PREDICTIVE MODELLING PROJECT

1. LINEAR REGRESSION:

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.

First step is to load all the necessary packages and importing the data.

- **data = pd.read_csv('cubic_zirconia.csv')**
- **data.head(5)**

	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

- **data.tail(5)**

	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

- **data.shape**

```
(26967, 11)
```

- **data.describe(include="all").T – we have both categorical and continuous data.**

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

Categorical- Cut, color,clarity

Continuous - carat, depth, x, y, z, price which is the target variable

- **data.info() – we have float, object and int data types**

```
<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
```

- **data.isnull().sum() –** depth has 697 null values

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

- **duplicate=data.duplicated()**
- **duplicate.sum() –** There is no duplicate values found

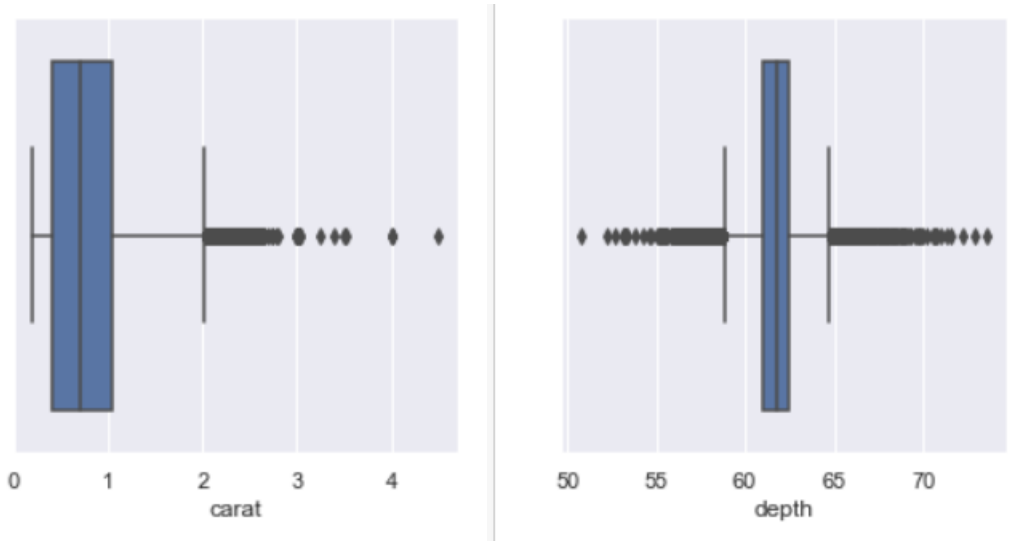
```
0
```

- **data[['carat' ,'depth', 'table', 'x', 'y', 'z','price']].nunique() –** shows unique number of values in each variable.

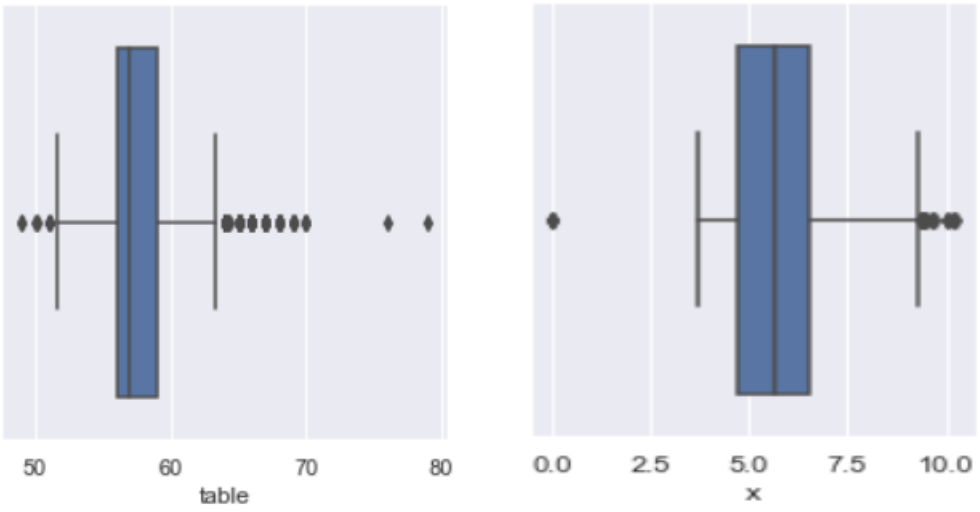
```
carat     257
depth     169
table     112
x         531
y         526
z         356
price    8742
dtype: int64
```

- **sns.set(rc={'figure.figsize':(4,4)})**
- **sns.boxplot(data['carat'])**
- **sns.set(rc={'figure.figsize':(4,4)})**
- **sns.boxplot(data['depth'])**
- **sns.set(rc={'figure.figsize':(4,4)})**
- **sns.boxplot(data['table'])**

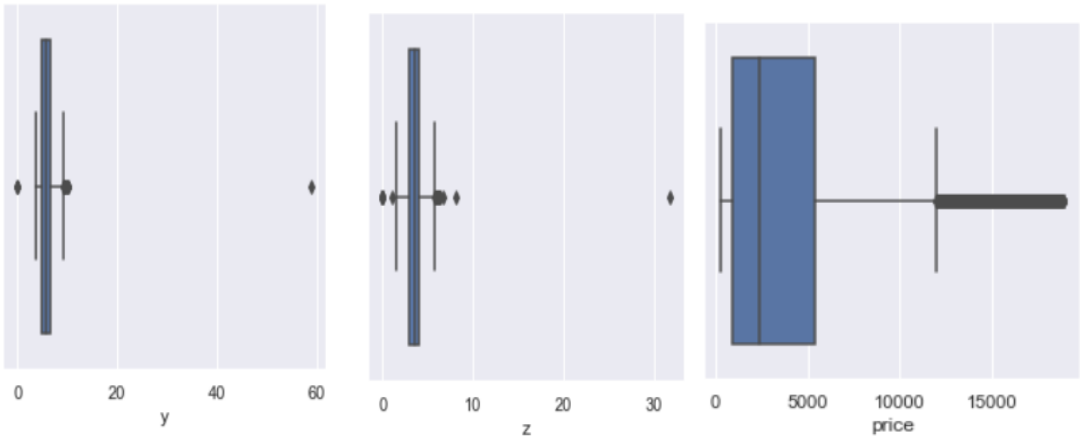
- **sns.set(rc={'figure.figsize':(3,4)})**
- **sns.boxplot(data['x'])**
- **sns.set(rc={'figure.figsize':(3,4)})**
- **sns.boxplot(data['y'])**
- **sns.set(rc={'figure.figsize':(3,4)})**
- **sns.boxplot(data['z'])**
- **sns.set(rc={'figure.figsize':(4,4)})**
- **sns.boxplot(data['price'])**



The boxplot of carat variable seems to have large number of outliers and the majority of data lies between 0-1. The boxplot of depth holds many outliers and ranges from 55-65.



The boxplot of table has outliers and distributed between 55-65. The boxplot of x shows that data consist of outliers and lies between 4-8

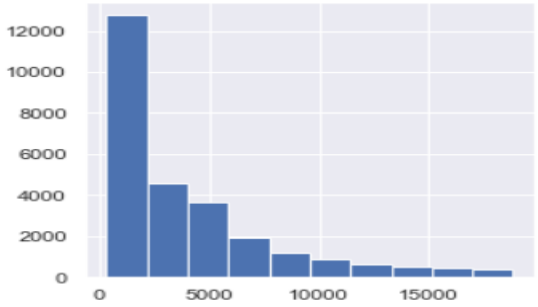


The boxplot of y, z and price consist of outliers and price values lies between 100-8000

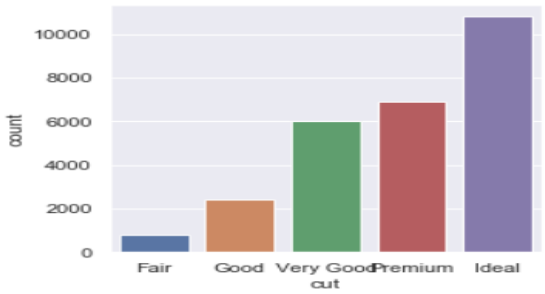
- **data.skew()**

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

- **plt.hist(data.price, bins = 10)**
- **plt.show()**

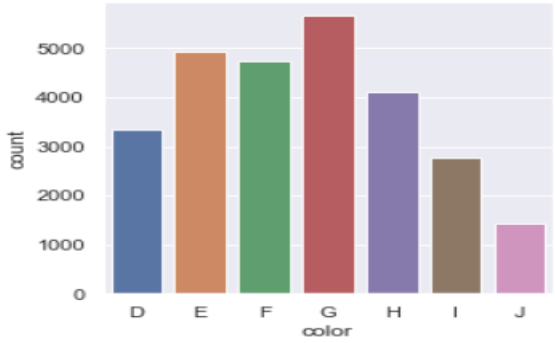


- **sns.countplot(data['cut'], order = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal'])**



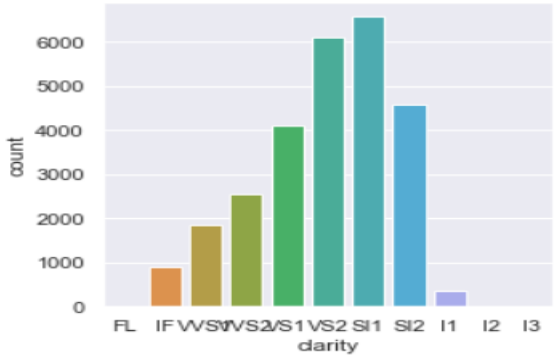
The mostly preferred cut is **Ideal** and the quality is present in increasing order.

- **sns.countplot(data['color'] ,order = ['D', 'E', 'F', 'G','H', 'I', 'J'])**



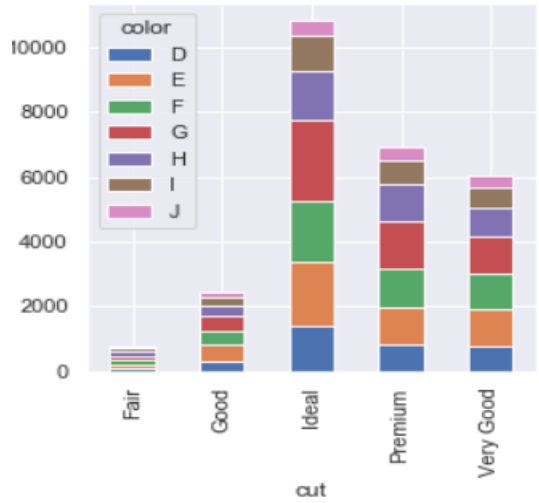
The most preferred and used color is G and it is present in the middle of all colors.

- **sns.countplot(data['clarity'], order = ['FL', 'IF', 'VVS1', 'VVS2', 'VS1', 'VS2', 'SI1', 'SI2', 'I1', 'I2', 'I3'])**



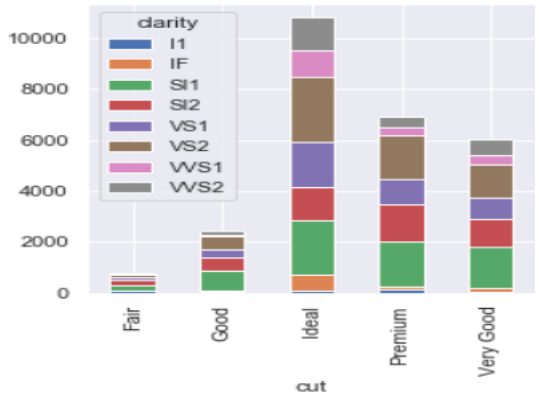
The most preferred clarity is VS2.

- **pd.crosstab(data['cut'], data['color']).plot(kind = 'bar', stacked = True)**



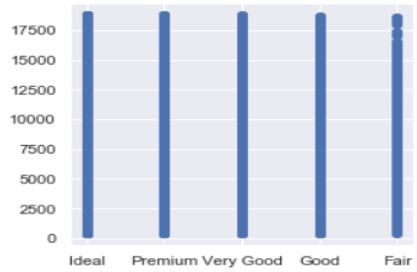
The barplot is plotted based on the cut and color. Ideal and G are the most used.

- pd.crosstab(data['cut'], data['clarity']).plot(kind = 'bar', stacked = True)

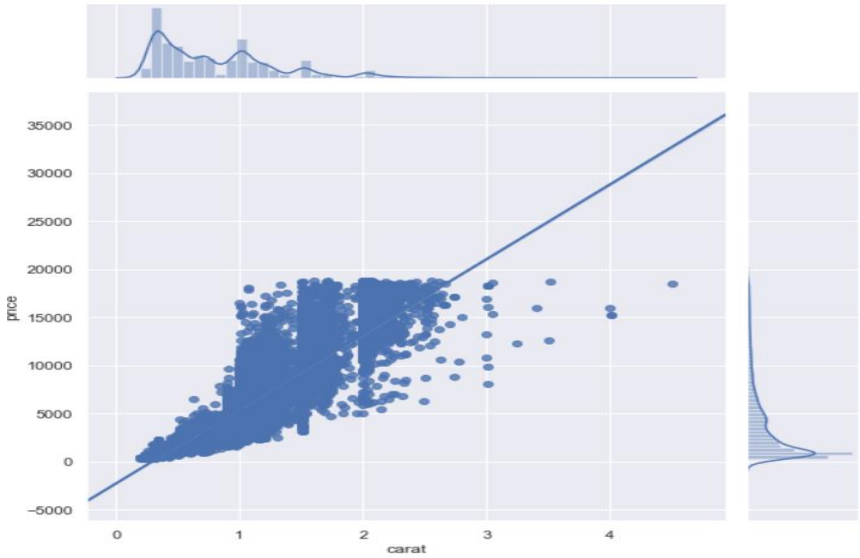


The barplot is plotted based on cut and clarity. Ideal and VS2 is most preferred.

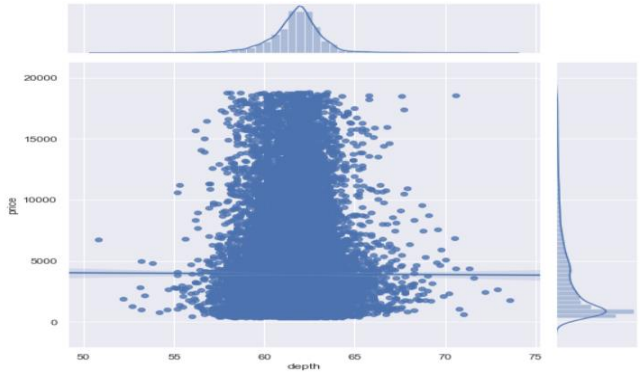
- plt.scatter(data['cut'], data['price'])
- plt.show()



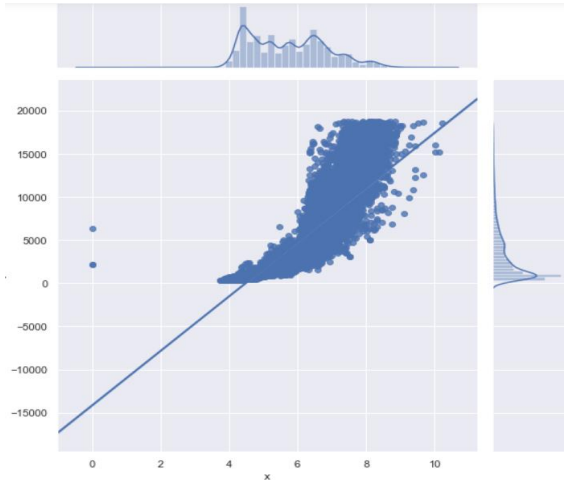
- a = sns.jointplot('carat', 'price', data = data, kind='reg', height=8)
- plt.show()



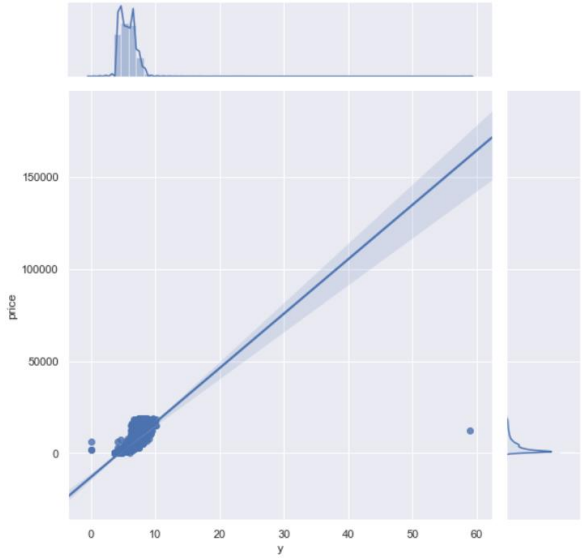
- b = sns.jointplot('depth', 'price', data = data, kind='reg', height=8)
- plt.show()



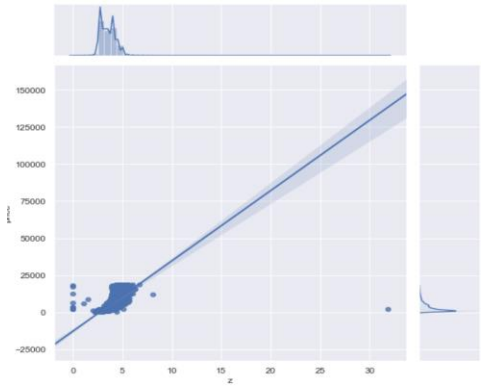
- c = sns.jointplot('x', 'price', data = data, kind='reg', height=8)
- plt.show()



- d = sns.jointplot('y', 'price', data = data, kind='reg', height=8)
- plt.show()

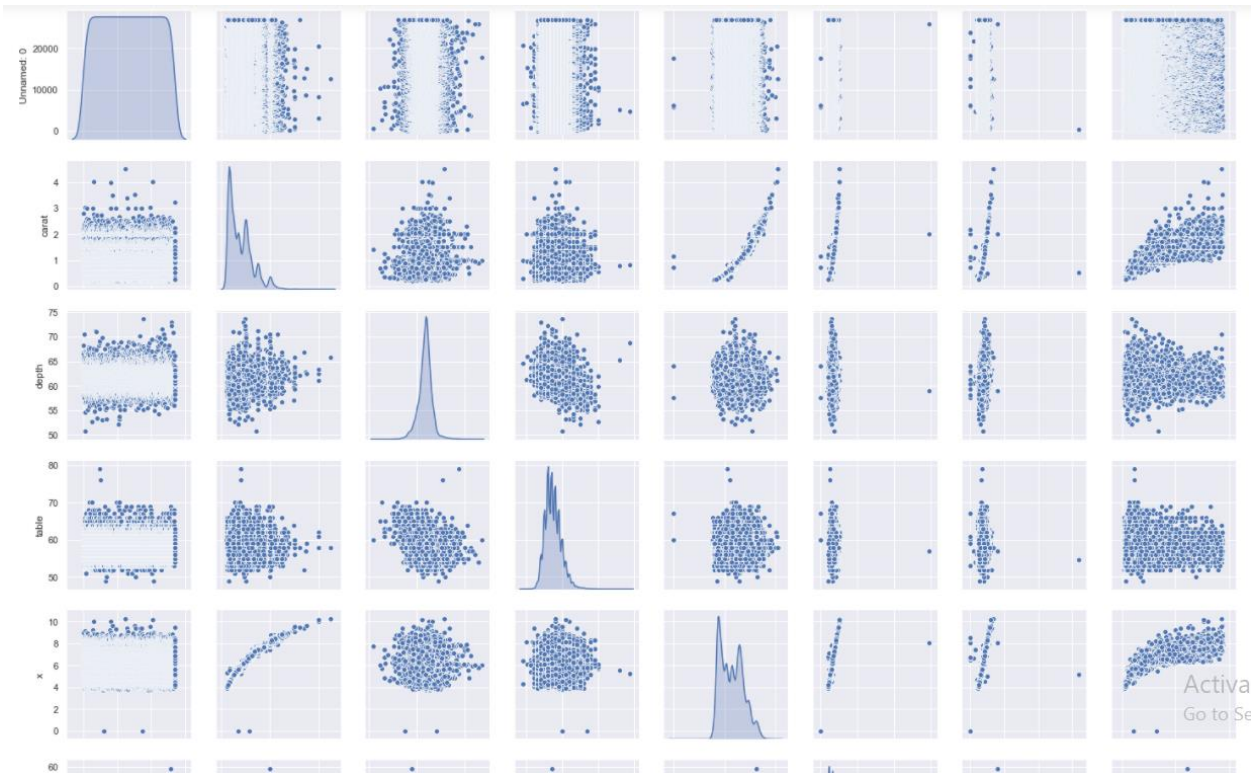


- e = sns.jointplot('z', 'price', data = data, kind='reg', height=8)
- plt.show()



Correlation between variables is plotted.

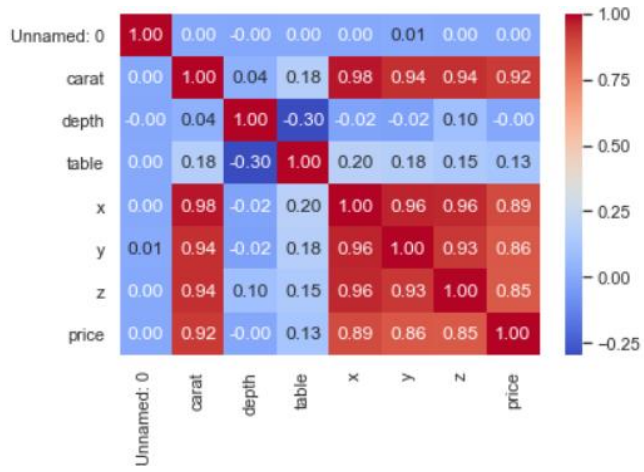
- sns.pairplot(data, diag_kind='kde')
- plt.show()



The distribution of the data is shown using Pairplot.

- cor = data.corr()
- plt.figure(figsize=(6,4))
- sns.heatmap(cor, annot=True, fmt = '.2f', cmap='coolwarm')

This matrix shows that there are multi collinearity in the data.



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?

- data.isnull().sum() – we have null values in the depth variable. Since it is a continuous variable we can impute using either mean or median.

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

- data.loc[((data['x'] == 0) | (data['y'] == 0) | (data['z'] == 0))]

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
5821	5822	0.71	Good	F	SI2	64.1	60.0	0.00	0.00	0.0	2130
6034	6035	2.02	Premium	H	VS2	62.7	53.0	8.02	7.95	0.0	18207
6215	6216	0.71	Good	F	SI2	64.1	60.0	0.00	0.00	0.0	2130
10827	10828	2.20	Premium	H	SI1	61.2	59.0	8.42	8.37	0.0	17265
12498	12499	2.18	Premium	H	SI2	59.4	61.0	8.49	8.45	0.0	12631
12689	12690	1.10	Premium	G	SI2	63.0	59.0	6.50	6.47	0.0	3696
17506	17507	1.14	Fair	G	VS1	57.5	67.0	0.00	0.00	0.0	6381
18194	18195	1.01	Premium	H	I1	58.1	59.0	6.66	6.60	0.0	3167
23758	23759	1.12	Premium	G	I1	60.4	59.0	6.71	6.67	0.0	2383

- data.drop(data[((data['x'] == 0) | (data['y'] == 0) | (data['z'] == 0))].index, inplace=True)
- for column in data.columns:
- if data[column].dtype != 'object':
- median = data[column].median()
- data[column] = data[column].fillna(median)

- data.isnull().sum()

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

After imputing median values, we don't have any null values.

- from sklearn.preprocessing import StandardScaler
- sc = StandardScaler()
- data_num = data.select_dtypes(exclude=['object'])
- data[data_num.columns] = sc.fit_transform(data_num)

Scaling is performed to reduce the multi collinearity in the data. If scaling is not performed, variance inflammation factor values are high which purely indicates the presence of multi collinearity. VIF is calculated after building the linear regression model and proved that scaling has no much impact in model score or coefficient or intercept.

- data1 = pd.get_dummies(data, columns=['cut','color','clarity'],drop_first=True)
- data1.head() – since linear regression model does not take categorical values, we convert the categorical into integer values.

	Unnamed: 0	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	...	color_H	color_I	color_J	clarity_IF	clarity_SI
0	-1.731904	-1.043125	0.253399	0.244112	-1.295920	-1.240065	-1.224865	-0.854851	0	1	...	0	0	0	0	
1	-1.731776	-0.980310	-0.679158	0.244112	-1.162787	-1.094057	-1.169142	-0.734303	0	0	...	0	0	0	1	
2	-1.731647	0.213173	0.325134	1.140496	0.275049	0.331668	0.335404	0.584271	0	0	...	0	0	0	0	
3	-1.731519	-0.791865	-0.105277	-0.652273	-0.807766	-0.802041	-0.806936	-0.709945	0	1	...	0	0	0	0	
4	-1.731390	-1.022187	-0.966099	0.692304	-1.224916	-1.119823	-1.238796	-0.785257	0	1	...	0	0	0	0	

5 rows × 25 columns

```
Index(['Unnamed: 0', 'carat', 'depth', 'table', 'x', 'y', 'z', 'price',
       'cut_Good', 'cut_Ideal', 'cut_Premium', 'cut_Very Good', 'color_E',
       'color_F', 'color_G', 'color_H', 'color_I', 'color_J', 'clarity_IF',
       'clarity_SI1', 'clarity_SI2', 'clarity_VS1', 'clarity_VS2',
       'clarity_VVS1', 'clarity_VVS2'],
      dtype='object')
```

- model = data1.drop(columns=['Unnamed: 0'], axis=1) – since unnamed 0 is not used.
- model.head()

	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	cut_Premium	...	color_H	color_I	color_J	clarity_IF	clarity
0	-1.043125	0.253399	0.244112	-1.295920	-1.240065	-1.224865	-0.854851	0	1	0	...	0	0	0	0	
1	-0.980310	-0.679158	0.244112	-1.162787	-1.094057	-1.169142	-0.734303	0	0	1	...	0	0	0	1	

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

- x = model.drop('price', axis=1)
- y = model[['price']]
- x.head(5)

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...	color_H	color_I	color_J	clarity_IF	clarity_
0	-1.043125	0.253399	0.244112	-1.295920	-1.240065	-1.224865	0	1	0	0	...	0	0	0	0	
1	-0.980310	-0.679158	0.244112	-1.162787	-1.094057	-1.169142	0	0	1	0	...	0	0	0	1	
2	0.213173	0.325134	1.140496	0.275049	0.331668	0.335404	0	0	0	1	...	0	0	0	0	
3	-0.791865	-0.105277	-0.652273	-0.807766	-0.802041	-0.806936	0	1	0	0	...	0	0	0	0	
4	-1.022187	-0.966099	0.692304	-1.224916	-1.119823	-1.238796	0	1	0	0	...	0	0	0	0	

5 rows × 23 columns

- x.shape
- y.shape

```
(26958, 23) // Splitting train and test set into 70:30 ratio
```

- x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3 , random_state=1)
- regression_model = LinearRegression() // **Using linear regression function and fitting be stfit model on training data.**
- regression_model.fit(x_train, y_train)

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

- for i, col_name in enumerate(x_train.columns):
- print("The coefficient for {} is {}".format(col_name, regression_model.coef_[0][i]))

```
The coefficient for carat is 1.367270935949181
The coefficient for depth is -0.027157297781958373
The coefficient for table is -0.015129062503321838
The coefficient for x is -0.3109893370891072
The coefficient for y is -0.0008718302715271618
The coefficient for z is -0.009459310770526735
The coefficient for cut_Good is 0.13136322591216146
The coefficient for cut_Ideal is 0.19405082192916592
The coefficient for cut_Premium is 0.1695361974418688
The coefficient for cut_Very Good is 0.1637510414681501
The coefficient for color_E is -0.045829921106505064
The coefficient for color_F is -0.06423152006658801
The coefficient for color_G is -0.10934322364634387
The coefficient for color_H is -0.23735034810633143
The coefficient for color_I is -0.3612269499771003
The coefficient for color_J is -0.5838191499347688
The coefficient for clarity_IF is 1.289947139967373
The coefficient for clarity_SI1 is 0.8895287879225809
The coefficient for clarity_SI2 is 0.6446204697130651
The coefficient for clarity_VS1 is 1.1118581585704668
The coefficient for clarity_VS2 is 1.0384035090938624
The coefficient for clarity_VVS1 is 1.2151510670753523
The coefficient for clarity_VVS2 is 1.1977150915884176
```

- intercept = regression_model.intercept_[0]
- print("The intercept for our model is {}".format(intercept))

```
The intercept for our model is -0.9907889549988897
```

- regression_model.score(x_train, y_train)
  - 0.9232445774547248 //**R square value for training data**
- regression_model.score(x_test, y_test)
  - 0.9171155258688372 //**R square value for test data**
- predicted_train=regression_model.fit(x_train, y_train).predict(x_train)
- np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
  - 0.274480473337745 //**RMSE on training data**
- predicted_test=regression_model.fit(x_train, y_train).predict(x_test)
- np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
  - 0.29399280117472737 //**RMSE on test data**

- from statsmodels.stats.outliers_influence import variance_inflation_factor
- vif = [variance_inflation_factor(X.values, ix) for ix in range(X.shape[1])]
- i=0
- for column in X.columns:
- if i < 11:
- print (column ,"--->", vif[i])
- i = i+1

```
carat ---> 25.485885004032028
depth ---> 1.5799537095045701
table ---> 1.7385495006353093
x ---> 48.735020135958294
y ---> 13.932118782114511
z ---> 16.131522272857556
cut_Good ---> 3.530404090036462
cut_Ideal ---> 14.62595830897545
cut_Premium ---> 8.74120100324553
cut_Very Good ---> 7.749342872170176
color_E ---> 2.3694860075909174
```

- train = pd.concat([x_train, y_train], axis=1)
- train.rename(columns = {'cut_Very Good' : 'cut_Very_Good'}, inplace = True)
- import statsmodels.formula.api as smf
- lm1 = smf.ols(formula= 'price ~ carat + depth + table + x+ y + z + cut_Good + cut_Ideal + cut_Premium + cut_Very_Good + color_E + color_F + color_G + color_H + color_I + color_J + clarity_IF + clarity_SI1 + clarity_SI2 + clarity_VS1 + clarity_VS2 + clarity_VVS1 + clarity_VVS2', data = train).fit()
- lm1.params

```
Intercept       -0.990789
carat            1.367271
depth           -0.027157
table           -0.015129
x               -0.310989
y               -0.000872
z               -0.009459
cut_Good         0.131363
cut_Ideal        0.194051
cut_Premium      0.169536
cut_Very_Good    0.163751
color_E         -0.045830
color_F         -0.064232
color_G         -0.109343
color_H         -0.237350
color_I         -0.361227
color_J         -0.583819
clarity_IF       1.289947
clarity_SI1      0.889529
clarity_SI2      0.644620
clarity_VS1      1.111858
clarity_VS2      1.038404
clarity_VVS1     1.215151
clarity_VVS2     1.197715
dtype: float64
```

- mse = np.mean((regression_model.predict(x_test)-y_test)**2)
- import math
- math.sqrt(mse)
  - 0.2939928011747273
- regression_model.score(x_test, y_test)
  - 0.9171155258688372
- print(lm1.summary())

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                  price   R-squared:                       0
.923
Model:                            OLS   Adj. R-squared:                  0
.923
Method:                 Least Squares   F-statistic:                     9
856.
Date:                Sat, 12 Jun 2021   Prob (F-statistic):
0.00
Time:                        13:16:08   Log-Likelihood:                -23
78.8
No. Observations:               18870   AIC:                             4
806.
```

```
Df Residuals:                    18846   BIC:                                4
994.
Df Model:                           23
Covariance Type:            nonrobust
==============================================================================
=======
                   coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept        -0.9908      0.021    -46.438      0.000      -1.033
-0.949
carat             1.3673      0.010    134.955      0.000       1.347
1.387
depth            -0.0272      0.003    -10.676      0.000      -0.032
-0.022
table            -0.0151      0.003     -5.662      0.000      -0.020
-0.010
x                -0.3110      0.013    -24.021      0.000      -0.336
-0.286
y                -0.0009      0.006     -0.139      0.889      -0.013
0.011
z                -0.0095      0.007     -1.396      0.163      -0.023
0.004
cut_Good          0.1314      0.014      9.356      0.000       0.104
0.159
cut_Ideal         0.1941      0.014     13.867      0.000       0.167
0.221
cut_Premium       0.1695      0.013     12.572      0.000       0.143
0.196
cut_Very_Good     0.1638      0.013     12.151      0.000       0.137
0.190
color_E          -0.0458      0.007     -6.188      0.000      -0.060
-0.031
color_F          -0.0642      0.008     -8.554      0.000      -0.079
-0.050
color_G          -0.1093      0.007    -14.942      0.000      -0.124
-0.095
color_H          -0.2374      0.008    -30.447      0.000      -0.253
-0.222
color_I          -0.3612      0.009    -41.645      0.000      -0.378
-0.344
color_J          -0.5838      0.011    -54.585      0.000      -0.605
-0.563
clarity_IF        1.2899      0.021     60.927      0.000       1.248
1.331
clarity_SI1       0.8895      0.018     48.947      0.000       0.854
0.925
clarity_SI2       0.6446      0.018     35.268      0.000       0.609
0.680
clarity_VS1       1.1119      0.019     59.990      0.000       1.076
1.148
clarity_VS2       1.0384      0.018     56.855      0.000       1.003
1.074
```

```
clarity_VVS1      1.2152      0.020     61.963      0.000       1.177
1.254
clarity_VVS2      1.1977      0.019     62.829      0.000       1.160
1.235
==============================================================================
====
Omnibus:                     5110.944   Durbin-Watson:                   1
.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           187422
.503
Skew:                           0.614   Prob(JB):
0.00
Kurtosis:                      18.390   Cond. No.
50.0
==============================================================================
====
```

We can drop the variables which are highly correlated. Dropping variables bring down the multi collinearity down.

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

From the above analysis, we can say that ideal cut had more profits to the company. The colors H,I,J have brought profits to the company. In clarity, there were no profits from I1, I2, I3 and no flawless stones. The ideal, premium was bringing profits whereas fair and good are not bringing profits. Using stats model, if we run more iterations, the P value and coefficient can bring us the better results.

- The ideal, premium, very good cut types are the one which are bringing profits so that we could use marketing for those cuts to make more profit in the business.
- The clarity of the diamond is also an important factor. If the diamonds are more clear, then the profit will be more.

2. LOGISTIC REGRESSION AND LDA

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.

The first step is to load the packages and import the data.

- 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

- data.tail()

	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

- data.shape

```
(872, 8)
```

- data.info() – no null values in the dataset and the data types are integer and object.

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

- data.describe(include ='all').T

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

Holliday_Package – Target variable

Salary, Age, Educ, no younger children, no older children of an employee who went to foreign helps to predict whether the family has undergone holiday package or not.

- data.isnull().sum()

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

- dup = data.duplicated()
- print('Number of duplicate rows = %d' % (dup.sum()))

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

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

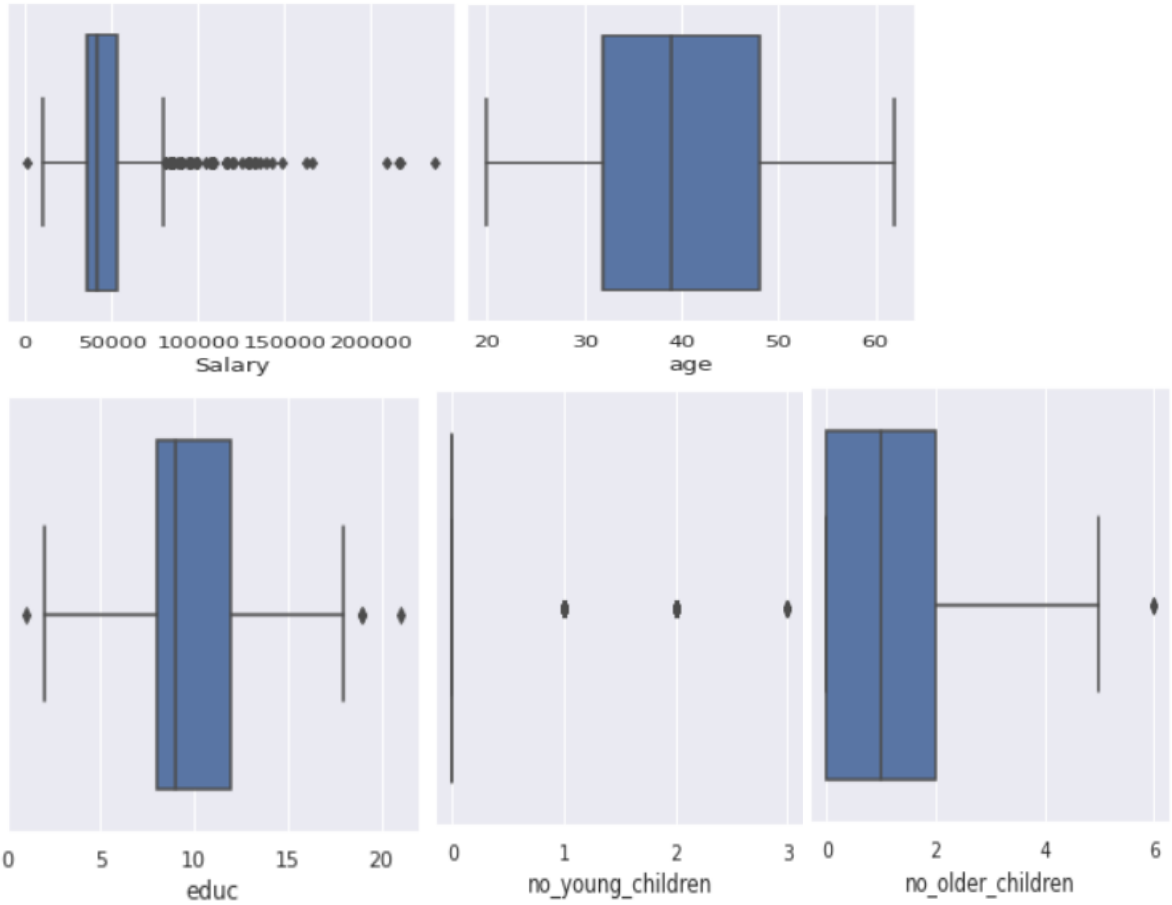
```
HOLLIDAY_PACKAGE :  2
yes     401
no      471
Name: Holliday_Package, dtype: int64


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

- data.Holliday_package.value_counts(1)

```
no      0.540138
yes     0.459862
```

  It shows that 45% of family are opting for the holiday package.

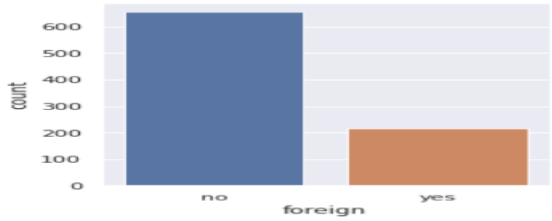
- cols = ['Salary' ,'age', 'educ', 'no_young_children', 'no_older_children']
- for i in cols:
- sns.boxplot(data[i])
- plt.show()



- data.skew()

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

- sns.countplot(x="foreign", data=data)

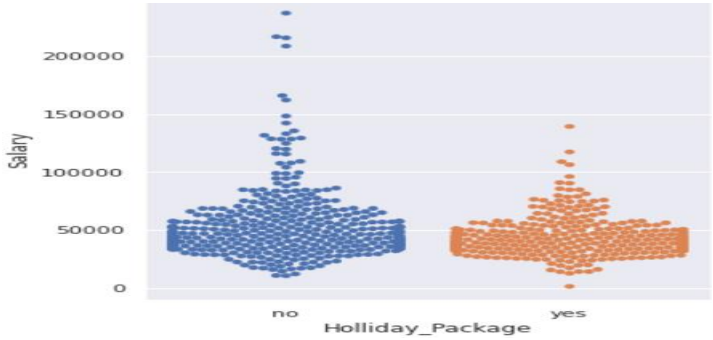


The countplot shows the total count of yes and no in the foreign variable

- sns.countplot(x="Holliday_Package", data=data)

The count plot shos that count of total families that has opted for holiday package and not opted.

- sns.catplot(x="Holliday_Package", y="Salary",kind="swarm",data=data)



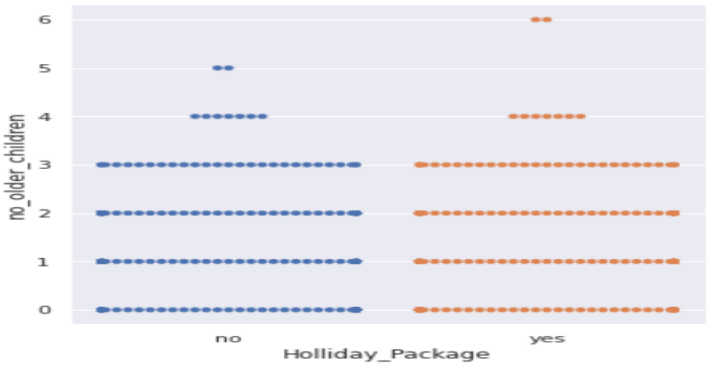
It is shown that employee who is getting lesser than 150000 salary are choosing the holiday package

- sns.catplot(x="Holliday_Package", y="age",kind="swarm",data=data)

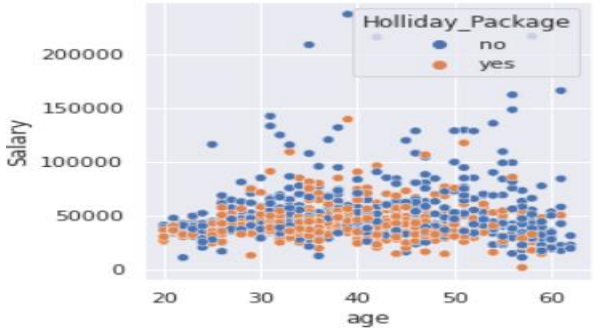
- sns.catplot(x="Holliday_Package", y="educ",kind="swarm",data=data)

- sns.catplot(x="Holliday_Package", y="no_young_children",kind="swarm",data=data)

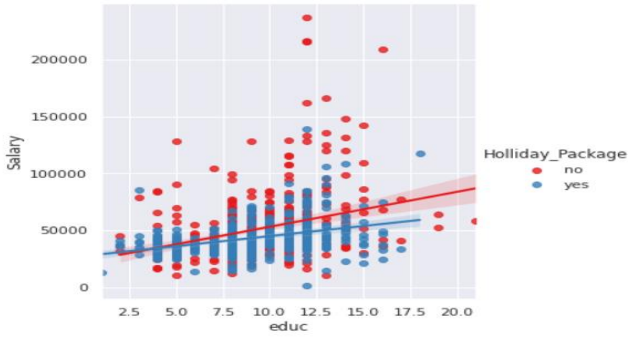
- sns.catplot(x="Holliday_Package", y="no_older_children",kind="swarm",data=data)



- sns.scatterplot(data = data, x='age',y='Salary', hue = 'Holliday_Package')

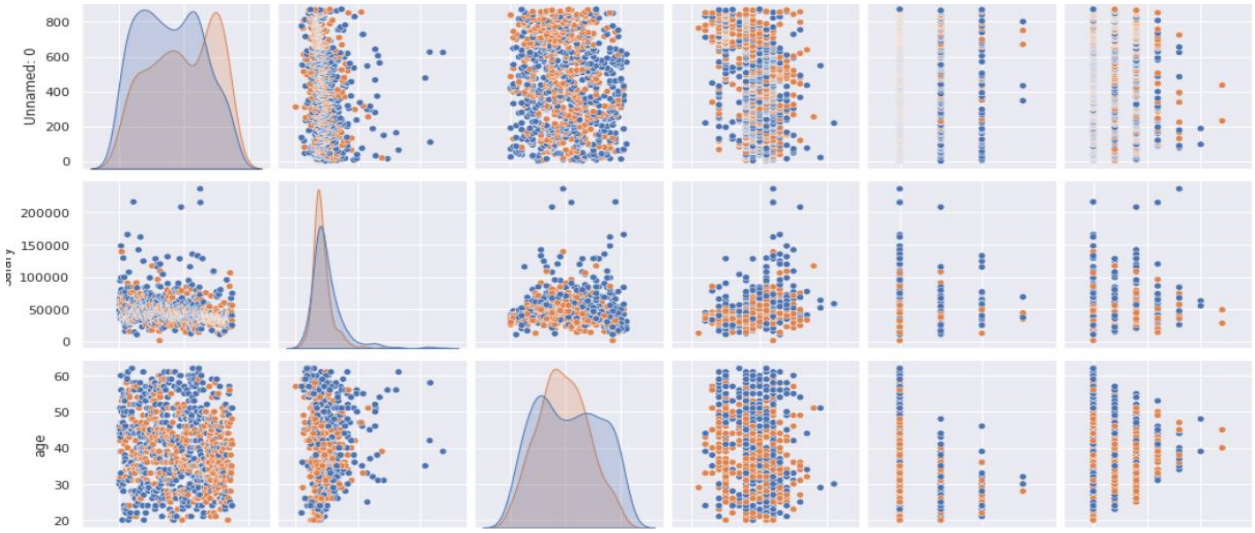


- sns.lmplot(x="educ", y="Salary", hue="Holliday_Package", data=data, palette ="Set1")



- sns.scatterplot(data = data, x='no_older_children',y='age', hue = 'Holliday_Package')

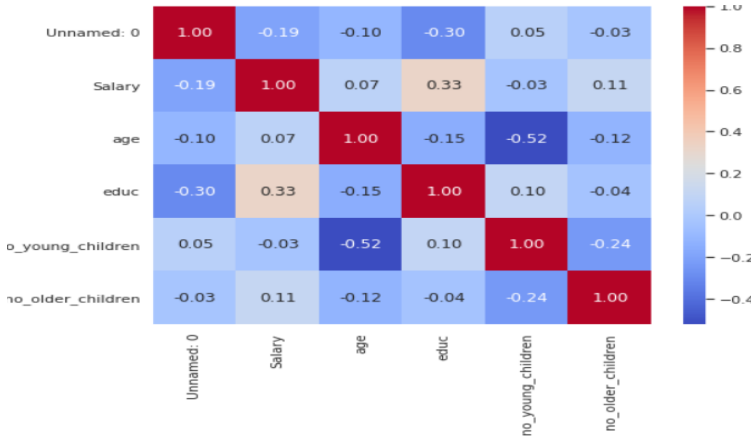
- sns.pairplot(data ,diag_kind='kde' ,hue='Holliday_Package')



The distribution shows that there is no correlation between the data and follows normal distribution.

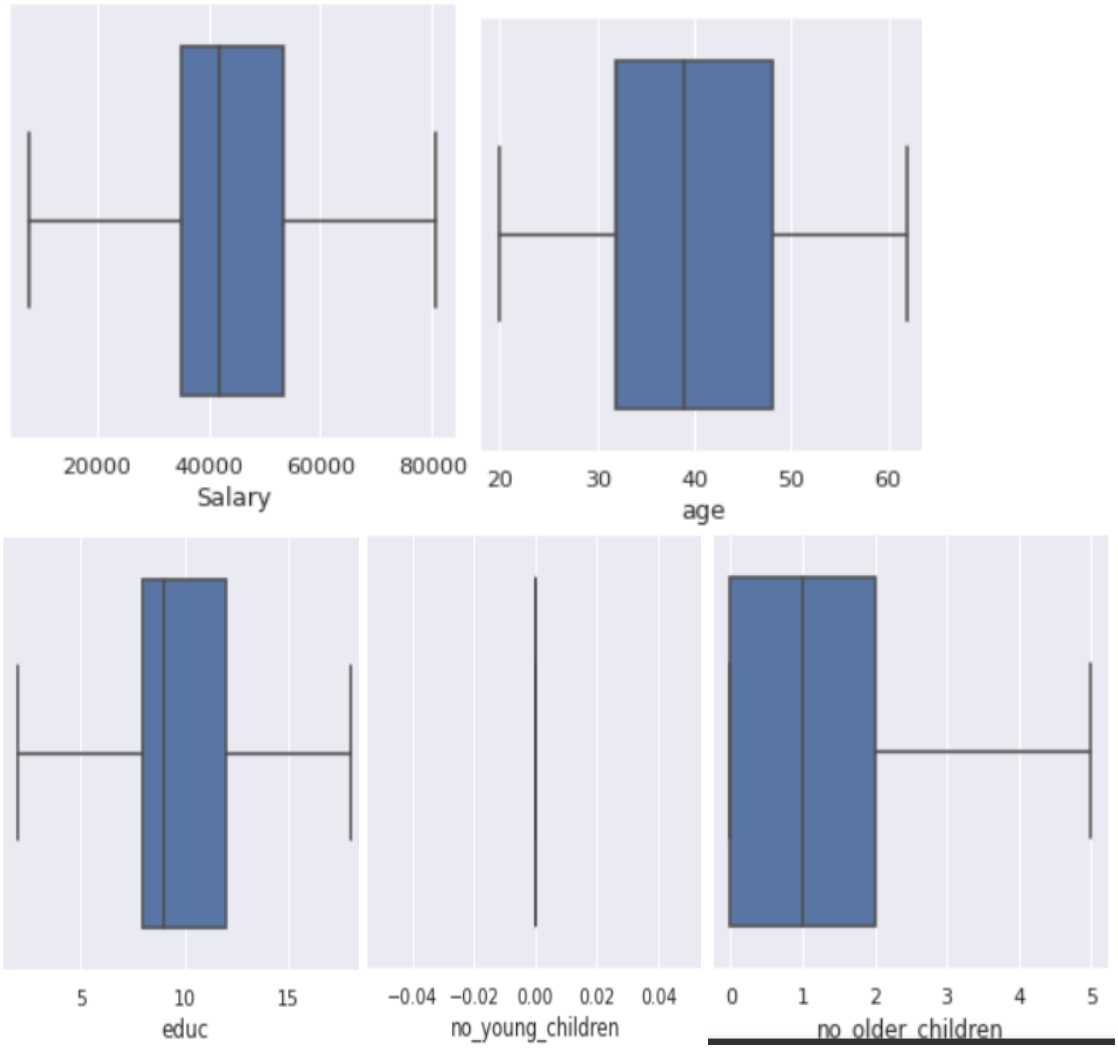
From the age and salary plot, it is shown that the age between 50-60 seems to be not taking the holiday package and age between 30-50 and salary less than 50000 people have opted more for holiday package.

- cor = data.corr()
- plt.figure(figsize=(8,6))
- sns.heatmap(cor, annot=True, fmt = '.2f', cmap='coolwarm')



There is no multi collinearity 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[out].columns:
- 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])
- columns = ['Salary' ,'age', 'educ', 'no_young_children', 'no_older_children']
- for i in columns:
- sns.boxplot(data[i])
- plt.show()



We had outliers in the data. Since LDA works based on numerical computation, treating on outliers will help in improving the model better. After treating outliers, there is no outliers in the data.

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).

- df = pd.get_dummies(data2, columns=['Holliday_Package','foreign'], drop_first = True)
- df.head()

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

- x = df.drop('Holliday_Package_yes', axis=1)
- y = df['Holliday_Package_yes']
- x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30 , random_state=1,st ratify=y)
- grid={'penalty':['l1','l2','none'],
- 'solver':['lbfgs', 'liblinear'],
- 'tol':[0.0001,0.000001]}
- model = LogisticRegression(max_iter=100000,n_jobs=2)
- grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = 3,n_jobs=-1,scoring='f1')
- grid_search.fit(x_train, y_train) //**To find the optimal solving and parameters.**

```
GridSearchCV(cv=3, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100000, multi_class='auto',
                                          n_jobs=2, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'penalty': ['l1', 'l2', 'none'],
                         'solver': ['lbfgs', 'liblinear'],
                         'tol': [0.0001, 1e-06]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=0)
```

- print(grid_search.best_params_,'\n') // **liblinear is suitable for small datasets**
- print(grid_search.best_estimator_)

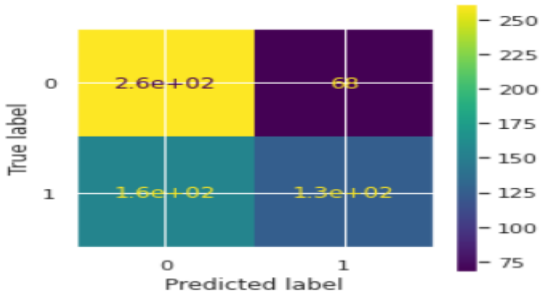
```
{'penalty': 'l2', 'solver': 'liblinear', 'tol': 1e-06}

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100000,
                   multi_class='auto', n_jobs=2, penalty='l2',
                   random_state=None, solver='liblinear', tol=1e-06, verbose=0,
                   warm_start=False)
```

- best_model = grid_search.best_estimator_
- ytrain_predict = best_model.predict(x_train) //**Predicting the training dataset**
- ytest_predict = best_model.predict(x_test)
- ytest_predict_prob=best_model.predict_proba(x_test) // **Probabilities on test dataset**
- pd.DataFrame(ytest_predict_prob).head()

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

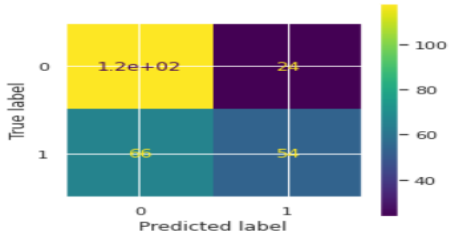
- plot_confusion_matrix(best_model,x_train,y_train) // **Confusion matrix for train set**
- print(classification_report(y_train, ytrain_predict),'\n');

```
              precision    recall  f1-score   support

           0       0.63      0.79      0.70       329
           1       0.65      0.45      0.53       281

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



- plot_confusion_matrix(best_model,x_test,y_test) //**Confusion matrix for test set**
- print(classification_report(y_test, ytest_predict),'\n');

```
              precision    recall  f1-score   support

           0       0.64      0.83      0.72       142
           1       0.69      0.45      0.55       120

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



- lr_train_acc = best_model.score(x_train, y_train)
- lr_train_acc

```
0.6344262295081967
```

 # predicting probabilities

- probs = best_model.predict_proba(x_train)

 # keeping probabilities for the positive outcome only

- probs = probs[:, 1]

 # calculating AUC

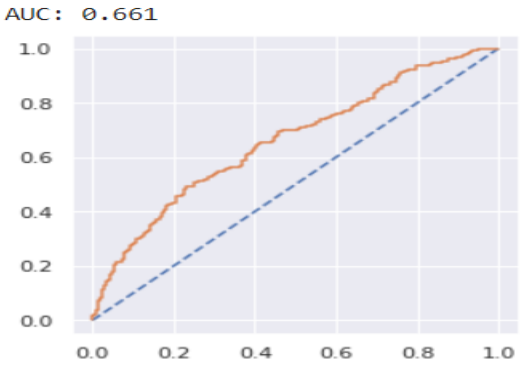
- lr_train_auc = roc_auc_score(y_train, probs)
- print('AUC: %.3f' % lr_train_auc)

 # calculating roc curve

- train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)

- plt.plot([0, 1], [0, 1], linestyle='--')

  # plotting the roc curve for the model
- plt.plot(train_fpr, train_tpr)



- lr_test_acc = best_model.score(x_test, y_test)
- lr_test_acc

```
0.6564885496183206
```

 # predicting probabilities

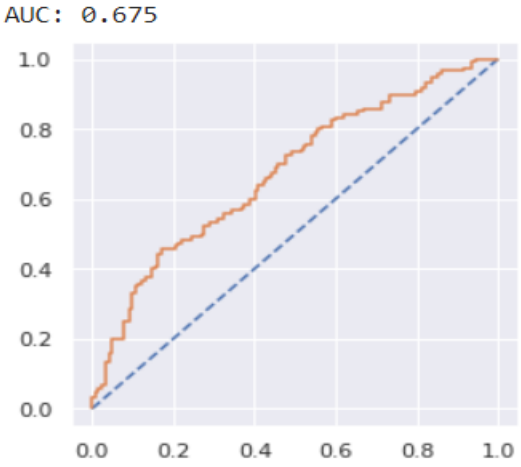
- probs = best_model.predict_proba(x_test)

  # keeping probabilities for the positive outcome only
- probs = probs[:, 1]

  # calculating AUC
- lr_test_auc = roc_auc_score(y_test, probs)
- print('AUC: %.3f' % lr_test_auc)

  # calculating roc curve
- test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
- plt.plot([0, 1], [0, 1], linestyle='--')

  # plotting the roc curve for the model
- plt.plot(test_fpr, test_tpr);



- lr_metrics=classification_report(y_train, ytrain_predict,output_dict=True)
- df=pd.DataFrame(lr_metrics).transpose()
- lr_train_f1=round(df.loc["1"][2],2)
- lr_train_recall=round(df.loc["1"][1],2)
- lr_train_precision=round(df.loc["1"][0],2)
- print ('lr_train_precision ',lr_train_precision)
- print ('lr_train_recall ',lr_train_recall)
- print ('lr_train_f1 ',lr_train_f1)

```
lr_train_precision  0.65
lr_train_recall  0.45
lr_train_f1  0.53
```

- lr_metrics=classification_report(y_test, ytest_predict,output_dict=True)
- df=pd.DataFrame(lr_metrics).transpose()
- lr_test_f1=round(df.loc["1"][2],2)
- lr_test_recall=round(df.loc["1"][1],2)
- lr_test_precision=round(df.loc["1"][0],2)
- print ('lr_test_precision ',lr_test_precision)
- print ('lr_test_recall ',lr_test_recall)
- print ('lr_test_f1 ',lr_test_f1)

```
lr_test_precision  0.69
lr_test_recall  0.45
lr_test_f1  0.55
```

#LDA

- X = data1.drop('Holliday_Package',axis=1)
- Y = data1.pop('Holliday_Package')
- X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.30, random_state=1,stratify = Y)
  # Copying all the predictor variables into X dataframe
- X = df.drop('Holliday_Package_yes', axis=1)
  # Copying target into the y dataframe.
- Y = df['Holliday_Package_yes']
- clf = LinearDiscriminantAnalysis()
- model=clf.fit(X_train,Y_train)
  # Training Data Class Prediction with a cut-off value of 0.5
- pred_class_train = model.predict(X_train)
  # Testing Data Class Prediction with a cut-off value of 0.5

- pred_class_test = model.predict(X_test)

  # Training Data Probability Prediction
- pred_prob_train = model.predict_proba(X_train)

  # Testing Data Probability Prediction
- pred_prob_test = model.predict_proba(X_test)
- lda_train_acc = model.score(X_train,Y_train)
- lda_train_acc

```
0.6327868852459017
```

- print(classification_report(Y_train, pred_class_train))

```
              precision    recall  f1-score   support

           0       0.62      0.80      0.70       329
           1       0.65      0.44      0.52       281

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

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

```
0.6564885496183206
```

- print(classification_report(Y_test, pred_class_test))

```
                        precision    recall  f1-score   support

                     0       0.64      0.83      0.72       142
                     1       0.69      0.45      0.55       120

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

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.

 # AUC and ROC for the training data

 # calculating AUC

- lda_train_auc = metrics.roc_auc_score(Y_train,pred_prob_train[:,1])
- print('AUC for the Training Data: %.3f' % lda_train_auc)

  # calculating roc curve
- fpr, tpr, thresholds = metrics.roc_curve(Y_train,pred_prob_train[:,1])
- plt.plot([0, 1], [0, 1], linestyle='--')

  # plotting the roc curve for the model

- plt.plot(fpr, tpr, marker='.',label = 'Training Data')
  # AUC and ROC for the test data
  # calculating AUC
- lda_test_auc = metrics.roc_auc_score(Y_test,pred_prob_test[:,1])
- print('AUC for the Test Data: %.3f' % lda_test_auc)
  # calculating roc curve
- fpr, tpr, thresholds = metrics.roc_curve(Y_test,pred_prob_test[:,1])
- plt.plot([0, 1], [0, 1], linestyle='--')
  # plotting the roc curve for the model
- plt.plot(fpr, tpr, marker='.',label='Test Data')
  # showing the plot
- plt.legend(loc='best')
- plt.show()

```
AUC for the Training Data: 0.661
AUC for the Test Data: 0.675
```

- lda_metrics=classification_report(Y_train, pred_class_train,output_dict=True)
- df=pd.DataFrame(lda_metrics).transpose()
- lda_train_f1=round(df.loc["1"][2],2)
- lda_train_recall=round(df.loc["1"][1],2)
- lda_train_precision=round(df.loc["1"][0],2)
- print ('lda_train_precision ',lda_train_precision)
- print ('lda_train_recall ',lda_train_recall)
- print ('lda_train_f1 ',lr_train_f1)

```
lda_train_precision  0.65
lda_train_recall  0.44
lda_train_f1  0.53
```

- lda_metrics=classification_report(Y_test, pred_class_test,output_dict=True)
- df=pd.DataFrame(lda_metrics).transpose()
- lda_test_f1=round(df.loc["1"][2],2)
- lda_test_recall=round(df.loc["1"][1],2)
- lda_test_precision=round(df.loc["1"][0],2)

- print ('lda_test_precision ',lda_test_precision)
- print ('lda_test_recall ',lda_test_recall)
- print ('lda_test_f1 ',lda_test_f1)

```
lda_test_precision  0.69
lda_test_recall  0.45
lda_test_f1  0.55
```

Comparing both Logistic regression and LDA model, it is found that both results are same. But LDA works better because there is a category target variable.

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

This business problem helps to predict whether an employee would opt holiday package or not. Both Logistic regression and linear discriminant analysis model are performed and both the results are same.

Some of the interpretations are:

- Customers above the age of 50 are not much interested in opting holiday package.
- People of age 30-50 seem to be much interested in holiday packages.
- People having salary less than 50000 have opted for holiday package.
- The most important factors are salary, age, foreign and holiday package.
- In order to improve the business, we can opt more locations for age old people lie temples and other religious places.
- Offering different packages for different salary people also helps in improving business.
- For employees having more children, opting holiday vacation places will be more recommended for more profit.