In [105]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.stats as stats
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import scale
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,c
```

In [106]:

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

In [4]:

```
df.head()
```

Out[4]:

	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

In [5]:

```
#Dropping unnecessary column:
df=df.drop('Unnamed: 0',axis=1)
```

In [6]:

```
df.head()
```

Out[6]:

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

In [7]:

```
df.shape
```

Out[7]:

```
(872, 7)
```

In [119]:

```
#checking data types:
df.info()
```

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

In [9]:

```
#Checking for missing values:
df.isnull().sum()
```

Out[9]:

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

In [10]:

```
df.isna().sum()
```

Out[10]:

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

In [11]:

```
#Checking for duplicate values:
df.duplicated().sum()
```

Out[11]:

```
0
```

In [12]:

```
#Checking statistical summary:
df.describe(include='all')
```

Out[12]:

	Holliday_Package	Salary	age	educ	no_young_children	no_older_child
count	872	872.000000	872.000000	872.000000	872.000000	872.000
unique	2	NaN	NaN	NaN	NaN	N
top	no	NaN	NaN	NaN	NaN	N
freq	471	NaN	NaN	NaN	NaN	N
mean	NaN	47729.172018	39.955275	9.307339	0.311927	0.982
std	NaN	23418.668531	10.551675	3.036259	0.612870	1.086
min	NaN	1322.000000	20.000000	1.000000	0.000000	0.000
25%	NaN	35324.000000	32.000000	8.000000	0.000000	0.000
50%	NaN	41903.500000	39.000000	9.000000	0.000000	1.000
75%	NaN	53469.500000	48.000000	12.000000	0.000000	2.000
max	NaN	236961.000000	62.000000	21.000000	3.000000	6.000

In [13]:

```
#Checking unique counts for all variables:
for feature in df.columns:
        print(feature)
        print(df[feature].value_counts())
        print(df[feature].nunique())
        print('\n')
```

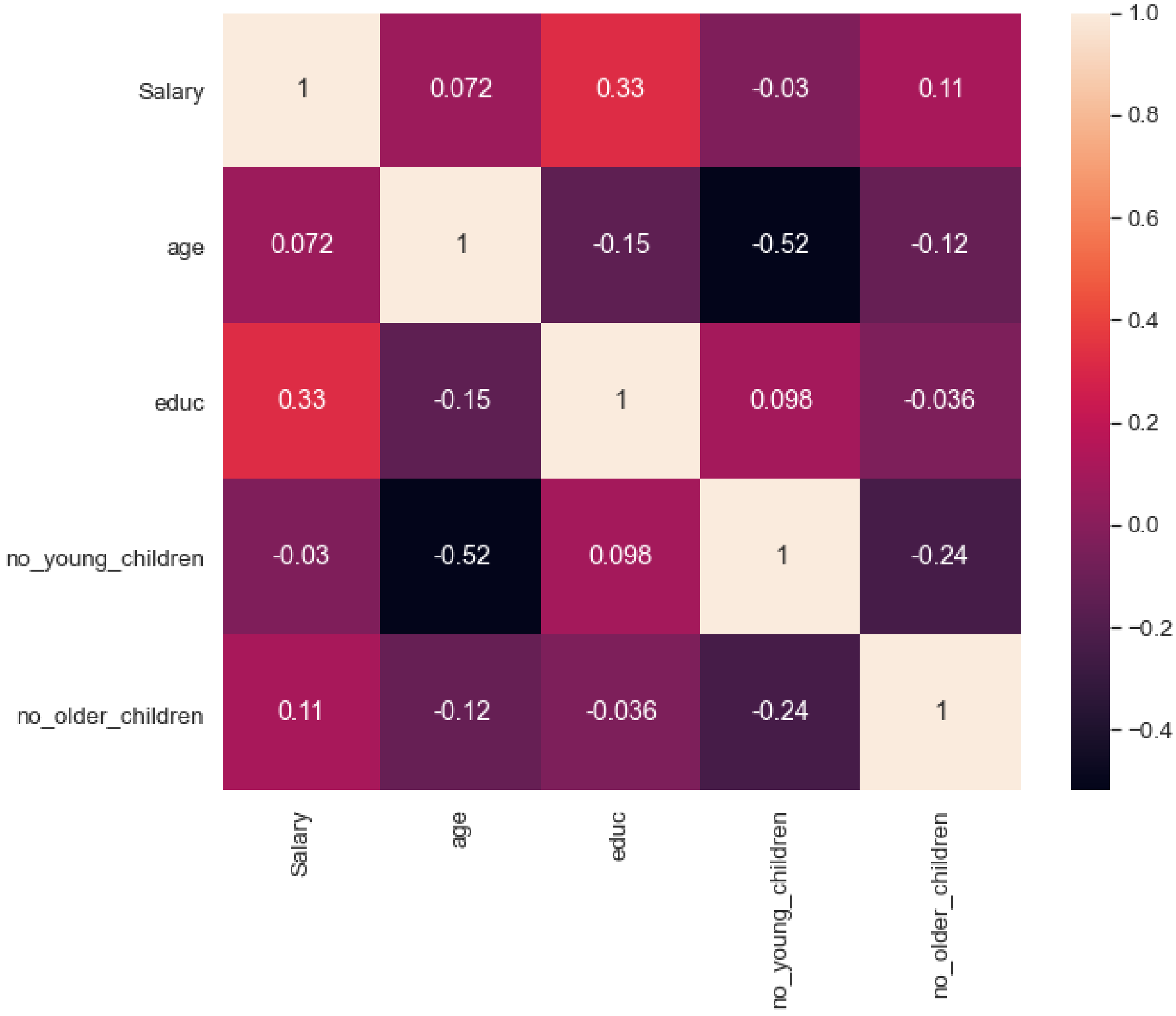
```
Holliday_Package
no     471
yes    401
Name: Holliday_Package, dtype: int64
2


Salary
46195     2
33357     2
39460     2
36976     2
40270     2
         ..
38352     1
119644    1
96072     1
115431    1
74659     1
Name: Salary, Length: 864, dtype: int64
```

In [14]:
```
#Checking proportion of target variable:
df.Holliday_Package.value_counts(normalize=True)
```

Out[14]:
```
no     0.540138
yes    0.459862
Name: Holliday_Package, dtype: float64
```

In [ ]:
```
```

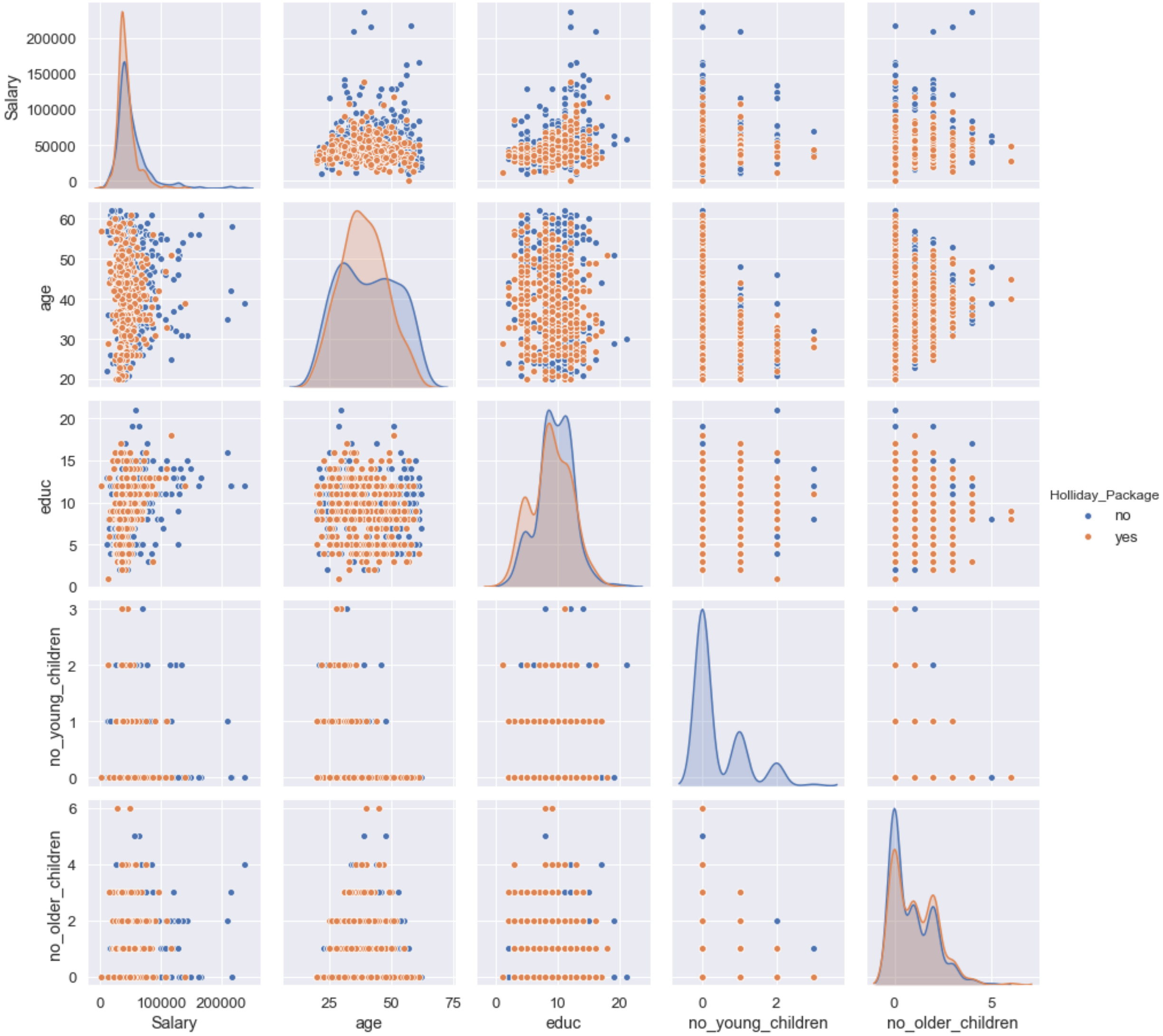
In [16]:
```
#Checking correlation between the continuous variables:
plt.figure(figsize=(10,8))
sns.set(font_scale=1.2)
sns.heatmap(df.corr(),annot=True,)
plt.show()
```



```
In [69]: sns.pairplot(df , hue='Holliday_Package' , diag_kind = 'kde')
```

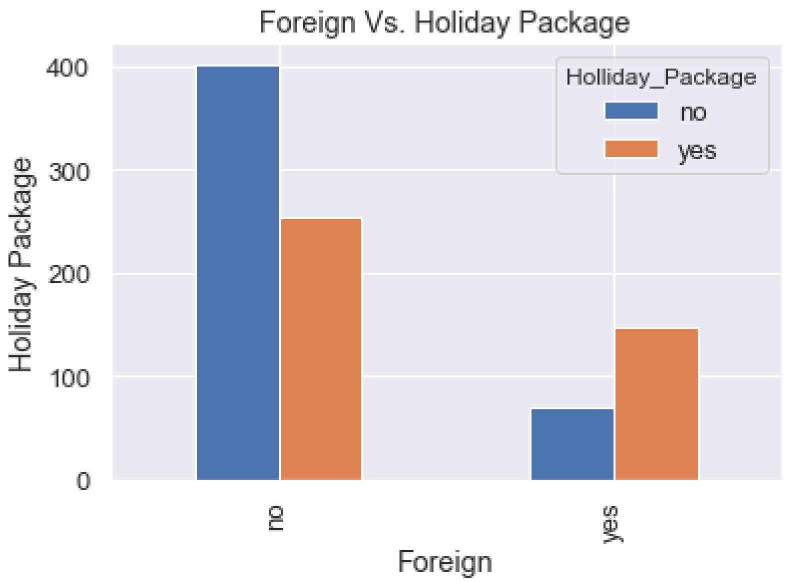
```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/seaborn/distribution
s.py:369: UserWarning: Default bandwidth for data is 0; skipping density
estimation.
  warnings.warn(msg, UserWarning)
```

```
Out[69]: <seaborn.axisgrid.PairGrid at 0x125b56be0>
```

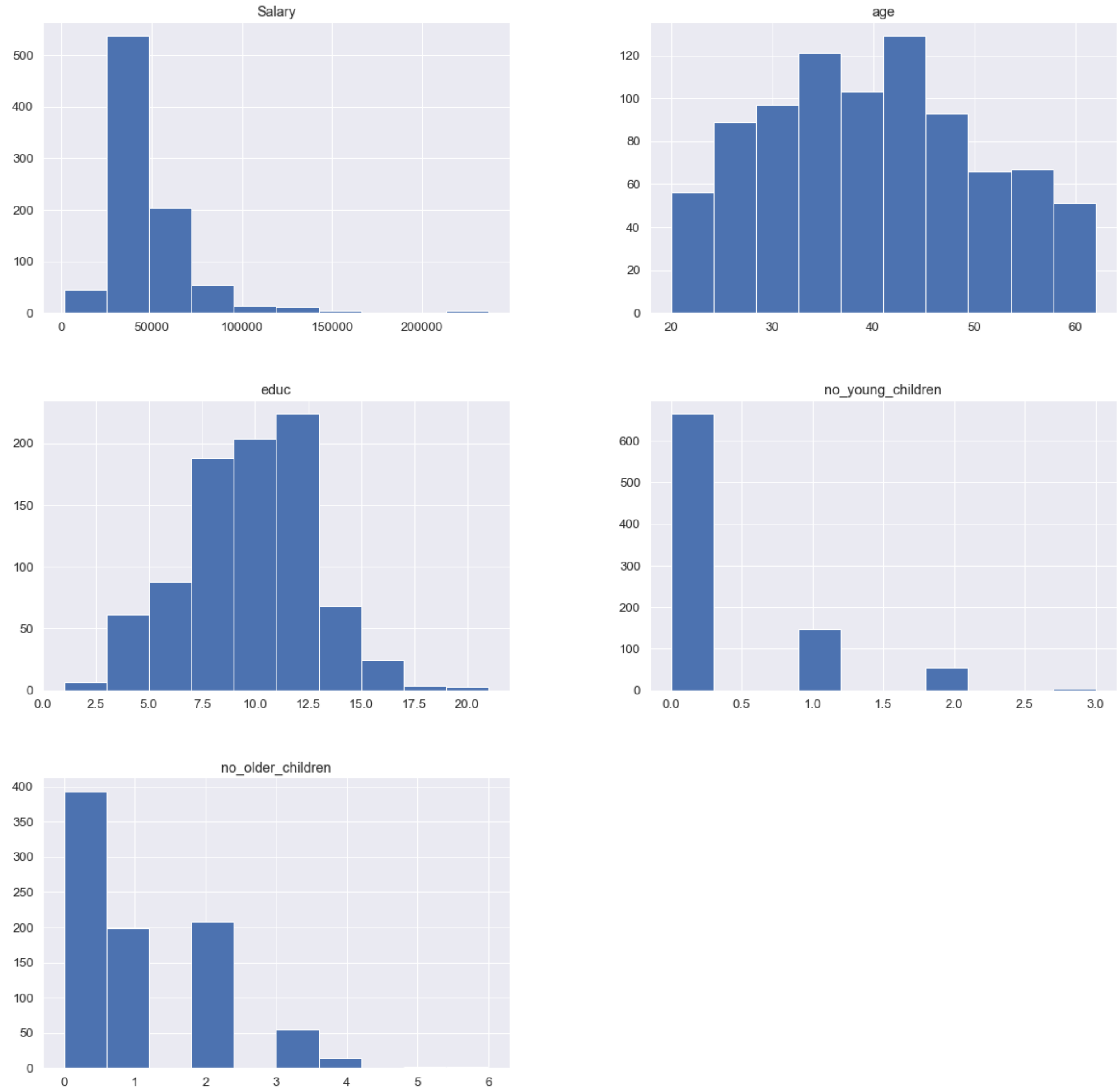


In [17]:
```
pd.crosstab(df.foreign,df.Holliday_Package).plot(kind='bar')
plt.title('Foreign Vs. Holiday Package')
plt.xlabel('Foreign')
plt.ylabel('Holiday Package')
```

Out[17]:
```
Text(0, 0.5, 'Holiday Package')
```

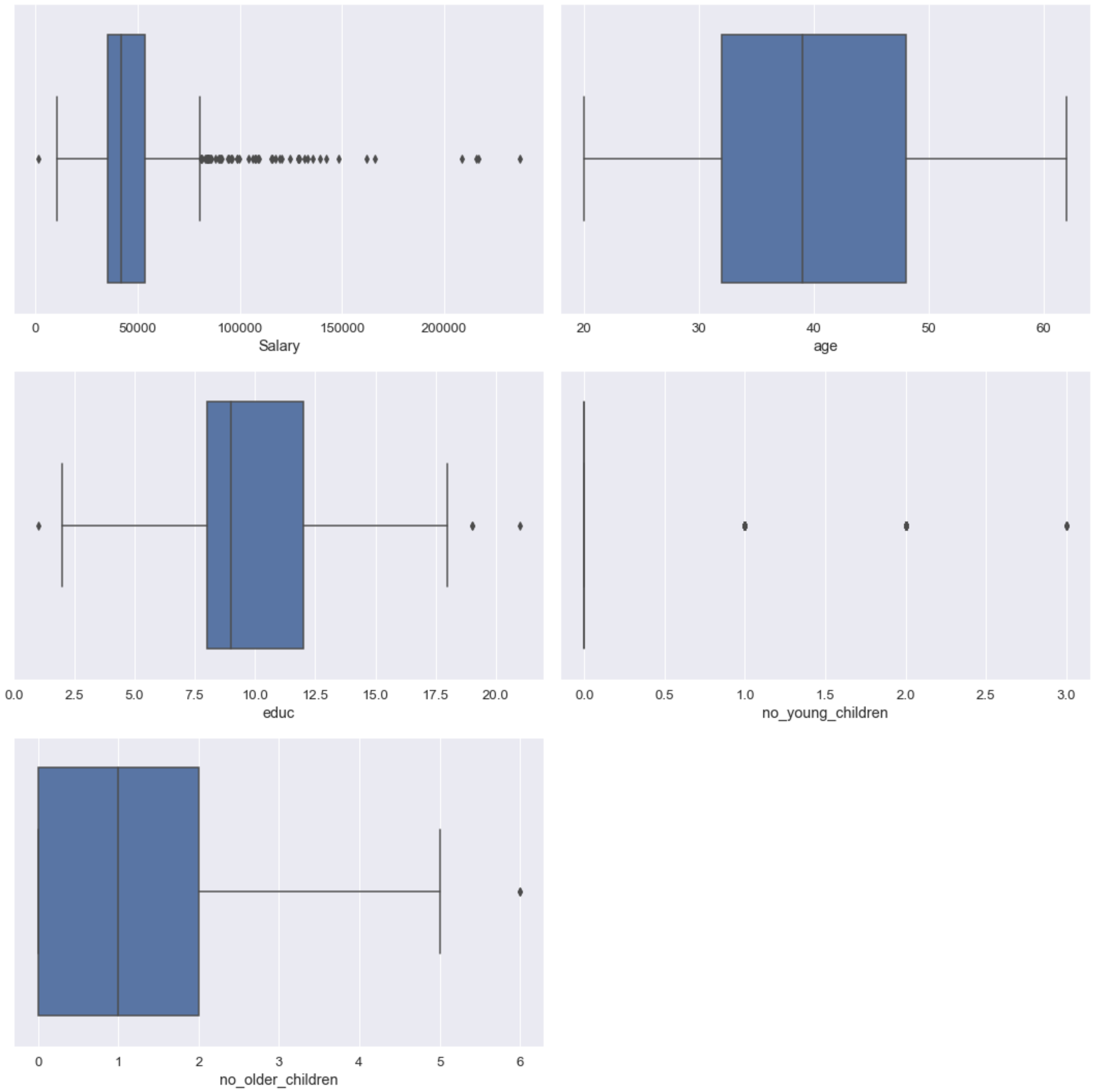


```
In [37]: from pylab import rcParams
         rcParams['figure.figsize'] = 20,20
         df[['Salary','age','educ','no_young_children','no_older_children']].hist();
```



In [51]:

```
#Checking for outliers in continuous variables:

data_plot=df[['Salary','age','educ','no_young_children','no_older_children'
fig=plt.figure(figsize=(15,15))
for i in range(0,len(data_plot.columns)):
    ax=fig.add_subplot(3,2,i+1)
    sns.boxplot(data_plot[data_plot.columns[i]])
    plt.tight_layout()
```



In [ ]:

```

```

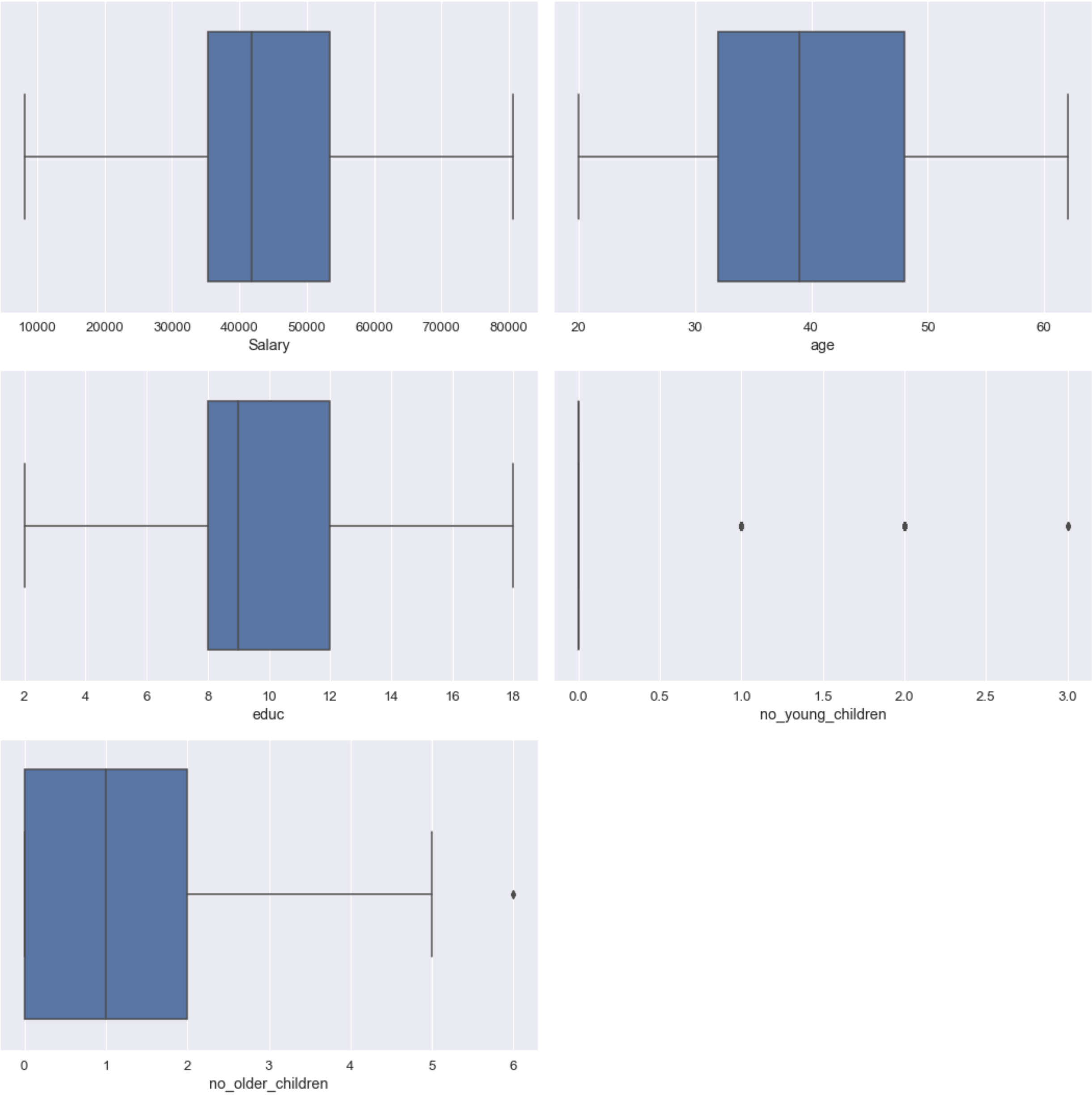
In [120]:

```python
#Treating the outliers:
def remove_outlier(col):
    sorted(col)
    Q1,Q3=col.quantile([0.25,0.75])
    IQR=Q3-Q1
    lower_range=Q1-(1.5*IQR)
    upper_range=Q3+(1.5*IQR)
    return lower_range,upper_range
```

In [121]:

```python
#Replacing outliers using capping and flooring

lrsalary,upsalary=remove_outlier(df['Salary'])
df['Salary']=np.where(df['Salary']>upsalary,upsalary,df['Salary'])
df['Salary']=np.where(df['Salary']<lrsalary,lrsalary,df['Salary'])

lredu,upedu=remove_outlier(df['educ'])
df['educ']=np.where(df['educ']>upedu,upedu,df['educ'])
df['educ']=np.where(df['educ']<lredu,lredu,df['educ'])
```

In [57]:
```
#Rechecking for outliers:
data_plot=df[['Salary','age','educ','no_young_children','no_older_children'
fig=plt.figure(figsize=(15,15))
for i in range(0,len(data_plot.columns)):
    ax=fig.add_subplot(3,2,i+1)
    sns.boxplot(data_plot[data_plot.columns[i]])
    plt.tight_layout()
```



In [122]:

```
#Converting object type data to categorical codes:
code1={"no":0, "yes":1}
df["Holliday_Package"]=df["Holliday_Package"].replace(code1)
```

In [123]:

```
code2={"no":0, "yes":1}
df["foreign"]=df["foreign"].replace(code2)
```

In [124]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 872 entries, 0 to 871
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         872 non-null    int64  
 1   Holliday_Package   872 non-null    int64  
 2   Salary             872 non-null    float64
 3   age                872 non-null    int64  
 4   educ               872 non-null    float64
 5   no_young_children  872 non-null    int64  
 6   no_older_children  872 non-null    int64  
 7   foreign            872 non-null    int64  
dtypes: float64(2), int64(6)
memory usage: 54.6 KB
```

In [125]:

```
#Cross-checking the categorical coding:
for column in df[['Holliday_Package', 'foreign']]:
    print(df[column].value_counts().sort_values())
    print('\n')
```

```
1    401
0    471
Name: Holliday_Package, dtype: int64


1    216
0    656
Name: foreign, dtype: int64
```

In [63]:

```
#Splitting the data into train and test sets:
x = df.drop('Holliday_Package', axis=1)
y = df[['Holliday_Package']]
```

In [65]:

```
#Splitting in the 70:30 ratio:

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

In [66]:

```
# Fitting the Logistic Regression model:
model = LogisticRegression(solver='newton-cg',max_iter=10000,penalty='none'
model.fit(x_train, y_train)
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/sklearn/utils/valida
tion.py:63: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent worker
s.
[Parallel(n_jobs=2)]: Done   1 out of   1 | elapsed:    3.6s finished
```

Out[66]:

```
LogisticRegression(max_iter=10000, n_jobs=2, penalty='none', solver='newt
on-cg',
                   verbose=True)
```

In [67]:

```
#Predicting on training and test data:
ytrain_predict = model.predict(x_train)
ytest_predict = model.predict(x_test)
```

In [68]:

```
#Checking predicted class and probabilities:
ytest_predict_prob=model.predict_proba(x_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[68]:

	0	1
0	0.773162	0.226838
1	0.273315	0.726685
2	0.903230	0.096770
3	0.958335	0.041665
4	0.512241	0.487759

In [82]:

```
#Finding accuracy of training data:
lr_train_acc=model.score(x_train, y_train)
lr_train_acc
```

Out[82]:

```
0.6754098360655738
```

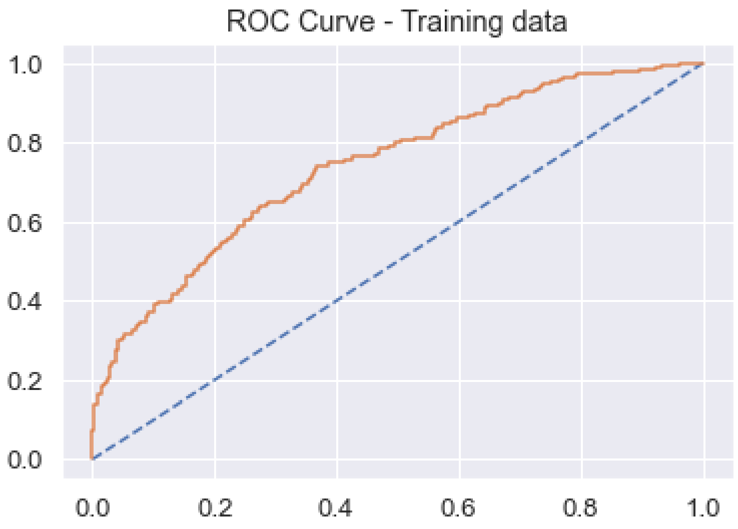
In [83]:

```
#Calculating AUC and plotting ROC curve for training data:
probs = model.predict_proba(x_train)
probs = probs[:, 1]

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

%matplotlib inline
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

plt.plot(train_fpr, train_tpr);
plt.title("ROC Curve - Training data")
```

```
AUC: 0.742
```

Out[83]:

```
Text(0.5, 1.0, 'ROC Curve - Training data')
```



In [84]:

```
# Finding Accuracy for Test Data:
lr_test_acc=model.score(x_test, y_test)
lr_test_acc
```

Out[84]:

```
0.6374045801526718
```

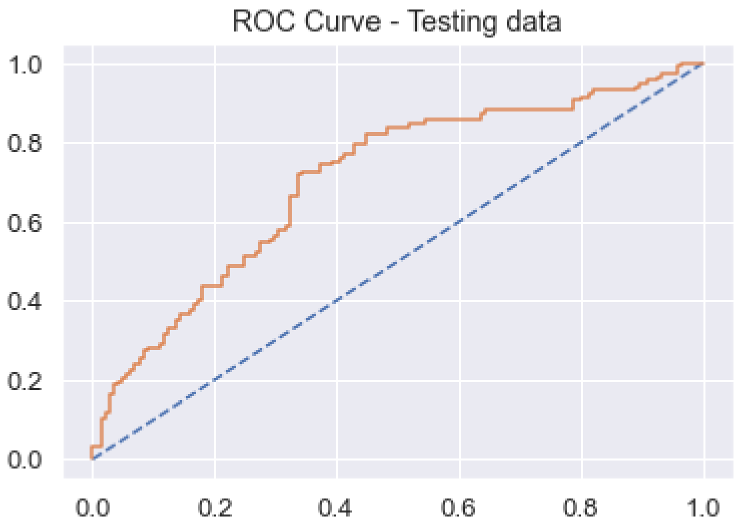
In [85]:

```
##Calculating AUC and plotting ROC curve for test data:
probs = model.predict_proba(x_test)

probs = probs[:, 1]

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

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

plt.plot(test_fpr, test_tpr);
plt.title("ROC Curve - Testing data")
```

```
AUC: 0.705
```

Out[85]:

```
Text(0.5, 1.0, 'ROC Curve - Testing data')
```



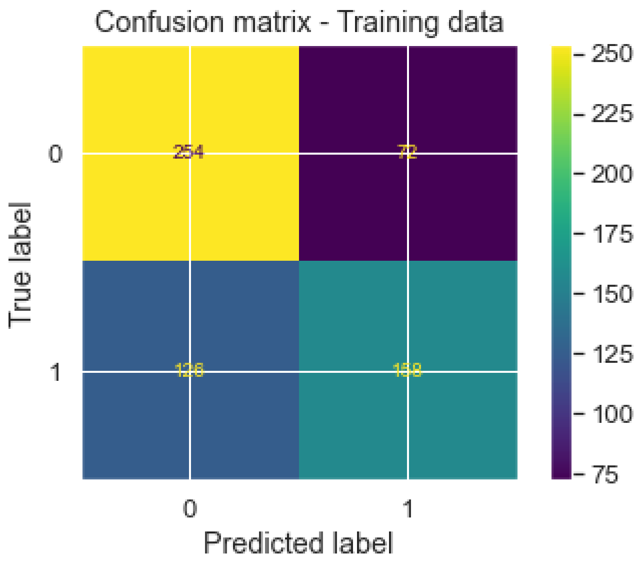
In [75]:

```
#Building confusion matrix for training data:
confusion_matrix(y_train, ytrain_predict)
```

Out[75]:

```
array([[252,  74],
       [124, 160]])
```

```
In [145]: plot_confusion_matrix(model,x_train,y_train);
          plt.title("Confusion matrix - Training data")

Out[145]: Text(0.5, 1.0, 'Confusion matrix - Training data')
```



```
In [77]: print(classification_report(y_train, ytrain_predict))

                       precision    recall  f1-score   support

                   0       0.67      0.77      0.72       326
                   1       0.68      0.56      0.62       284

            accuracy                           0.68       610
           macro avg       0.68      0.67      0.67       610
        weighted avg       0.68      0.68      0.67       610
```

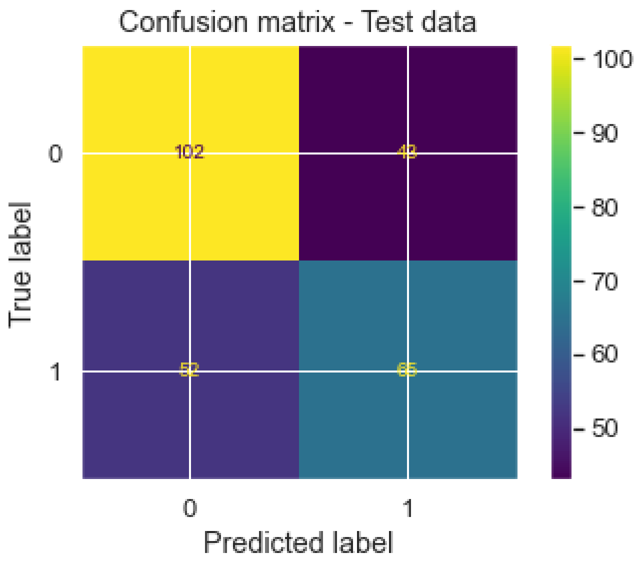
```
In [ ]:
```

```
In [78]: #Building confusion matrix for test data:
         confusion_matrix(y_test, ytest_predict)
```

```
Out[78]: array([[102,  43],
                [ 52,  65]])
```

```
In [146]: plot_confusion_matrix(model,x_test,y_test);
          plt.title("Confusion matrix - Test data")
```

```
Out[146]: Text(0.5, 1.0, 'Confusion matrix - Test data')
```



```
In [80]: print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.66      0.70      0.68       145
           1       0.60      0.56      0.58       117

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

```
In [95]: lr_train_recall=0.56
         lr_train_precision=0.68
         lr_train_F1=0.62
```

In [99]:

```
#Storing data for comparison:
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'Log-reg Train':[lr_train_acc,lr_train_auc,lr_train_re
                    'Log-reg Test':[lr_test_acc,lr_test_auc,'0.56','0.60',
round(data,2)
```

Out[99]:

	Log-reg Train	Log-reg Test
Accuracy	0.68	0.637405
AUC	0.74	0.705158
Recall	0.56	0.56
Precision	0.68	0.60
F1 Score	0.62	0.58

In [127]:

```
#Creating a copy of dataset to perform LDA:
df1=df.copy()
df1.head()
```

Out[127]:

	Unnamed: 0	Holliday_Package	Salary	age	educ	no_young_children	no_older_children	foreign
0	1	0	48412.0	30	8.0	1	1	0
1	2	1	37207.0	45	8.0	0	1	0
2	3	0	58022.0	46	9.0	0	0	0
3	4	0	66503.0	31	11.0	2	0	0
4	5	0	66734.0	44	12.0	0	2	0

In [128]:

```
#Dropping unnecessary column:
df1=df1.drop('Unnamed: 0',axis=1)
```

In [129]:

```
#Splitting the data into train and test sets:
X = df1.drop('Holliday_Package', axis=1)
Y = df1[['Holliday_Package']]
```

In [113]:

```

```

In [130]:

```

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.30,random_
```

In [131]:

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

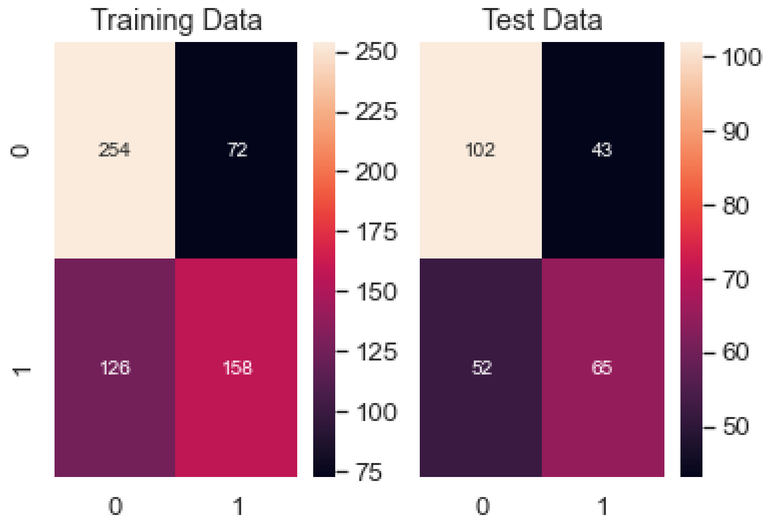
```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/sklearn/utils/valida
tion.py:63: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
```

In [132]:

```
#Making predictions on the training data with 0.5 cut-off value:

pred_class_train = model.predict(X_train)

# Test Data Class Prediction with a cut-off value of 0.5:
pred_class_test = model.predict(X_test)
```

In [136]:

```
f,a =  plt.subplots(1,2,sharex=True,sharey=True,squeeze=False)
fig=plt.figure(figsize=(20,20))
#Plotting confusion matrix for the different models for the Training Data

plot_0 = sns.heatmap((metrics.confusion_matrix(Y_train,pred_class_train)),a
a[0][0].set_title('Training Data')

plot_1 = sns.heatmap((metrics.confusion_matrix(Y_test,pred_class_test)),ann
a[0][1].set_title('Test Data');
```



```
<Figure size 1440x1440 with 0 Axes>
```

In [137]:
```
#Classification report for training and test:
print('Classification Report of the training data:\n\n',metrics.classificat
print('Classification Report of the test data:\n\n',metrics.classification_
```

```
Classification Report of the training data:

               precision    recall  f1-score   support

           0       0.67      0.78      0.72       326
           1       0.69      0.56      0.61       284

    accuracy                           0.68       610
   macro avg       0.68      0.67      0.67       610
weighted avg       0.68      0.68      0.67       610


Classification Report of the test data:

               precision    recall  f1-score   support

           0       0.66      0.70      0.68       145
           1       0.60      0.56      0.58       117

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

In [138]:
```
# Training Data Probability Prediction
pred_prob_train = model.predict_proba(X_train)

# Test Data Probability Prediction
pred_prob_test = model.predict_proba(X_test)
```

In [140]:
```
Ytest_predict_prob=model.predict_proba(X_test)
pd.DataFrame(Ytest_predict_prob).head()
```

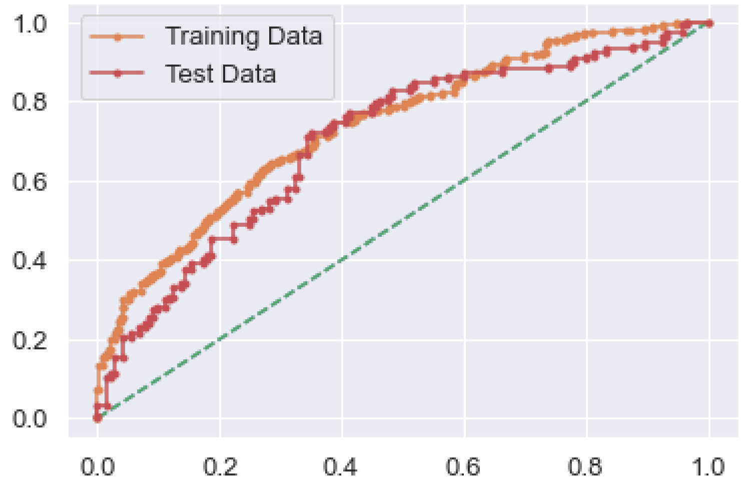
Out[140]:

	0	1
0	0.763849	0.236151
1	0.278109	0.721891
2	0.887980	0.112020
3	0.950289	0.049711
4	0.507535	0.492465

In [141]:

```
# AUC and ROC for the training data

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

#  calculate and plot roc curve
fpr, tpr, thresholds = metrics.roc_curve(Y_train,pred_prob_train[:,1])
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.',label = 'Training Data')


# AUC and ROC for the test data

# calculate AUC
auc = metrics.roc_auc_score(Y_test,pred_prob_test[:,1])
print('AUC for the Test Data: %.3f' % auc)

#  calculate and plot roc curve
fpr, tpr, thresholds = metrics.roc_curve(Y_test,pred_prob_test[:,1])
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.',label='Test Data')
# show the plot
plt.legend(loc='best')
plt.show()
```

```
AUC for the Training Data: 0.739
AUC for the Test Data: 0.703
```



In [142]:

```
#Storing data for comparison:
LDA_train_acc=0.68
LDA_test_acc=0.64
LDA_train_auc=0.739
LDA_test_auc=0.703
LDA_train_recall=0.56
LDA_test_recall=0.56
LDA_train_precision=0.69
LDA_test_precision=0.60
LDA_train_f1=0.61
LDA_test_f1=0.58
```

In [144]:

```
ndex=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
ata = pd.DataFrame({'LDA Train':[LDA_train_acc,LDA_train_auc,LDA_train_recal
                    'LDA Test':[LDA_test_acc,LDA_test_auc,LDA_test_recall,LI
ound(data,2)
```

Out[144]:

	LDA Train	LDA Test
Accuracy	0.68	0.64
AUC	0.74	0.70
Recall	0.56	0.56
Precision	0.69	0.60
F1 Score	0.61	0.58

In [ ]: