **Q1 – 25<sup>th</sup> Percentile**
  - **Q2 – 50<sup>th</sup> Percentile (Median)**
  - **Q3 – 75<sup>th</sup> Percentile**
  - **Max**
- Here in all the numeric variables, **mean is almost near to the median**. Hence we can infer that these data follows a **normal distribution**.
- In most of the columns, there is larger difference between 75% and maximum values, hence indicating there are **many outliers in all the variables**.
- The ‘cut’, ‘clarity’ and ‘color’ are the categorical variables in the dataset. The describe function shows that there **5,7 and 8 unique values in ‘cut’, ‘color’ and ‘clarity’ respectively**.
- This method also shows the most frequently occurred categories (**i.e mode**) for each of the above categorical variables.
  - **Ideal** is the most frequently present cut category in the dataset, referring to **10816** out of 26967 entries having ‘Ideal’ as their ‘cut’ value.
  - **‘G’** is the most occurred ‘color’ category, where **5661** entries are with ‘G’ as their ‘color’
  - **‘SI1’** is the mode of the ‘clarity’, where **6571** entries are with ‘SI1’ as their ‘clarity’ type.

## VALUE COUNTS , UNIQUE & NUNIQUE

- **.unique()** method is used to check what are all the different values present in the categorical columns.
- Using this method, the unique values of the 'cut', 'color' and 'clarity' categorical columns in the dataset are listed below.

```
for col in df.columns:
    if (df[col].dtype == 'object'):
        print(col, "\t", df[col].unique())

cut      ['Ideal' 'Premium' 'Very Good' 'Good' 'Fair']
color    ['E' 'G' 'F' 'D' 'H' 'J' 'I']
clarity  ['SI1' 'IF' 'VVS2' 'VS1' 'VVS1' 'VS2' 'SI2' 'I1']
```

- **.nunique()** method is used to check the no. of unique values present in the categorical column.
- **.value\_counts()** function is used to display the no. of entries of each categories in the categorical columns( cut, color and clarity)

```
for col in df.columns:
    if (df[col].dtype == 'object'):
        print(col, " ", df[col].nunique())
        print("-----")
        print(df[col].value_counts())
        print()
```

cut      5	color    7	clarity   8
-----	-----	-----
Ideal      10805	G      5650	SI1      6564
Premium    6880	E      4916	VS2      6092
Very Good  6027	F      4722	SI2      4561
Good       2434	H      4091	VS1      4086
Fair        779	D      3341	VVS2     2530
Name: cut, dtype: int64	I      2765	VVS1     1839
	J      1440	IF        891
	Name: color, dtype: int64	I1        362
		Name: clarity, dtype: int64

- cut – **Describe the cut quality of the cubic zirconia** and the quality is in the increasing order of the Fair, Good, Very Good, Premium, Ideal.
- color - **Colour of the cubic zirconia**. With D being the best and J the worst.
- Clarity - cubic zirconia clarity refers to the absence of the Inclusions and Blemishes. (In order from Best to Worst, FL = flawless, I3= level 3 inclusions) FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3. **Here in the dataset given, the clarity ranges from Best to worst in the following order: IF,VVS1,VVS2,VS1,VS2,SI1, SI2,I1**

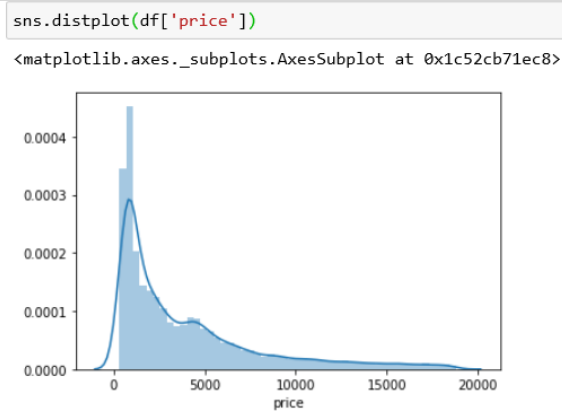
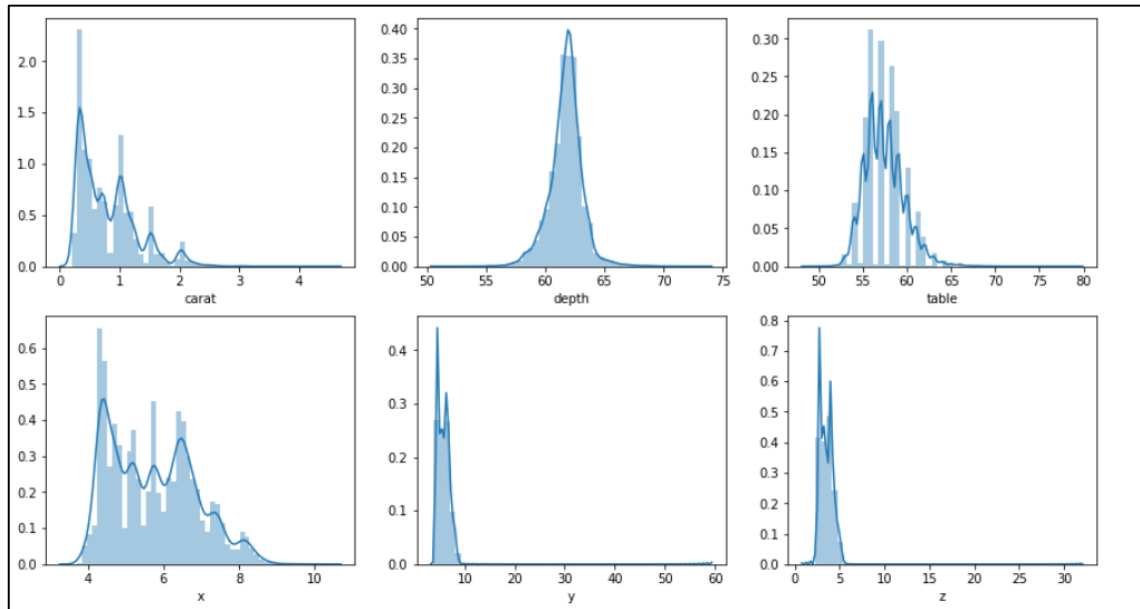
### Univariate Analysis : Analysis over a single variable

We can describe patterns found in univariate data including **central tendency** (mean, mode and median) and **dispersion**: range, variance, maximum, minimum, quartiles (including the interquartile range), and standard deviation.

#### DISTPLOT ()

- **histogram** could be drawn for the analysis of the numerical variables. Distplot() method is used in Seaborn to draw the histogram.
- The distplot flexibly plots the univariate analysis of the data and shows that most of the variables have many **peaks** in the distribution
- distplot() shows most of the variables are rightly skewed, where data is piled on left and leaving a long tail to follow in the right.
- The '**depth**' variable seems to follow **normal distribution** among the other variables.

```
fig, axes = plt.subplots(nrows=2, ncols=3)
fig.set_size_inches(15, 8)
sns.distplot(df['carat'], ax=axes[0][0])
sns.distplot(df['depth'], ax=axes[0][1])
sns.distplot(df['table'], ax=axes[0][2])
sns.distplot(df['x'], ax=axes[1][0])
sns.distplot(df['y'], ax=axes[1][1])
sns.distplot(df['z'], ax=axes[1][2])
```



## Skewness

Skewness is a measurement of the symmetry of a distribution. Therefore it describes how much a distribution differs from a normal distribution, either to the left or to the right. The skewness value can be either positive, negative or zero. The perfect normal distribution would have a skewness of zero because the mean equals the median.

```
df.skew()
```

carat	1.114871
depth	-0.025042
table	0.764890
x	0.402010
y	3.888607
z	2.639529
price	1.619055
dtype:	float64

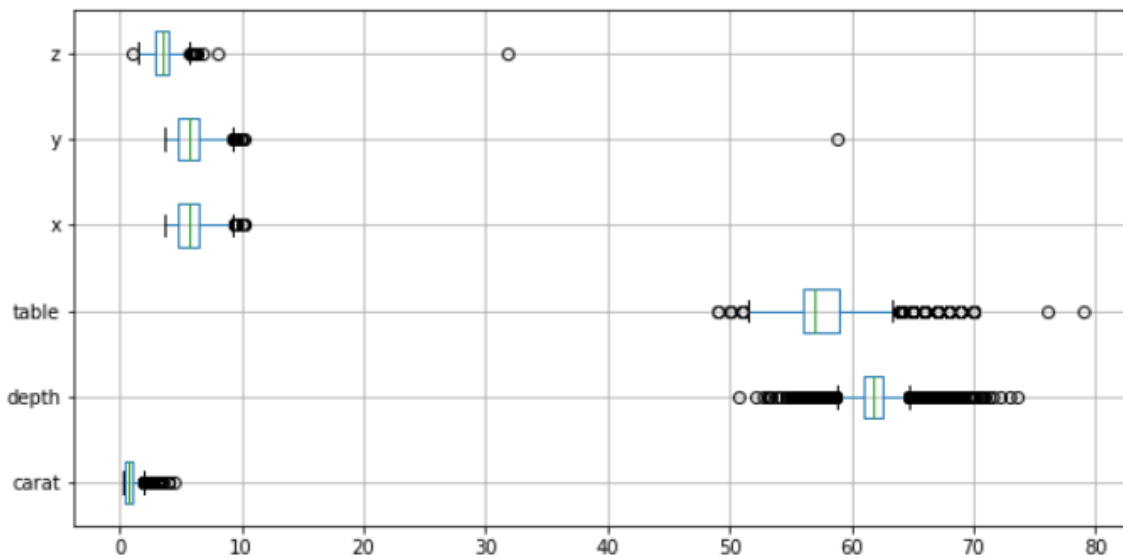
Here, in the given dataset, all the continuous variable except 'depth' are **positively skewed** as the value calculated with skew() function are greater than 0. This means the data are piled towards the left, leaving the tail pointing to the right. It is also known as right skewed.

The describe() function also shows that the 75% of each continuous variable are well below the max value, leaving the tail of the distribution extended much towards the right. Hence the distribution is **right skewed for all the continuous variable present in the dataset**.

Whereas the 'depth' variable is almost equal 0, hence it is very much **normally distributed** compared to other variables.

### BOXPLOT ( )

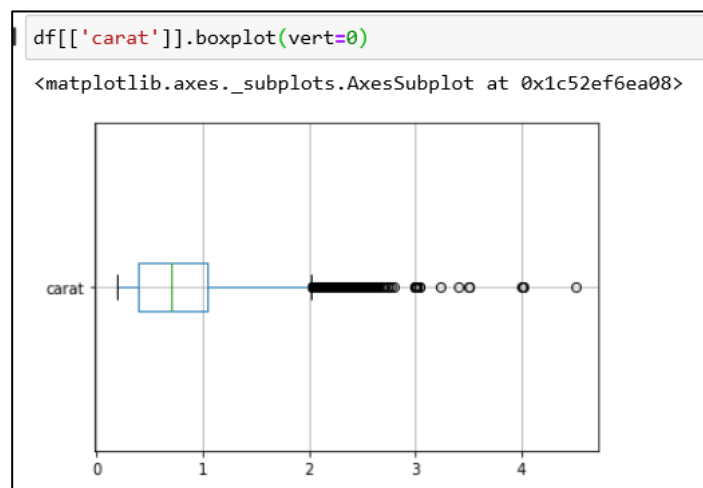
**Boxplot** is used to graphically display the dispersion of the data points in dataset. It is one way of representing the **Five Point Summary** and check if any outliers are present in the dataset. Here in the dataset **all the continuous variables have outliers in the data**, as we have seen the max value is far from 75% percentile value using describe ( ) method.



#### Five – Point Summary:

- The Minimum
- First quartile - 25<sup>th</sup> percentile (Q1)
- Second quartile or Median - 50<sup>th</sup> percentile (Q2)
- Third quartile – 75<sup>th</sup> percentile (Q3)
- The maximum

#### Analysis of 'carat':



```
df[['carat']].describe()
```

count	26925.000000
mean	0.797821
std	0.477085
min	0.200000
25%	0.400000
50%	0.700000
75%	1.050000
max	4.500000
Name: carat, dtype: float64	

The above boxplot for `df['carat']` shows the following:

- Min value of the carat is 0.20 , ie. Min=0.2
- 25%,  $Q_1 = 0.40$
- 50%,  $Q_2 = 0.70$
- 75%,  $Q_3 = 1.05$
- Max value = 4.50

**The interquartile range (IQR) is often used to find outliers in data.  $IQR = Q_3 - Q_1$ . Outliers are defined as observations that fall below  $Q_1 - 1.5 IQR$  or above  $Q_3 + 1.5 IQR$ .**

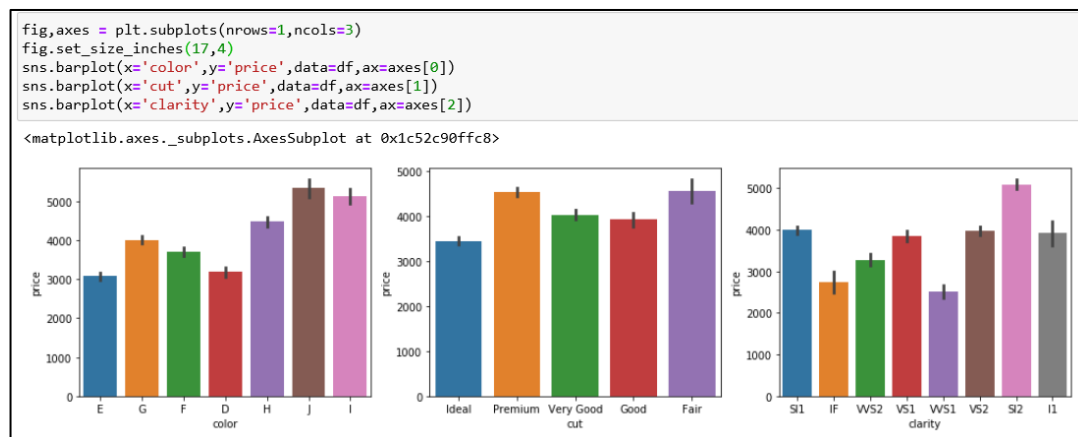
- $IQR = Q_3 - Q_1 = 1.05 - 0.40 = 0.62$
- Lower  $1.5 * IQR$  whisker =  $Q_1 - 1.5 * IQR = 0.40 - 1.5 * 0.62 = -0.73$
- Upper  $1.5 * IQR$  whisker =  $Q_3 + 1.5 * IQR = 1.05 + 1.5 * 0.62 = 1.98$
- Lower IQR whisker = -0.73, Hence all the datapoints below -0.73 are considered as outliers.
- Higher IQR whisker = 1.98, Hence all the datapoints above 1.98 are considered as outliers.

Therefore, carat value below -0.73 and above 1.98 are considered as outliers. Here, carat value doesn't tend to negative values and the minimum is 0.20, hence no outliers in the lower end. **Whereas, the maximum value present in the dataset is 4.50 and clearly it demonstrates, the presence of outliers in the dataset with respect to carat column.**

Similarly, all the other continuous variables such as 'depth', 'table', 'x', 'y' and 'z' also have outliers, as the max value of these variable is much higher than the 75<sup>th</sup> percentile, which will make the datapoints fall below  $Q_1 - 1.5 IQR$  or above  $Q_3 + 1.5 IQR$ .

## BARPLOT()

A bar plot represents an estimate of central tendency for a numeric variable with the height of each rectangle, that is a **bar plot shows only the mean value, to demonstrate the distribution of values at each level of the categorical variables.**



## Color Vs Price:

The above bar plot shows that average price of the each categories of color (D,E,F,G,H,I,J) in the 'color' column and it is clear that color 'J' seems to have highest average price and color 'E' has the lowest average price in the given dataset.

```
df.groupby('color')['price'].mean()
```

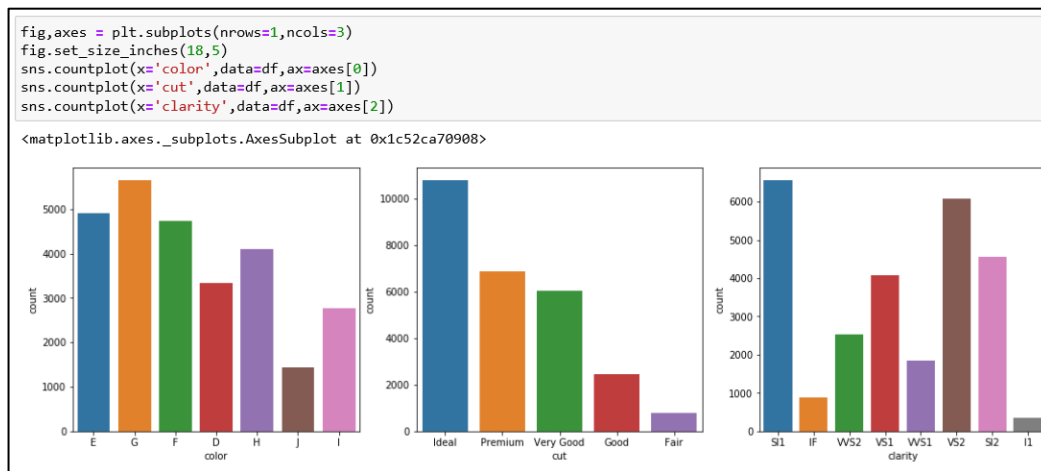
color	price
D	3184.827597
E	3073.940399
F	3700.277001
G	4004.967434
H	4469.778049
I	5124.816637
J	5329.706250

Similarly, 'Fair' cut category has the highest average price and 'S2' clarity has the highest average price in the given dataset.

df.groupby('cut')['price'].mean()		df.groupby('clarity')['price'].mean()	
cut		clarity	
Fair	4565.768935	I1	3915.013812
Good	3927.074774	IF	2739.534231
Ideal	3454.820639	SI1	3996.614564
Premium	4540.186192	SI2	5088.169919
Very Good	4032.267961	VS1	3838.130201
Name: price, dtype: float64		VS2	3963.159225
		VVS1	2502.874388
		VVS2	3263.042688
		Name: price, dtype: float64	

## COUNTPLOT ( )

Countplot is used to graphically visualise the categorical data. It shows the bar chart to display the number of occurrences for each categorical data present in the dataset.

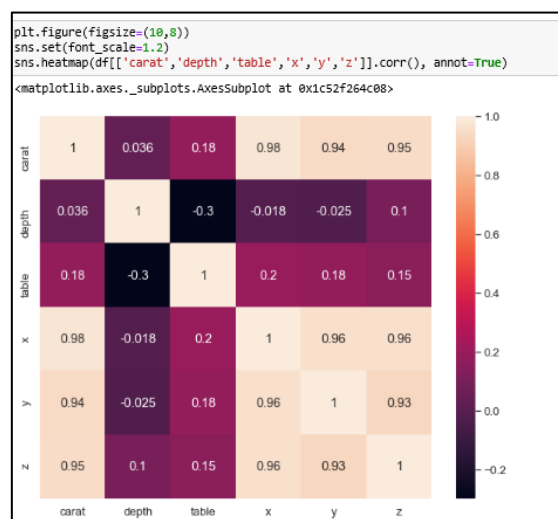


The above graph shows the number of occurrence of each categories under 'color', 'cut' and 'clarity' columns. Here, among the categories of color column (D,E,F,G,H,I,J), 'G' occurred most no. of times and 'J' is the least occurred category. Similarly under the 'cut' column, 'Ideal' type of cut is the most occurred and 'Fair' occurs the least no. of times. Among the 8 different categories, SI1 is the most occurred clarity type in the given dataset.

**Multivariate Analysis:** It is performed to understand interactions between different fields in the dataset (or) finding interactions between variables more than 2.

Ex:- Pair plot, scatter plot, Heatmap

**Heatmap:** It is used to graphically represent the correlation between each variable graphically. From the heatmap, it could be found that the variable 'x', 'y' and 'z' are highly correlated with 'carat'. The variable 'depth' is negatively correlated with 'table', 'x' and 'y'.

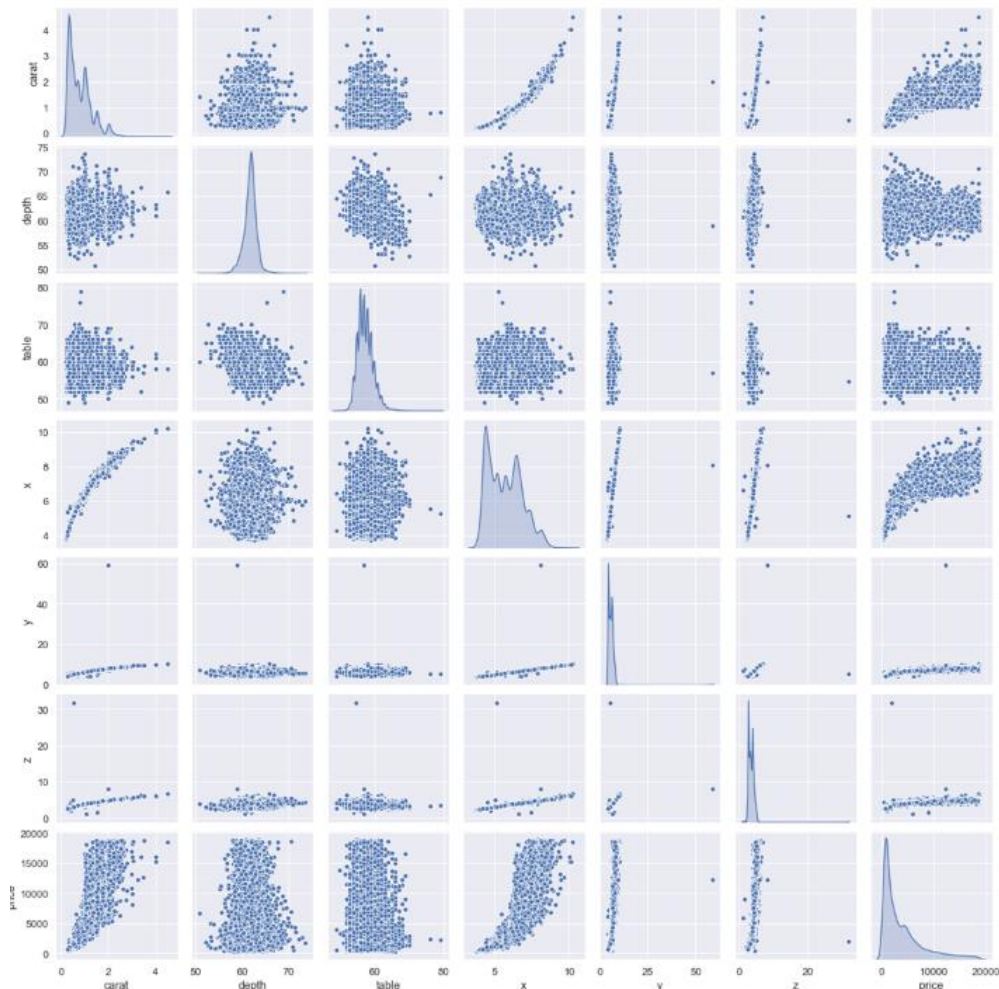




## Pair plot:

This shows a view of all variables and their relationship with all other variables. The below pairplot shows that x,y and z the variables follows high positive correlation with 'carat'. There is almost no correlation between 'depth' with other variables as the data points seems to be clustered as whole in the scatter plot. The target 'price' variable has the positive correlation with 'carat' and 'x' variables. The variables 'depth' and 'table' follow a negative correlation.

**sns.pairplot(df)**



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

## Null & Duplicate Values:

- With the EDA done, we have found that there are null and duplicate values present in the dataset.
- It is found that there are 34 duplicate rows and all the duplicate data found should be dropped from the dataset.
- The 'depth' variable has 697 null values, hence we try to impute those null values with 'Median'.
- As already found that the 'depth' variable has more outliers, we have chosen to impute the null values with 'Median'.

```
print('Before', df.shape)
# write the code to drop duplicates
df.drop_duplicates(inplace=True)
print('After', df.shape)
Before (26967, 10)
After (26933, 10)
```

```
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
Number of duplicate rows = 0
```

```
df['depth']=df['depth'].fillna(df['depth'].median())

df.isnull().sum()

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

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

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

With the describe() function, we have seen that the min value of 'x', 'y' and 'z' is 0. Whereas,

- x – Length of the cubic zirconia in mm
- y – Width of the cubic zirconia in mm
- z – Height of the cubic zirconia in mm

**A value of '0' for any of these three variables is meaningless, as the dimensions of the diamond cannot be '0' and only 8 rows are with such data. Hence dropping these 8 rows will not have much impact on the model to be built.**

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

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

carat cut color clarity depth table x y z price
df.shape
(26925, 10)
```

### Scaling:

In Multiple Linear regression, it is often **recommended to center the variables** so that the predictors have mean 0. Standardizing the features around the center with mean of 0 with a standard deviation of 1 is important **when we compare measurements that have different units**.

For example, A variable that ranges between 0 and 10000 will outweigh a variable that ranges between 0 and 1. Using these variables without standardization will give the variable with the larger range weight of 1000 in the analysis. **Transforming the data to comparable scales can prevent this problem. Typical data standardization procedures equalize the range and/or data variability.**

In the given dataset,

- The range of 'carat' variable is between 0.20 to 4.5.
- 'depth' and 'table' variables vary between 49 and 80
- When the categorical variables are encoded with One-hot encoding technique, the dummy variables will have 0 and 1.

**Hence it is recommended to scale the data to equalize the range and data variability.**

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

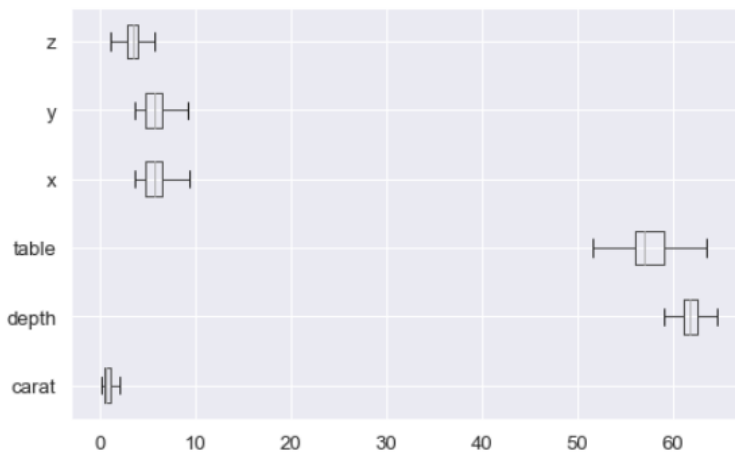
### **Outlier Treatment:**

Linear regression models are not robust to outliers. Outliers can affect both the result and assumptions. Boxplot( ) shows that there are outliers in all the variables of the dataset. Hence, before building the model, it is very important to treat the outliers.

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

```
for column in ['carat','depth','table','x','y','z']:  
    lr,ur=remove_outlier(df[column])  
    df[column]=np.where(df[column]>ur,ur,df[column])  
    df[column]=np.where(df[column]<lr,lr,df[column])
```

```
plt.figure(figsize=(8,5))  
df[['carat','depth','table','x','y','z']].boxplot(vert=0)  
  
<matplotlib.axes._subplots.AxesSubplot at 0x1c53a2f2508>
```

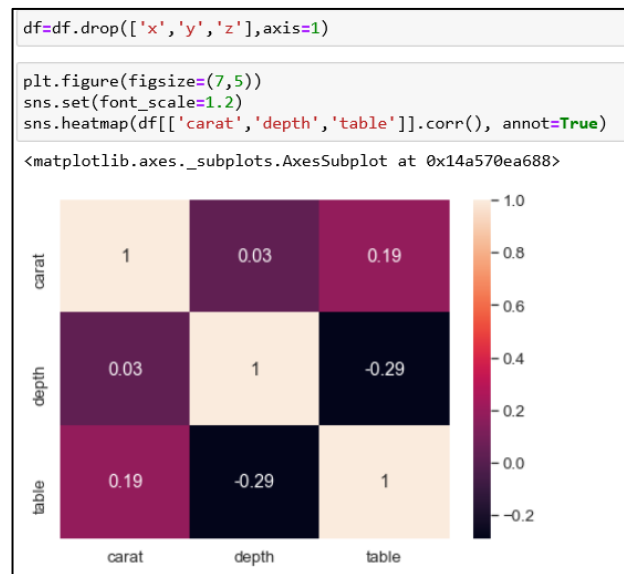


### **Remove Multicollinearity:**

Multicollinearity occurs when independent variables in a regression model are correlated. In regression, "multicollinearity" refers to predictors that are correlated with other predictors. **One of the assumptions of the linear regression is lack of perfect multicollinearity in the predictors.** In other words, it results when you have factors that are a bit redundant.



The above heatmap shows that 'x', 'y' and 'z' variables are highly correlated with 'carat'. And 'x','y' 'z' are highly correlated with each other too. It is only when the correlation is so strong that they do not convey extra information. Hence we can remove these variables.



Other way to measure multi-collinearity is the variance inflation factor (VIF), which assesses how much the variance of an estimated regression coefficient increases if your predictors are correlated. If no factors are correlated, the VIFs will all be 1.

### Encode the data:

```
# Create dummy variables in the same dataset df
df=pd.get_dummies(df,columns=['cut','color','clarity'])
```

```
df.head()
```

	carat	depth	table	price	cut_Fair	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	color_D	...	color_I	color_J	clarity_I1	clarity_IF	clarity_SI1	clarity_
0	0.30	62.1	58.0	499	0	0	1	0	0	0	...	0	0	0	0	1	
1	0.33	60.8	58.0	984	0	0	0	1	0	0	...	0	0	0	1	0	
2	0.90	62.2	60.0	6289	0	0	0	0	1	0	...	0	0	0	0	0	
3	0.42	61.6	56.0	1082	0	0	1	0	0	0	...	0	0	0	0	0	
4	0.31	60.4	59.0	779	0	0	1	0	0	0	...	0	0	0	0	0	

5 rows × 24 columns

Some of the features in the dataset are categorical and do not fit into a linear regression model. All the independent variables are required to be in numerical form for Linear Regression. Hence, we encode the categorical columns in the dataset.

Label Encoding makes the categories randomly as 0,1,2,3.... Now since  $0 < 1 < 2$ , the equations in your regression model may consider one category has a higher value than the other, which is of course not true.

To solve this situation we have a concept called **Dummy variables**. In regression analysis, a dummy variable is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. This is called **One – Hot Encoding**.

**Since the categorical columns have ordinal values, we could map directly map the values to the categories based on its increasing order or use One – Hot Encoding.**

**One-hot encoding** is a great tool for turning some of these categorical features into multiple binary features; the presence or absence of the individual categorical unit can then be fit into the linear regression.

**.get\_dummies( )** method is used to get the dummies for the categorical column such as 'cut', 'color' and 'clarity' in the dataset.

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

```
x=df.drop(['price'],axis=1)
y=df.price
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=1,test_size=0.30)
```

Separate the independent and dependent variables into 2 dataframes say x and y respectively. The target dependent variable '**price**' is in the dataframe 'y' and all other independent variables are in 'x'

Using the train\_test\_split function imported from sklearn.model\_selection, the dataframe x(independent variables) and y(dependent variable) is split into training and testing data with ratio of 70:30 respectively.

- **X\_train** : Training data of the independent variables. From 26925 records, the values of 23 independent variables of 18847 records are taken into x\_train.
- **X\_test** : Testing data of the independent variables. The remaining 8078 records are taken as testing data into x\_test
- **y\_train** : Training data of dependent variables. The target variable 'price' of 18847 records are considered as y\_train
- **y\_test** : Testing data of dependent variables and the target variable 'price' of 8078 records are considered as y\_test

### Scaling:

```
from sklearn.preprocessing import MinMaxScaler
ss=MinMaxScaler()
# we are scaling the data for Linear Regression.
x_train_scaled=ss.fit_transform(x_train)
x_test_scaled=ss.transform(x_test)
```

**Normalization** is a scaling technique in which values are shifted and rescaled so that they end up ranging between **0 and 1**. It is also known as **Min-Max scaling**. To normalize the data, we need to import the **MinMaxScaler** from the sklearn library and apply it to our dataset.

Since our train and test data have one-hot encoded features, it better to apply **MinMaxScaler()** than **StandardScaler()**, as all the variables will range from 0 to 1.

### Apply Linear Regression:

```
# invoke the LinearRegression function and find the bestfit model on training data
regression_model = LinearRegression()
regression_model.fit(x_train_scaled, y_train)

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

- Using a **LinearRegression()** method imported from the package **sklearn.linear\_model**, the Linear Regression model is built.
- Fit the model with the training data. Pass the parameters **x\_train\_scaled** and **y\_train** for **fit()**

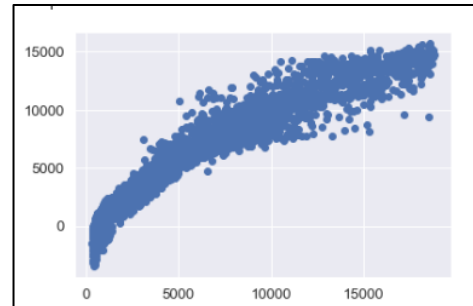
### Prediction on Test data:

After the model is built using `LinearRegression()`, we could make predictions of the target variable of test data (`y_test`) with the help of scaled independent variables of the test data (`x_test_scaled`).

The below scatter plot shows the linear model built between the original test data of the target values (`y_test`) and predicted target values (`y_pred`)

```
# Prediction on Test data
y_pred = regression_model.predict(x_test)

plt.scatter(y_test, y_pred)
```



### Performance Metrics:

#### ➤ R-square

- **R-squared** is a goodness-of-fit measure for linear regression models.
- This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively.
- R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 – 100% scale.
- It is a statistical measure of how close the data are to the fitted regression line. **It is also known as the coefficient of determination.**
- **Both the train and test data has R-square value of 91% and in general, the higher the R-Squared, the better the model fits our data.**

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

0.9156675453075128

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

0.9184175323124976
```

#### ➤ MSE

The Mean Squared Error (MSE) tells us how close a regression line is to set of points. It is calculated by measuring the distances from the data points to the regression line and squaring to them to remove any negative values.

General steps to calculate the mean squared error from a set of X and Y values:

- Find the regression line.
- Insert your X values into the linear regression equation to find the new Y values (Y').
- Subtract the new Y value from the original to get the error.
- Square the errors.
- Add up the errors.
- Find the mean.

```
# Calculate train MSE
import math
mse = np.mean((regression_model.predict(x_train_scaled)-y_train)**2)
math.sqrt(mse)

1163.944335362369

# Calculate test MSE
import math
mse = np.mean((regression_model.predict(x_test_scaled)-y_test)**2)
math.sqrt(mse)

1156.9761425818556
```

**Mean\_squared\_error( )** method imported from the package `sklearn.metrics` could be used to calculate the average of a set of errors in the train and test data. Here we look at the root mean squared error (RMSE) which is **1156.97**

```
# MSE of train data
np.sqrt(mean_squared_error(y_train,regression_model.predict(x_train_scaled)))

1163.944335362369

# MSE of test data
np.sqrt(mean_squared_error(y_test,regression_model.predict(x_test_scaled)))

1156.9761425818556
```

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

The estimated values of the 'price' are calculated using the regression equation and the co-efficient values of each explanatory variable that we have used for preparing the model. **These co-efficients** tells us how each variable are impactful in determining the price of the diamond.

```
# build Linear regression using ols()

import statsmodels.formula.api as smf
lm1 = smf.ols(formula= expr, data = data_train).fit()
lm1.params

Intercept      -113.956360
carat           9203.975185
depth          -25.056148
table          -36.487581
cut_Fair       -590.288385
cut_Good       -48.701502
cut_Ideal      244.956685
cut_Premium    153.505394
cut_Very_Good  126.571448
color_D         798.044658
color_E         603.023745
color_F         475.254029
color_G         270.194699
color_H        -198.079176
color_I        -680.450208
color_J       -1381.944107
clarity_I1     -3397.945465
clarity_IF     1546.578322
clarity_SI1    -346.019656
clarity_SI2   -1262.843926
clarity_VS1     641.912190
clarity_VS2     336.337884
clarity_VVS1   1241.161162
clarity_VVS2   1126.863130
dtype: float64
```

From the above listed co-efficients of the independent variable, we could see the most impactful variables in predicting the price of cubic zirconia, in terms of magnitude are :

- **Carat**
- **clarity\_I1: cubic zirconia with clarity I1**
- **clarity\_IF: cubic zirconia with clarity IF**
- **'color\_J' : zirconia with color type 'J'**
- **'clarity\_SI2': zirconia with clarity SI2**

These are the 5 most impactful variables in predicting the price of the cubic zirconic. As we could see, **the co-efficient of 'Carat' is positive with a magnitude of 9203.97, and it denotes that a unit increase in the carat increases the price of zirconia by 9203.97**

**'clarity\_I1'** is the next most impactful and important factor in predicting the price of the zirconia. **As this variable denotes that zirconia with least clarity type I1, the co-efficient of 'clarity\_I1' is negative with a high magnitude of 3397. Hence, this variable has a negative impact on the price and with this clarity, it tends to decrease the price of the zirconia.**

**'clarity\_IF'** is next in the list of most impactful variables in predicting the price. **As we know, 'clarity\_IF' denotes the best clarity of the zirconia in the given dataset, the co-efficient of this variable is also with higher magnitude and positive. Hence, the price of the zirconia tends to increase by 1546 units with one unit change in the 'clarity\_IF' attribute.**

**'depth' and 'table' doesnot have much impact on the price of the zirconia as their magnitues are very small compared to other varaibles.**



## 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.

### Read the data:

To start with import the libraries and load the dataset.

- Import pandas library, *import pandas as pd*
- Use read\_csv method to read the raw data in the CSV file into a data frame, data
- Use head() method of the data frame to show the first five rows of the data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score, roc_curve, classification_report, confusion_matrix
```

```
data=pd.read_csv("Holiday_Package.csv")
```

```
data.head()
```

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

### Exploratory Data Analysis:

Exploratory data analysis is the first step in the data analysis process to describe data that are used in future model building. It is an approach to describe and summarize the characteristics of the data especially with graphical representations.

#### INFO () & SHAPE

- “.info()” function gives information there are **872 records and 8 variables in the dataset**. Hence there are 872 records of employee data with respect to 8 paramters.
- “.shape” function gives the total number of rows and columns in the data frame. i.e 872 rows and 8 columns
- The variable ‘Unanmed: 0’ is similar to index column and it has no specific use, hence we can drop that variable. Hence after removing ‘Unamed: 0’, there will be 872 observations and 7 records.

```
data.drop('Unamed: 0',axis =1,inplace = True)
```

```
data.shape
```

```
(872, 7)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 872 entries, 0 to 871
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   Holliday_Package      872 non-null   object  
1   Salary                872 non-null   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
```

## DATA TYPES

- “.info()” function also gives information about the data types of the variables present in the dataset along with the no. of entries in each variables
- .dtypes() method could also be used to list the data types of all the variables.
- From the above use of info() function,
  - 5 variables (Salary, age, educ, no\_young\_children, no\_older\_children) are **integer data types**
  - 2 variables (Holliday\_package, foreign) are **object data types**.
- The target dependent variable ‘Holliday\_Package’ is expected to be an **categorical data** to perform Logistic Regression.
- The independent variables in the dataset should be numeric values to build the Logistic Regression and LDA.

## NULL & DUPLICATE VALUES

- .info() method also shows that there are **no null values** in the dataset.
- .isnull() method gives detailed information about how many null values are present in each variable of the dataset.
- .duplicated() method shows the no. of duplicate rows in the dataset which could be removed. In the given dataset, there are **no duplicate rows**.

```
data.isnull().sum()
```

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

```
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print(data.shape)
```

```
Number of duplicate rows = 0
(872, 7)
```

## VALUE\_COUNTS & UNIQUE

- **unique()** method is used to check what are all the different values present in the categorical columns.
- Using this method, the unique values of the ‘Holliday\_Package’ and ‘foreign’ categorical columns in the dataset are listed below.
- **value\_counts()** function is used to display the no. of entries of each categories in the categorical columns(‘Holliday\_Package’ and ‘foreign’)

```
data['Holliday_Package'].unique()
array(['no', 'yes'], dtype=object)
```

```
data['foreign'].unique()
array(['no', 'yes'], dtype=object)
```

```
data['Holliday_Package'].value_counts()
```

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

```
data['foreign'].value_counts()
```

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

## Descriptive Statistics:

Descriptive statistical Analysis is performed to describe, show or summarize data in a meaningful way. It also describes the calculation of various measures such as:

- Measure of central tendency (Mean, Median, Mode)
- Measures of variability (Range, IQR, Variance, Standard Deviation)
- Five Point Summary (Min, Three quartiles, Max)
- Skewness

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

- “.describe()” function gives the summary of the data and returns the **count, mean, standard deviation, minimum, maximum and quantiles (25%, 50% and 75%)** of the data.
- This function gives the **Five Point Summary**
  - **Min**
  - **Q1 – 25<sup>th</sup> Percentile**
  - **Q2 – 50<sup>th</sup> Percentile (Median)**
  - **Q3 – 75<sup>th</sup> Percentile**
  - **Max**
- Here in all the numeric variables, **mean is almost near to the median**. Hence we can infer that these data would follow a **normal distribution**.
- In most of the columns, there is larger difference between 75% and maximum values, hence indicating there are **outliers in all the variables**.
- The ‘Holliday\_Package’ and ‘foreign’ are the **categorical variables** in the dataset. The describe function shows that there **2 unique values in both variables**.
- This method also shows the most frequently occurred categories (**i.e mode**) for each of the above categorical variables. That is ‘no’ is the mode of these 2 categorical variables.
- As we know:
  - no\_young\_children -> no. of younger children
  - No\_older\_children -> no. of older children

Calculating mean and median would be meaningless, as no. of children couldn't be a decimal number.

**Hence, considering these variables as categorical data, would give us better inference of the dataset with respect to the target variable ‘Holliday\_Package’**

## Measures of Central Tendency (Mean, Median, Mode)

**Mean** is the average of all the values in a continuous variable. For example : Consider the “Salary” variable from the given dataset, “data”. The mean of “Salary” is obtained by the code: `data['Salary'].mean()`. Similarly, the mean of all the continuous variable could be seen in the **mean** attribute obtained using the `data.describe()` function

```
data['Salary'].mean()
47729.172018348625
```

Median is the middle value in the sorted list of numbers in the continuous variable. The median of the ‘Salary’ variable could be obtained with `data['Salary'].median()`. It could also be seen in the **50%** attribute obtained using the `data.describe()` function

```
data['Salary'].median()
41903.5
```

**Mode** gives the most frequently occurred value of the variable in the dataset. Most occurred value in the ‘foreign’ variable in the dataset ‘data’ is obtained with `data['foreign'].mode()`. The mode could be calculated for all the categorical variables to determine the most occurred value of that variable. ‘top’ and ‘freq’ attribute of describe() function gives the most occurred value and its frequency respectively.

- Mode of ‘Holliday\_Package’ id ‘no’ as it occurs for 471 out 872 records in the dataset.
- Mode of ‘foreign’ is ‘no’ as it occurs for 656 out 872 records in the dataset.

```
data['foreign'].mode()
0      no
dtype: object
```

## Measures of variability (Range , IQR, Variance, Standard Deviation)

The most popular variability measures are the range, interquartile range (IQR), variance, and standard deviation. These are used to measure the **amount of spread or variability within your data**.

### Range

The range is the measure of spread and it is obtained by computing the difference between the largest observed value of the variable in a dataset and the smallest one in the dataset.

$$\text{Range} = \text{max} - \text{min}$$

The max and min attribute of `‘data.describe()’` gives the largest value and smallest value of each continuous variable present in the dataset.

### Analysis on Range

- The maximum age of the person in the dataset is 62 and the minimum age in the dataset is 20. Therefore, **the age is spread between 20 and 62, defining the range as 42.**
- Similarly, the maximum Salary of the employee in the dataset is ‘236961’ whereas the min Salary is ‘1322’. Hence Salary of the employee ranges between 1322 and 236961.
- The education of the employees is spread between 1 and 21 years.
- The no. of younger children ranges between 0 and 3 whereas the no. of older children is spread over 0 and 6 for the employees in the dataset.

## Inter Quartile Range (IQR)

IQR is the statistical dispersion between the upper 75<sup>th</sup> quartile (Q3) and lower 25<sup>th</sup> quartile(Q1). **While range measures beginning and end of the datapoints, IQR measures where majority of the values lie.**

$$\text{IQR} = Q_3 - Q_1$$

**Analysis on IQR** - Consider 'Salary' variable:

- lower (25%) quartile  $Q_1 = 35324$
- median (50%)  $Q_2 = 41903$
- upper (75%) quartile  $Q_3 = 53469$
- interquartile range,  $\text{IQR} = Q_3 - Q_1 = 18145$
- lower 1.5\*IQR whisker =  $Q_1 - 1.5 * \text{IQR} = 35324 - 18145 = 17179$
- upper 1.5\*IQR whisker =  $Q_3 + 1.5 * \text{IQR} = 53469 + 18145 = 71614$

**The interquartile range is often used to find outliers in data. Outliers are defined as observations that fall below  $Q_1 - 1.5 \text{ IQR}$  or above  $Q_3 + 1.5 \text{ IQR}$ .**

Therefore, any Salary below 17179 and above 71614 are considered as outliers. Here, the maximum Salary present in the dataset is 236961 and clearly it demonstrates, the presence of outliers in the dataset with respect to Salary column.

Similarly, all the other continuous variables such as 'age', 'educ', 'no\_older\_children' and 'no\_young\_children' also have some outliers, as the max value of these variable is much higher than the 75<sup>th</sup> percentile, which will make the datapoints fall **below  $Q_1 - 1.5 \text{ IQR}$  or above  $Q_3 + 1.5 \text{ IQR}$ .**

## Variance and Standard Deviation

Variance and Standard deviation, also measure the spread of data. The variance of each continuous variable is found by the `var()` function.

The variance is computed by finding the difference between every data point and the mean, squaring them, summing them up and then taking the average of those numbers. **The problem with Variance is that because of the squaring, it is not in the same unit of measurement as the original data.**

Standard Deviation is used more often because it is in the original unit. It is simply the square root of the variance and because of that, it is returned to the original unit of measurement.

<pre>data['Salary'].var()</pre>	<pre>data['Salary'].std()</pre>
548434035.7683095	23418.66853107387
<pre>data['age'].var()</pre>	<pre>data['age'].std()</pre>
111.33783666354182	10.551674590487607

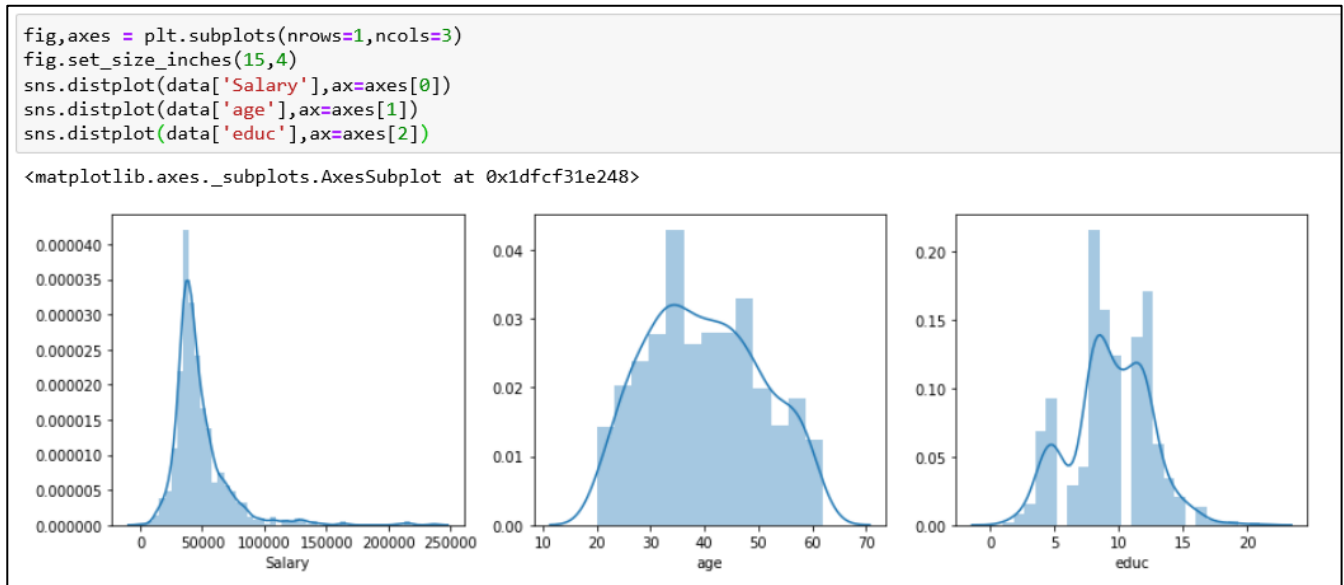
When you have a low standard deviation, your data points tend to be close to the mean. A high standard deviation means that your data points are spread out over a wide range. Hence, there is deviation of around 10.55 from the mean age value of 40.

## Univariate Analysis - Analysis over a single variable

We can describe patterns found in univariate data including **central tendency** (mean, mode and median) and **dispersion**: range, variance, maximum, minimum, quartiles (including the interquartile range), and standard deviation.

### DISTPLOT ()

- **histogram** could be drawn for the analysis of the numerical variables. Distplot() method is used in Seaborn to draw the histogram.
- The below histogram shows that the 'age' and 'educ' are almost **normally distributed**.
- 'Salary' is **right skewed** as the datapoints are piled on left with a long tail on the right side.



### Skewness

Skewness is a measurement of the symmetry of a distribution. Therefore it describes how much a distribution differs from a normal distribution, either to the left or to the right. The skewness value can be either positive, negative or zero. The perfect normal distribution would have a skewness of zero because the mean equals the median.

```
data.skew()
```

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

Here, in the given dataset, all the continuous variable except 'educ' are **positively skewed** as the value calculated with skew() function are greater than 0. This means the data are piled towards the left, leaving the tail pointing to the right. **It is also known as right skewed.**

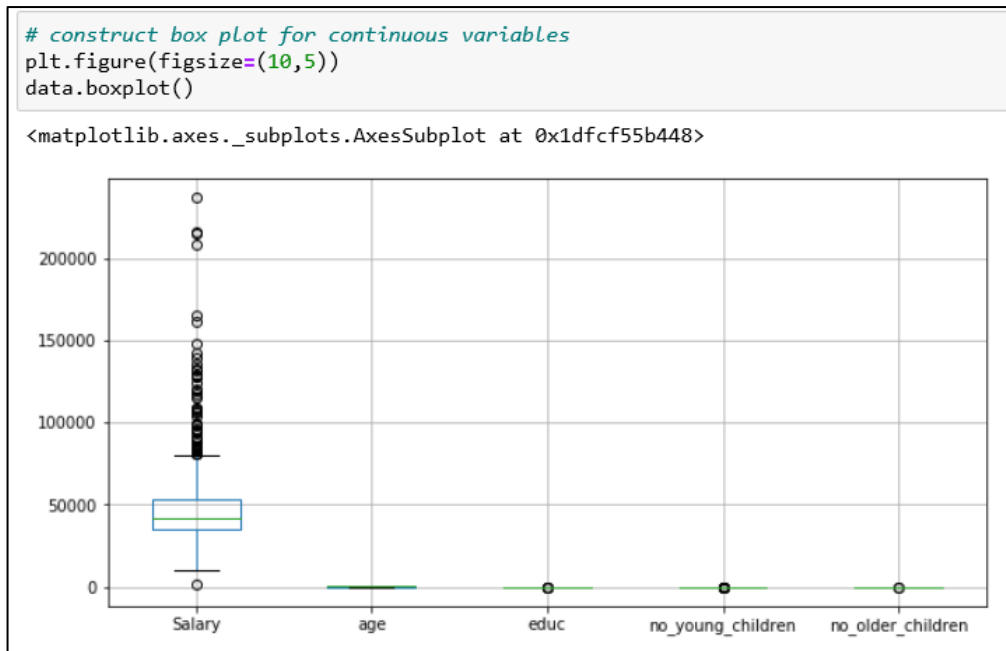
Eventhough 'educ' variable is **negatively skewed**, it is **almost equal 0**, hence it is very much **normally distributed** compared to other variables.

## BOXPLOT ( )

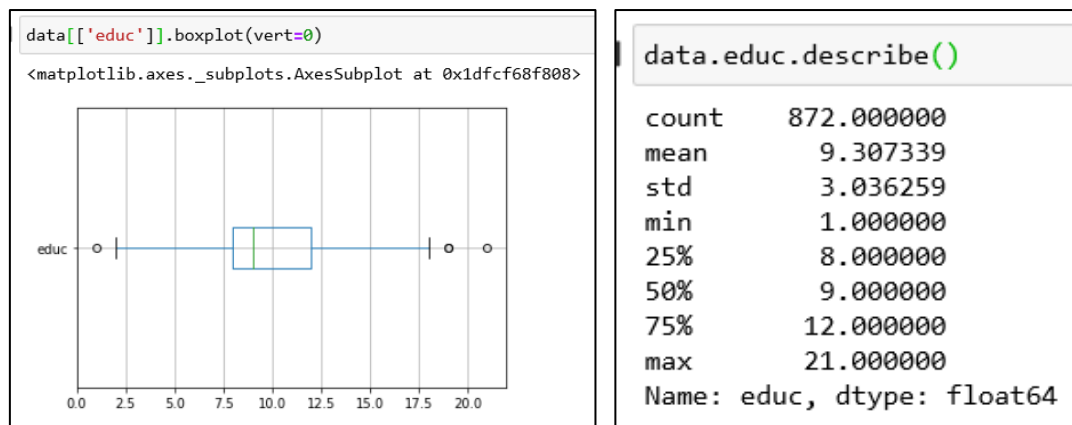
**Boxplot** is used to graphically display the dispersion of the data points in dataset. It is one way of representing the **Five Point Summary** and check if any outliers are present in the dataset. Here in the dataset **almost all the continuous variables have outliers in the data**, as we have seen the max value is far from 75% percentile value using describe ( ) method.

### Five – Point Summary:

- The Minimum
- First quartile - 25<sup>th</sup> percentile (Q1)
- Second quartile or Median - 50<sup>th</sup> percentile (Q2)
- Third quartile – 75<sup>th</sup> percentile (Q3)
- The maximum



### Analysis of ‘educ’:



The above boxplot for data[‘educ’] shows the following:

- Min value = 1
- 25%, Q1 = 8
- 50%, Q2 = 9
- 75%, Q3 = 12
- Max value = 21

**The interquartile range (IQR) is often used to find outliers in data.  $IQR = Q_3 - Q_1$ . Outliers are defined as observations that fall below  $Q_1 - 1.5 IQR$  or above  $Q_3 + 1.5 IQR$ .**

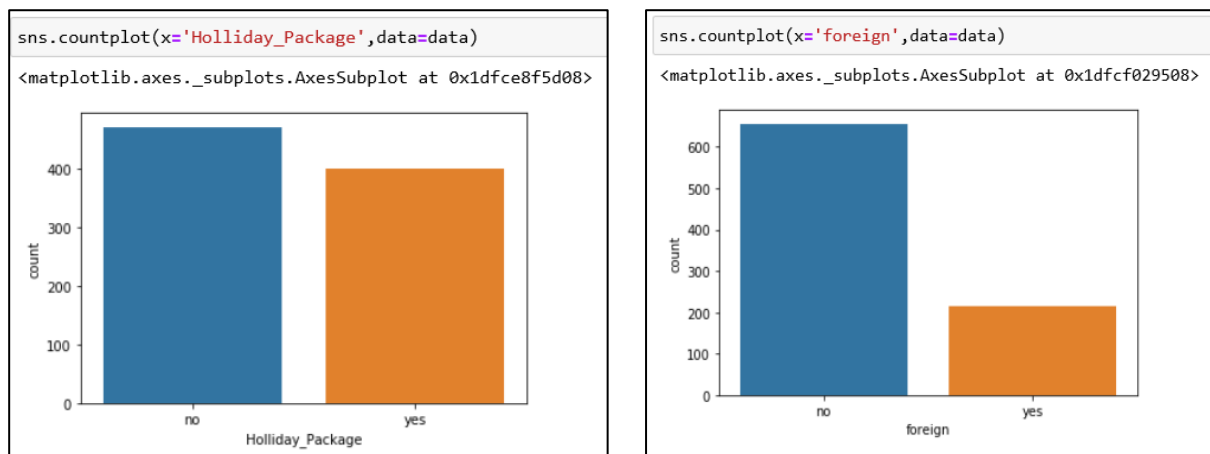
- $IQR = Q_3 - Q_1 = 12 - 8 = 4$
- Lower  $1.5 * IQR$  whisker =  $Q_1 - 1.5 * IQR = 8 - 1.5 * 4 = 2$
- Upper  $1.5 * IQR$  whisker =  $Q_3 + 1.5 * IQR = 12 + 1.5 * 4 = 18$
- Lower IQR whisker = 2, Hence all the data points below 2 are considered as outliers.
- Higher IQR whisker = 18, Hence all the data points above 18 are considered as outliers.

**Whereas, the maximum value for no. of years in education present in the dataset is 21 and min value is 1, hence clearly it demonstrates the presence of outliers in the dataset with respect to 'educ' column. And there is no outliers in the 'age' column in the dataset.**

**Similarly, other continuous variables such as 'Salary', 'no\_young\_children' and 'no\_older\_children' also have outliers, as the max value of these variable is much higher than the 75<sup>th</sup> percentile, which will make the datapoints fall below  $Q_1 - 1.5 IQR$  or above  $Q_3 + 1.5 IQR$ .**

## COUNTPLOT ( )

Count plot is used to graphically visualise the categorical data. It shows the bar chart to display the number of occurrence for each categorical data present in the dataset.



As we have seen already, there are 401 records who didn't opt for the Holiday Package and 471 records opted for the package representing 'no' and 'yes' respectively in 'Holiday\_Package' column. Similarly, there are 656 records who are foreigners and 216 employees who are not foreigners in the given dataset.

## CROSS TAB( ) – Frequency table of the factors

The `crosstab()` function is used to compute a simple cross tabulation of two (or more) factors. By default computes a frequency table of the factors unless an array of values and an aggregation function are passed.

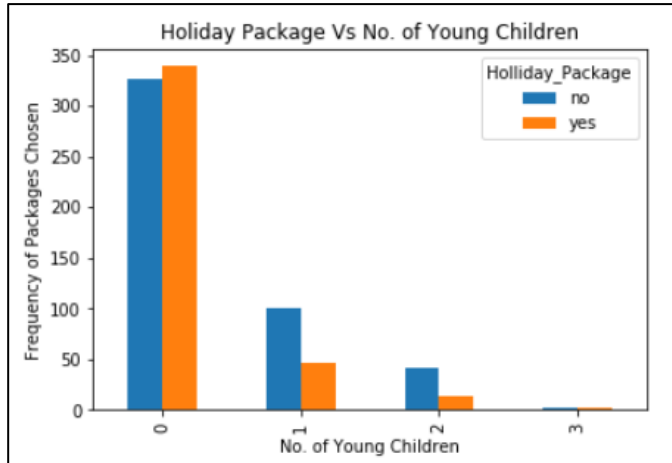
### Holiday Package Vs No. of Young Children

The below frequency table and bar chart of the cross tabulation shows that:

- Employee with no younger child are almost equally distributed between 'no' and 'yes' of the 'Holiday\_Package' and no clear inference could be gained from this data.
- Employee with 1 and 2 younger children preferred not to opt for the package.
- Employee with 3 younger are almost equally distributed among yes and no.



```
pd.crosstab(data.no_young_children,data.Holliday_Package).plot(kind='bar')
plt.title('Holiday Package Vs No. of Young Children')
plt.xlabel('No. of Young Children')
plt.ylabel('Frequency of Packages Chosen')
Text(0, 0.5, 'Frequency of Packages Chosen')
```



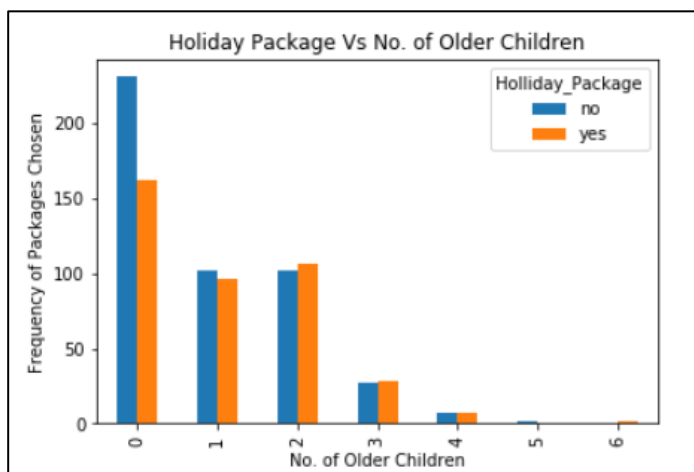
Holliday_Package	no	yes
no_young_children		
0	326	339
1	100	47
2	42	13
3	3	2

### Holiday Package Vs No. of Older Children

The below frequency table and bar chart of the cross tabulation shows that:

- Employee with no older child are preferred not to opt for the package.
- Employee with 1,2,3,and 4 older children are equally distributed among 'no' and 'yes' of 'Holliday\_Package'.

```
pd.crosstab(data.no_older_children,data.Holliday_Package).plot(kind='bar')
plt.title('Holiday Package Vs No. of Older Children')
plt.xlabel('No. of Older Children')
plt.ylabel('Frequency of Packages Chosen')
```



Holliday_Package	no	yes
no_older_children		
0	231	162
1	102	96
2	102	106
3	27	28
4	7	7
5	2	0
6	0	2

## Multivariate Analysis:

It is performed to understand interactions between different fields in the dataset (or) finding interactions between variables more than 2.

Ex:- Pair plot, scatter plot, Heatmap

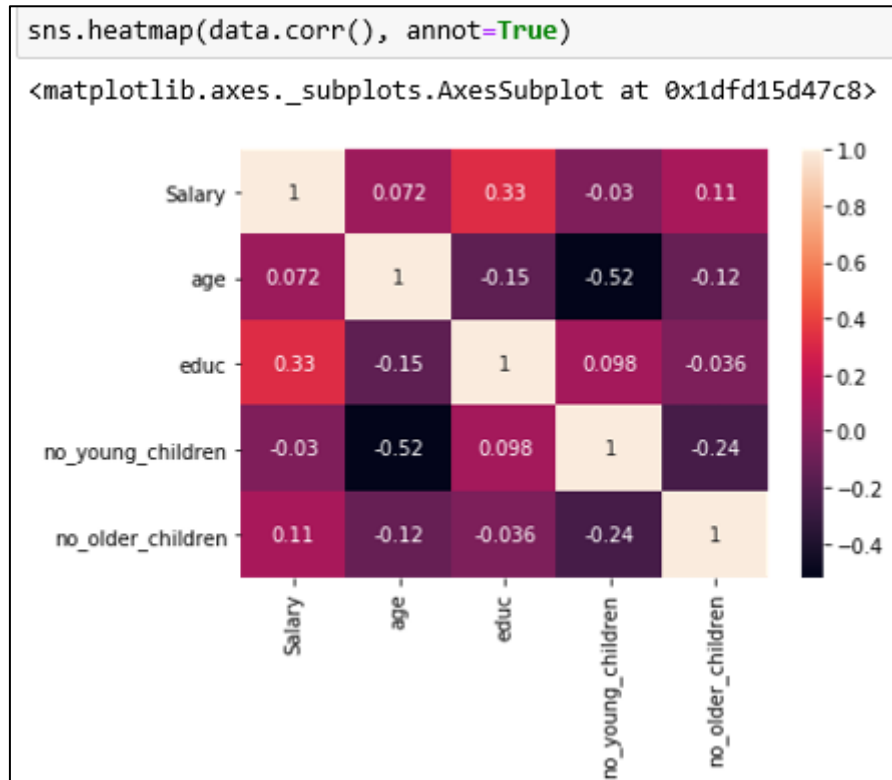
### PAIR PLOT

This shows a view of all variables and their relationship with all other variables. There is hardly any correlation between any variables in the given dataset. Especially, no\_young\_children has no correlation with any of the variables.



## HEATMAP

It is used to graphically represent the correlation between each variable graphically. From the heatmap, it could be found that the variable age and no. of younger children have **negative correlation**, indicating as age increases no. of younger children decreases. 'Salary' and 'educ' slightly have a **positive correlation** indicating as age increases Salary also increases to some extent. There is hardly any correlation between most of the variables in the dataset.



**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).**

### Encode the data – Convert all objects to categorical codes:

'Holliday\_Package' and 'foreign' are variables with object data types. Hence these 2 variables needed to be encoded with categorical codes for the unique object values present in these columns. There are 2 unique values 'yes' and 'no' in both categorical columns. Each of these values are assigned with 0 and 1 in both columns.

```
for feature in data.columns:
    if data[feature].dtype == 'object':
        print('\n')
        print('feature:', feature)
        print(pd.Categorical(data[feature].unique()))
        print(pd.Categorical(data[feature].unique()).codes)
        data[feature] = pd.Categorical(data[feature]).codes
```

```
feature: Holliday_Package
[no, yes]
Categories (2, object): [no, yes]
[0 1]
```

```
feature: foreign
[no, yes]
Categories (2, object): [no, yes]
[0 1]
```

### Check for Class imbalance:

Checking for class imbalance problem in the target variable 'Holliday\_Package'. As seen below, 54% of the data belongs to class 'no' and 46 % data belongs to class 'yes'. **Hence there is no issue of class imbalance here as we have reasonable proportions in both the classes.**

```
data['Holliday_Package'].value_counts(normalize=True)
```

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

```
data['Holliday_Package'].value_counts()
```

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

### Outlier Treatment:

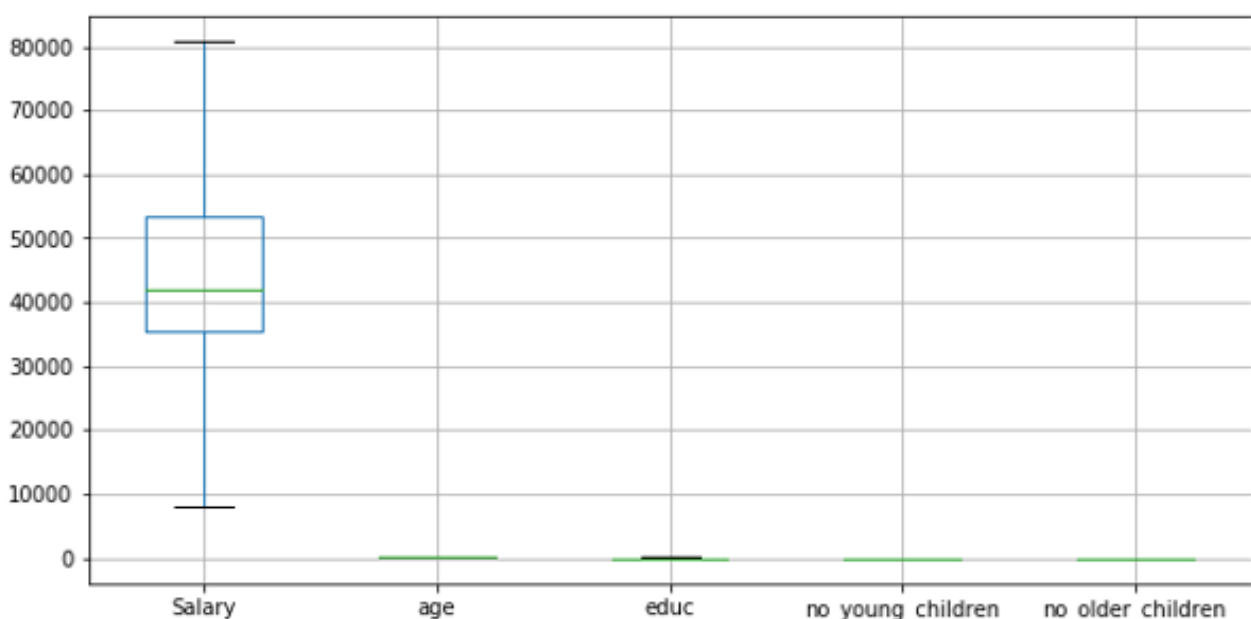
Outliers can affect both the result and assumptions. Boxplot( ) shows that there are outliers in all the variables of the dataset. Hence, before building the model, it is very important to treat the outliers. After treating outliers using IQR, there are no outliers in the data.

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

```
for column in data.columns:
    if data[column].dtype != 'object':
        lr,ur=remove_outlier(data[column])
        data[column]=np.where(data[column]>ur,ur,data[column])
        data[column]=np.where(data[column]<lr,lr,data[column])
```

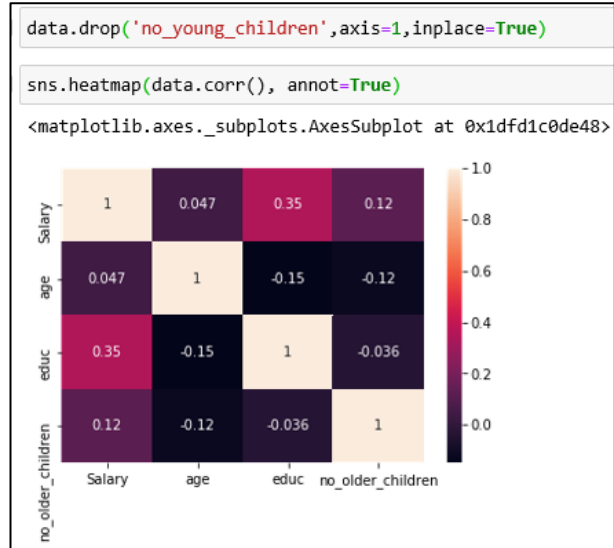
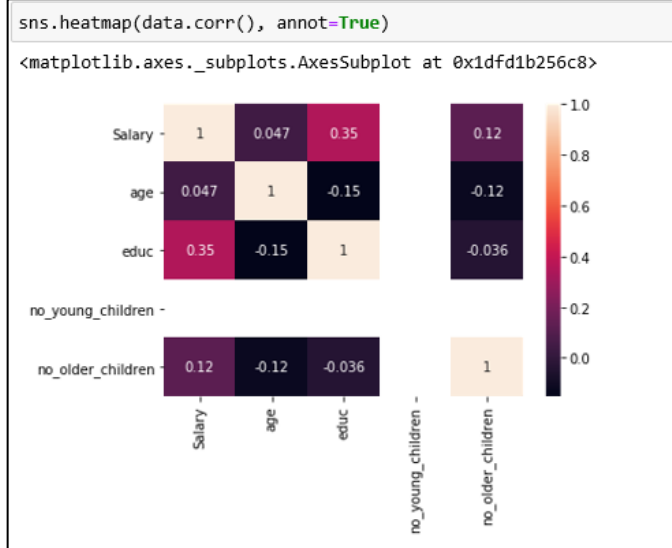
```
# construct box plot for continuous variables
plt.figure(figsize=(10,5))
data.boxplot()
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x1dfd1a714c8>

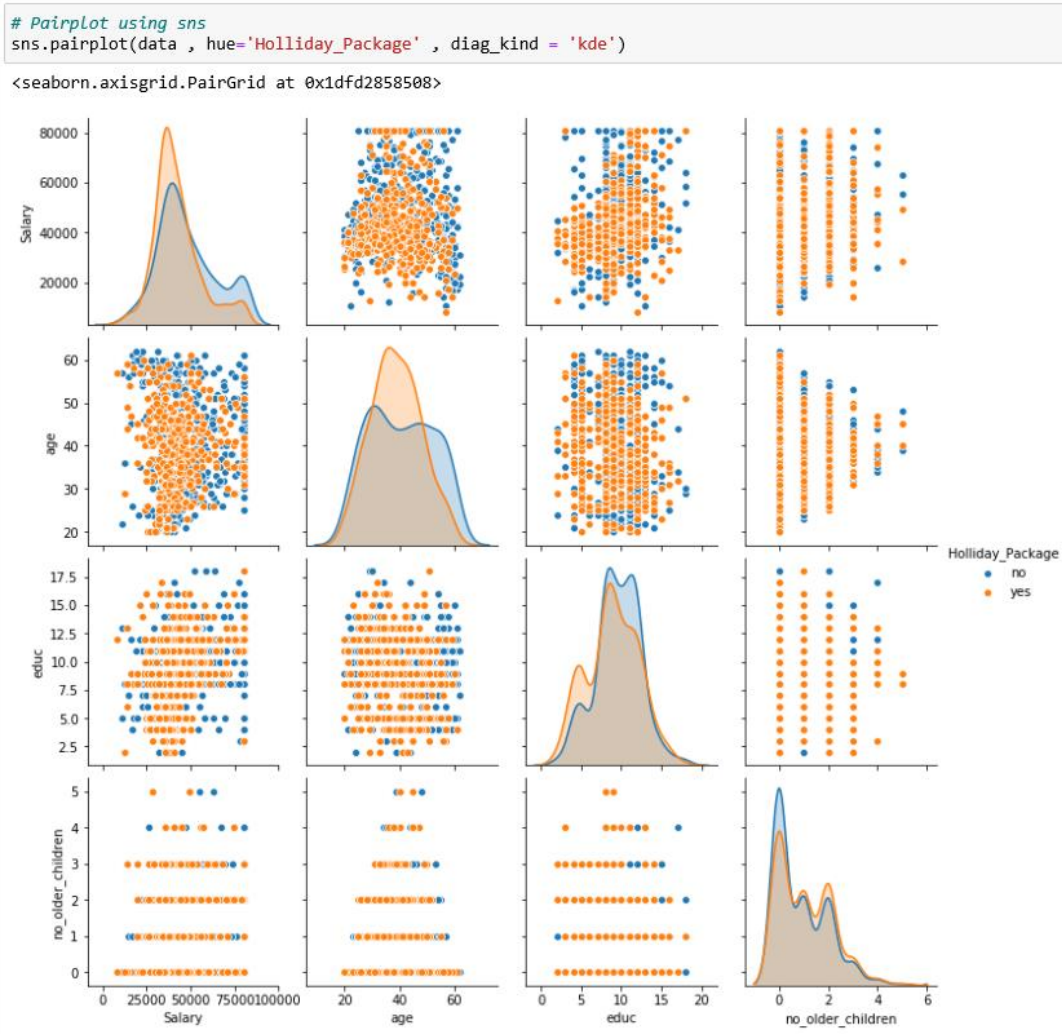


### Heatmap – Check for the correlation after treating outliers:

After treating outliers, almost all the values became 0 in 'no\_young\_children' and there is no relationship with other variables. Hence, this column could be dropped from the dataset.



### PairPlot – Check for relationship between independent variables with dependent variable.



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

```
# Copy all the predictor variables into X dataframe
X = data.drop('Holliday_Package', axis=1)

# Copy target into the y dataframe.
y = data[['Holliday_Package']]

# Split X and y into training and test set in 70:30 ratio
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

Separate the independent and dependent variables into 2 dataframes say X and y respectively. The target dependent variable 'Holliday\_Package' is in dataframe 'y' and all other independent variables are in 'X'.

Using the train\_test\_split function imported from sklearn.model\_selection, the dataframe X (independent variables) and y(dependent variable) is split into training and testing data with ratio of 70:30 respectively.

- **X\_train** : Training data of the independent variables.
- **X\_test** : Testing data of the independent variables.
- **y\_train** : Training data of the dependent variable.
- **y\_test** : Testing data of dependent variable.

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(610, 5)
(262, 5)
(610, 1)
(262, 1)
```

### Apply Logistic Regression:

- Using a LogisticRegression( ) method imported from the package sklearn.linear\_model, the Logistic Regression model is built.
- Fit the model with the training data. Pass the parameters X\_train\_scaled and y\_train for fit()
- The train and test data are predicted for their classes (no and yes) of target variable 'Holliday\_Package' using the predict() function on the model built.

```
# Fit the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
ytrain_predict = model.predict(X_train)
ytest_predict = model.predict(X_test)
```

### Apply LDA:

- Using a LinearDiscriminantMethod( ) method imported from the package sklearn.discriminant\_analysis, the LDA model is built.
- Fit the model with the training data. Pass the parameters X\_train\_scaled and y\_train for fit()

```
clf = LinearDiscriminantAnalysis()

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

```
# Predict it
pred_class = model.predict(X_test)
```

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

Model Performance measures help us to understand the goodness of the model that we have trained using the dataset. With these measures, we could get confidence in the performance of the model for future predictions.

**i) Accuracy**

- It is the ratio of number of correct predictions to the total number of input samples  
**Accuracy = No. of correct predictions / Total no. of predictions made**
- It tells us how accurately / cleanly does the model classify the data points.
- Lesser the false predictions , greater the accuracy.

**Confusion Matrix**

- Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.
- It is a 2x2 matrix reflecting the performance of the model built.

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

- **Accuracy :  $(TP + TN) / (TP + TN + FP + FN)$**
- **Sensitivity or Recall or True Positive Rate (TPR):** It describes how many of the actual true data points are identified as true data points by the model.  
**Recall =  $TP / (TP + FN)$**
- **Specificity :** How many of the actual negative data points are identified as negative data points.  
**Specificity =  $TN / (TN + FP)$**
- **Precision :** It tells that among the data points identified as positive, how many are really positive.  
**Precision =  $TP / (FP + TP)$**

**iii) ROC Curve**

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. Higher the AUC, the model predicts the target classes correctly. **The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.**

**True Positive Rate (TPR) =  $TP / (TP + FN)$**

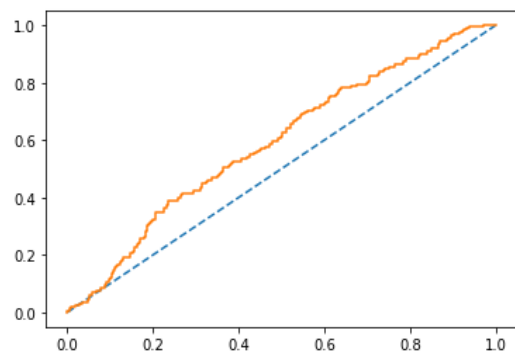
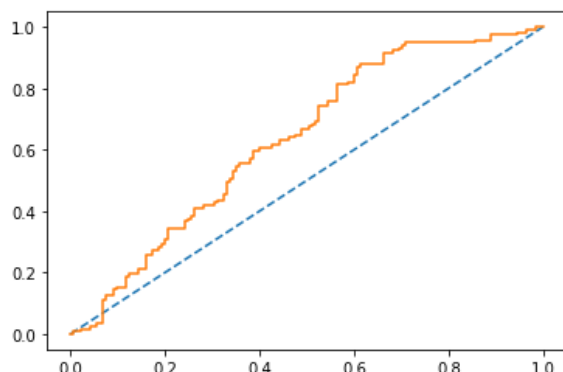
**False Positive Rate (FPR) =  $1 - \text{Specificity} = FP / (FP + TN)$**

TPR and FPR both have values in the range [0,1] and are computed at different thresholds such as (0.00, 0.02, 0.04 ,...,1.00) and a graph is drawn. AUC is the Area Under the Curve of plot FPR vs TPR at different points between [0,1].

```
# predict probabilities
probs = model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr)
```

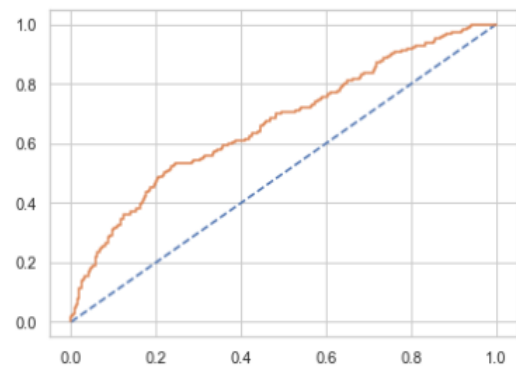
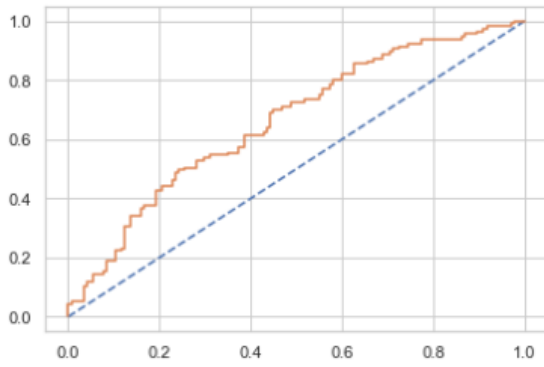


## PERFORMANCE METRICS : LOGISTIC REGRESSION MODEL

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre># Accuracy - Training Data model.score(X_train, y_train)</pre> <p>0.5344262295081967</p>	<pre># Accuracy - Test Data model.score(X_test, y_test)</pre> <p>0.5534351145038168</p>																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(y_train, ytrain_predict)</pre> <p>array([[326,  0],        [284,  0]], dtype=int64)</p>	<pre>confusion_matrix(y_test, ytest_predict)</pre> <p>array([[145,  0],        [117,  0]], dtype=int64)</p>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(y_train, ytrain_predict))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.53</td><td>1.00</td><td>0.70</td><td>326</td></tr><tr><td>1</td><td>0.00</td><td>0.00</td><td>0.00</td><td>284</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.53</td><td>610</td></tr><tr><td>macro avg</td><td>0.27</td><td>0.50</td><td>0.35</td><td>610</td></tr><tr><td>weighted avg</td><td>0.29</td><td>0.53</td><td>0.37</td><td>610</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.53	1.00	0.70	326	1	0.00	0.00	0.00	284	accuracy			0.53	610	macro avg	0.27	0.50	0.35	610	weighted avg	0.29	0.53	0.37	610	<pre>print(classification_report(y_test, ytest_predict))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.55</td><td>1.00</td><td>0.71</td><td>145</td></tr><tr><td>1</td><td>0.00</td><td>0.00</td><td>0.00</td><td>117</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.55</td><td>262</td></tr><tr><td>macro avg</td><td>0.28</td><td>0.50</td><td>0.36</td><td>262</td></tr><tr><td>weighted avg</td><td>0.31</td><td>0.55</td><td>0.39</td><td>262</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.55	1.00	0.71	145	1	0.00	0.00	0.00	117	accuracy			0.55	262	macro avg	0.28	0.50	0.36	262	weighted avg	0.31	0.55	0.39	262
	precision	recall	f1-score	support																																																									
0	0.53	1.00	0.70	326																																																									
1	0.00	0.00	0.00	284																																																									
accuracy			0.53	610																																																									
macro avg	0.27	0.50	0.35	610																																																									
weighted avg	0.29	0.53	0.37	610																																																									
	precision	recall	f1-score	support																																																									
0	0.55	1.00	0.71	145																																																									
1	0.00	0.00	0.00	117																																																									
accuracy			0.55	262																																																									
macro avg	0.28	0.50	0.36	262																																																									
weighted avg	0.31	0.55	0.39	262																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 0.591</p> <p>[&lt;matplotlib.lines.Line2D at 0x1dfd42c82c8&gt;]</p> 	<p>AUC: 0.591</p> <p>[&lt;matplotlib.lines.Line2D at 0x1dfd4645dc8&gt;]</p> 																																																												



## PERFORMANCE METRICS : LDA MODEL

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre># Accuracy - Training Data model.score(X_train, y_train)  0.6426229508196721</pre>	<pre># Accuracy - Test Data model.score(X_test, y_test)  0.6297709923664122</pre>																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(y_train, ytrain_predict)  array([[269,  57],        [161, 123]], dtype=int64)</pre>	<pre>confusion_matrix(y_test, ytest_predict)  array([[113,  32],        [ 65,  52]], dtype=int64)</pre>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(y_train, ytrain_predict))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.63</td><td>0.83</td><td>0.71</td><td>326</td></tr><tr><td>1</td><td>0.68</td><td>0.43</td><td>0.53</td><td>284</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.64</td><td>610</td></tr><tr><td>macro avg</td><td>0.65</td><td>0.63</td><td>0.62</td><td>610</td></tr><tr><td>weighted avg</td><td>0.65</td><td>0.64</td><td>0.63</td><td>610</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.63	0.83	0.71	326	1	0.68	0.43	0.53	284	accuracy			0.64	610	macro avg	0.65	0.63	0.62	610	weighted avg	0.65	0.64	0.63	610	<pre>print(classification_report(y_test, ytest_predict))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.63</td><td>0.78</td><td>0.70</td><td>145</td></tr><tr><td>1</td><td>0.62</td><td>0.44</td><td>0.52</td><td>117</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.63</td><td>262</td></tr><tr><td>macro avg</td><td>0.63</td><td>0.61</td><td>0.61</td><td>262</td></tr><tr><td>weighted avg</td><td>0.63</td><td>0.63</td><td>0.62</td><td>262</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.63	0.78	0.70	145	1	0.62	0.44	0.52	117	accuracy			0.63	262	macro avg	0.63	0.61	0.61	262	weighted avg	0.63	0.63	0.62	262
	precision	recall	f1-score	support																																																									
0	0.63	0.83	0.71	326																																																									
1	0.68	0.43	0.53	284																																																									
accuracy			0.64	610																																																									
macro avg	0.65	0.63	0.62	610																																																									
weighted avg	0.65	0.64	0.63	610																																																									
	precision	recall	f1-score	support																																																									
0	0.63	0.78	0.70	145																																																									
1	0.62	0.44	0.52	117																																																									
accuracy			0.63	262																																																									
macro avg	0.63	0.61	0.61	262																																																									
weighted avg	0.63	0.63	0.62	262																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 0.667</p> <p>[&lt;matplotlib.lines.Line2D at 0x1dfd484d708&gt;]</p> 	<p>AUC: 0.667</p> <p>[&lt;matplotlib.lines.Line2D at 0x1dfd4672408&gt;]</p> 																																																												

## Comparison – Logistic Regression Vs LDA Model

### Comparison of Logistic Regression & LDA Model – AUC Score:

- Both the train ROC\_AUC score and test ROC\_AUC score for Logistic Regression model is **0.59**. As the training data and testing data behaves similarly, we conclude that the model built is not over fitted.
- Similarly for LDA model, both the train and test ROC\_AUC score is **0.66**. In this model too, the train and test set behaves similarly and not over fitted or under fitted.

In terms of AUC Score, the LDA model built is the optimized model with the train and test AUC score of 0.66

### Comparison of Logistic Regression & LDA Model – Accuracy, Precision, Recall, F1 Score:

- In Logistic Regression model, the class 'no' (**encoded with 0**) data points are identified 100 % accurately in both training and testing dataset whereas the class 'yes' (**encoded with 1**) values are not identified correctly for any of the data points.
- In LDA model, the prediction the class 'no' values is more accurately done compared with class 'yes' values in both train and test data.

#### Accuracy:

Accuracy is a ratio of correctly predicted observation to the total observations. In this case, the test data of Logistic Regression has accuracy of 55 % whereas the prediction of test data in LDA is said to have 63% accuracy.

#### Precision:

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. This tells us among the predicted insurance claimed, how many actually claimed ?

In Logistic Regression, the precision is 0.00 for the class 'yes' values as none of the 'yes' data points are predicted correctly in both train and test data, whereas in LDA, the precision of class 'yes' in the train and test data is about 68% and 62% respectively. Thus LDA performs well compared to Logistic Regression with respect to Precision.

#### Recall:

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. This shows out of the insurance claimed, how many did we label correctly? With respect to Recall, LDA performs well compared to Logistic Regression in this model, as the class 'yes' are predicted with a score of 44% in LDA. But it is 0% in case of Logistic Regression, as none of the class 'yes' values are predicted correctly.

#### F1 score:

F1 Score is the weighted average of Precision and Recall. It takes both false positives and false negatives into account. F1 score is usually more useful than accuracy, especially if you have an uneven class distribution. But in this case, there is no class imbalance problem as we have 54:46 ratio of class 'no' and 'yes' values in the target 'Holliday\_Package' variable and LDA performs well compared to Logistic Regression with a score of 52%

## Inference – Logistic Regression Vs LDA

As seen above, model built using Logistic Regression is **“under-fitted”** in terms of very low predictions done for the class ‘yes’ values in both train and test data. But LDA model has reasonable score in every aspects of Performance metrics such as Accuracy, AUC, Precision, Recall and F1 score compared with Logistic Regression for the given dataset.

### Reason for low performance of Logistic Regression:

#### ➤ Unstable With Well Separated Classes:

Logistic regression can become unstable when the classes are well separated. We have already seen the class ‘no’ and ‘yes’ in the ratio of 54:46, which very well separated in the target ‘Holliday\_Package’ variable. This makes the model built very unstable and under-fitted.

#### ➤ Unstable With Few Examples:

Logistic regression can become unstable when there are few examples from which to estimate the parameters.

In the give dataset, there is only 872 records and hence the model built using Logistic Regression is unstable and under-fitted.

Hence, the model built using Logistic Regression has **‘High Bias’** (referring to the scenario of model being ‘Under-fitted’) where the model built does not give any accurate picture of relationship between inputs and the predicted output value, hence the **high error**.

**LDA address each of the above issues and hence LDA model is optimized model compared to Logistic Regression in terms of all the performance metrics such as AUC, Accuracy, Precision, Recall and F1 Score.**

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

Based on overall performance metrics, it is evident that LDA is comparatively optimized than the Logistics Regression for making predictions about whether the employee has opted for the holiday package or not. The co\_efficients can give us insight into a problem by telling us what variables are the most discerning between classes.

The discriminant coefficient is estimated by maximizing the ratio of the variation between the classes and the variation within the classes of the target variable. In other words, points belonging to the same class should be close together, while also being far away from the other clusters.

```
# Let us explore the coefficients for each of the independent attributes
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, model.coef_[0][idx]))
```

```
The coefficient for Salary is -1.8319623552908975e-05
The coefficient for age is -0.009003817204418329
The coefficient for educ is 0.06513802792962087
The coefficient for no_older_children is 0.1874710395626886
The coefficient for foreign is 1.3765145540963202
```

Based on the coefficients of the above variables, **in terms of magnitude**, ‘Salary’ and ‘foreign’ has the more impact on choosing the holiday package compared to other variables. ‘Salary’ has the negative sign, indicating, it has a negative impact on the opting for the holiday package. **Hence as the Salary increases, the probability of opting for holiday packages decreases and if they are foreigners, there is a high probability of opting for the package.**

Thus the predictions is,

- **If the employees are foreigners there is more possibility of choosing the holiday package.** Hence foreigners can be given with existing offers on the holiday packages, whereas non-foreigners could be given additional exciting offers on stay and places to be visited, so we could try persuade them to choose the holiday packages.
- **Low Salaried employees in the company prefer to choose the holiday package than the high salaried employees.**