In [21]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings("ignore")

from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_score,f1_score,
classification_report
from scipy.stats import chi2_contingency

import statsmodels.api as sm

from statsmodels.formula.api import ols

from sklearn.model_selection import train_test_split

import xgboost as xgb
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
```

In [22]:

```
df = pd.read_csv('HFEA20.12.csv')
```

In [23]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80488 entries, 0 to 80487
Data columns (total 26 columns):
Patient_Age_at_Treatment                                          80488 non-null object
Total_Number_of_Previous_treatments_Both_IVF_and_DI_at_clinic     80488 non-null int64
Total_Number_of_Previous_IVF_cycles                               80488 non-null int64
Total_number_of_previous_pregnancies_Both_IVF_and_DI              80488 non-null int64
Total_number_of_IVF_pregnancies                                   80488 non-null int64
Stimulation_used                                                  80488 non-null int64
Donated_embryo                                                    79621 non-null float64
Specific_treatment_type                                           80488 non-null object
Elective_Single_Embryo_Transfer                                   79621 non-null float64
Egg_Source                                                        79621 non-null object
Sperm_From                                                        80488 non-null object
Fresh_Cycle                                                       79621 non-null float64
Eggs_Thawed                                                       79621 non-null float64
Fresh_Eggs_Collected                                              79621 non-null float64
Fresh_Eggs_Stored                                                 79621 non-null float64
Total_Eggs_Mixed                                                  79621 non-null float64
Eggs_Mixed_With_Partner_Sperm                                     79621 non-null float64
Eggs_Mixed_With_Donor_sperm                                       79621 non-null float64
Total_Embryos_Created                                             79621 non-null float64
Eggs_Microinjected                                                79621 non-null float64
Embryos_from_Eggs_Microinjected                                   79621 non-null float64
Total_Embryos_Thawed                                              79621 non-null float64
Embryos_Transfered                                                79621 non-null float64
Embryos_Transfered_from_Eggs_Microinjected                        79621 non-null float64
Embryos_Stored_For_Use_By_Patient                                 79621 non-null float64
Number_of_Live_Births                                             80488 non-null int64
dtypes: float64(16), int64(6), object(4)
memory usage: 16.0+ MB
```

Null value Removal

In [24]:

```
df.dropna(axis = 0,inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 79621 entries, 0 to 79620
Data columns (total 26 columns):
Patient_Age_at_Treatment                                        79621 non-null object
Total_Number_of_Previous_treatments_Both_IVF_and_DI_at_clinic   79621 non-null int64
Total_Number_of_Previous_IVF_cycles                             79621 non-null int64
Total_number_of_previous_pregnancies_Both_IVF_and_DI            79621 non-null int64
Total_number_of_IVF_pregnancies                                 79621 non-null int64
Stimulation_used                                                79621 non-null int64
Donated_embryo                                                  79621 non-null float64
Specific_treatment_type                                         79621 non-null object
Elective_Single_Embryo_Transfer                                 79621 non-null float64
Egg_Source                                                      79621 non-null object
Sperm_From                                                      79621 non-null object
Fresh_Cycle                                                     79621 non-null float64
Eggs_Thawed                                                     79621 non-null float64
Fresh_Eggs_Collected                                            79621 non-null float64
Fresh_Eggs_Stored                                               79621 non-null float64
Total_Eggs_Mixed                                                79621 non-null float64
Eggs_Mixed_With_Partner_Sperm                                   79621 non-null float64
Eggs_Mixed_With_Donor_sperm                                     79621 non-null float64
Total_Embryos_Created                                           79621 non-null float64
Eggs_Microinjected                                              79621 non-null float64
Embryos_from_Eggs_Microinjected                                 79621 non-null float64
Total_Embryos_Thawed                                            79621 non-null float64
Embryos_Transfered                                              79621 non-null float64
Embryos_Transfered_from_Eggs_Microinjected                      79621 non-null float64
Embryos_Stored_For_Use_By_Patient                               79621 non-null float64
Number_of_Live_Births                                           79621 non-null int64
dtypes: float64(16), int64(6), object(4)
memory usage: 16.4+ MB
```

Target Fitness

In [25]:

```
# print("Target fitness",'\n')

target = 'Number_of_Live_Births'

# print(df[target].value_counts(),'\n')

# expected_columns = round((3299*3)**(1/2))

# existed_columns = df.shape[1]

# print('*'*75,'\n')

# print("Expected_columns : {0}    Columns : {1}".format(expected_columns,existed_columns ) )
```

Balancing the target

In [26]:

```
df = df[df[target]!=2]


low = list(df[target].value_counts())[-1]

print("least value of target label : ",low,'\n')

print('*'*45,'\n')

df.reset_index(inplace = True)

df.drop(df.columns[0],axis =1,inplace = True)

df_one = df[df[target]==1].head(low)
```

```
df_zero = df[df[target]==0].head(low)

df = pd.concat([df_one,df_zero],axis = 0)

print(df[target].value_counts(),'\n')
```

```
least value of target label :  19723

*********************************************

1    19723
0    19723
Name: Number_of_Live_Births, dtype: int64
```

In [27]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39446 entries, 0 to 26424
Data columns (total 26 columns):
Patient_Age_at_Treatment                                         39446 non-null object
Total_Number_of_Previous_treatments_Both_IVF_and_DI_at_clinic    39446 non-null int64
Total_Number_of_Previous_IVF_cycles                              39446 non-null int64
Total_number_of_previous_pregnancies_Both_IVF_and_DI             39446 non-null int64
Total_number_of_IVF_pregnancies                                  39446 non-null int64
Stimulation_used                                                 39446 non-null int64
Donated_embryo                                                   39446 non-null float64
Specific_treatment_type                                          39446 non-null object
Elective_Single_Embryo_Transfer                                  39446 non-null float64
Egg_Source                                                       39446 non-null object
Sperm_From                                                       39446 non-null object
Fresh_Cycle                                                      39446 non-null float64
Eggs_Thawed                                                      39446 non-null float64
Fresh_Eggs_Collected                                             39446 non-null float64
Fresh_Eggs_Stored                                                39446 non-null float64
Total_Eggs_Mixed                                                 39446 non-null float64
Eggs_Mixed_With_Partner_Sperm                                    39446 non-null float64
Eggs_Mixed_With_Donor_sperm                                      39446 non-null float64
Total_Embryos_Created                                            39446 non-null float64
Eggs_Microinjected                                               39446 non-null float64
Embryos_from_Eggs_Microinjected                                  39446 non-null float64
Total_Embryos_Thawed                                             39446 non-null float64
Embryos_Transfered                                               39446 non-null float64
Embryos_Transfered_from_Eggs_Microinjected                       39446 non-null float64
Embryos_Stored_For_Use_By_Patient                                39446 non-null float64
Number_of_Live_Births                                            39446 non-null int64
dtypes: float64(16), int64(6), object(4)
memory usage: 8.1+ MB
```

In [28]:

```
df[target].value_counts()
```

Out[28]:

```
1    19723
0    19723
Name: Number_of_Live_Births, dtype: int64
```

In [29]:

```
def cat_col_f(Dataframe,target):

    cat_col = list(Dataframe.select_dtypes(include=['object','category','bool']).columns)

    try:
        cat_col.remove(target)

        return cat_col
        return cat_col

    except:

        return cat_col

def num_col_f(Dataframe,target):

    num_col = list(Dataframe.select_dtypes(include=['int','float']).columns)

    try:
        num_col.remove(target)

        return num_col

    except:

        return num_col


Cat_col = cat_col_f(df,target)

Num_col = num_col_f(df,target)

print("Cat_col :",Cat_col,'\n')

print("Num_col :",Num_col,'\n')
```

```
Cat_col : ['Patient_Age_at_Treatment', 'Specific_treatment_type', 'Egg_Source', 'Sperm_From']

Num_col : ['Donated_embryo', 'Elective_Single_Embryo_Transfer', 'Fresh_Cycle', 'Eggs_Thawed', 'Fre
sh_Eggs_Collected', 'Fresh_Eggs_Stored', 'Total_Eggs_Mixed', 'Eggs_Mixed_With_Partner_Sperm',
'Eggs_Mixed_With_Donor_sperm', 'Total_Embryos_Created', 'Eggs_Microinjected',
'Embryos_from_Eggs_Microinjected', 'Total_Embryos_Thawed', 'Embryos_Transfered',
'Embryos_Transfered_from_Eggs_Microinjected', 'Embryos_Stored_For_Use_By_Patient']
```

Customised catogorical columns

In [30]:

```
custom_cat_col =
['Stimulation_used','Donated_embryo','Elective_Single_Embryo_Transfer','Fresh_Cycle']
```

Datatype Conversion

In [31]:

```
print(df[custom_cat_col].dtypes)

print('_'*45)
for i in ['Donated_embryo','Elective_Single_Embryo_Transfer','Fresh_Cycle']:

    df[i] = df[i].astype('int')

df[custom_cat_col].dtypes
```

```
Stimulation_used                     int64
Donated_embryo                     float64
Elective_Single_Embryo_Transfer    float64
Fresh_Cycle                        float64
dtype: object
_____________________________________________
```

Out[31]:

```
Stimulation_used                   int64
Donated_embryo                     int32
Elective_Single_Embryo_Transfer    int32
Fresh_Cycle                        int32
Fresh_Cycle                        int32
dtype: object
```

Statistical Analysis

1.Chi2 Analysis

In [32]:

```
def chi2(cat_col,target,Dataframe):
    p_values = []
    for idx,col_name in enumerate(cat_col):
        chi2, p_value, dof, expected = chi2_contingency(pd.crosstab(Dataframe[col_name],Dataframe[target]))
        p_values.append([col_name,round(p_value,3)])
    return dict(p_values)
def pvalue_significance(p_values_dict):
    fea_sel = [key for key,value in p_values_dict.items() if value <= 0.05 ]
    return fea_sel
chi_info = chi2(Cat_col+custom_cat_col,target,df)
cat_sel = pvalue_significance(chi_info)
print(chi_info,'\n')
print("*"*55,"selected columns","*"*55,'\n')
print(cat_sel)
```

```
{'Patient_Age_at_Treatment': 0.0, 'Specific_treatment_type': 0.0, 'Egg_Source': 0.0, 'Sperm_From':
0.0, 'Stimulation_used': 0.473, 'Donated_embryo': 0.0, 'Elective_Single_Embryo_Transfer': 0.0, 'Fr
esh_Cycle': 0.141}

******************************************************* selected columns
*******************************************************

['Patient_Age_at_Treatment', 'Specific_treatment_type', 'Egg_Source', 'Sperm_From',
'Donated_embryo', 'Elective_Single_Embryo_Transfer']
```

2.Anova Analysis

In [33]:

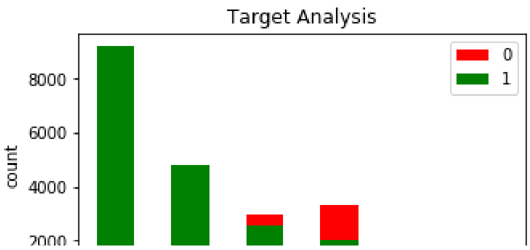
```
def anova(cat_col,target,Dataframe):
    if len(cat_col)>1:
        individual = ' + '.join(cat_col)
        String = str(target)+ ' ~ ' '+'+individual
        mod = ols(String, data = Dataframe).fit()
        aov_table = sm.stats.anova_lm(mod, typ=2)
        anova_fea_imp = []
        for idx,col_name in enumerate(cat_col):
            anova_fea_imp.append([col_name, round(aov_table['PR(>F)'][idx],3)])
        return dict(anova_fea_imp)

    else:

        String = target+ ' ~ ' +cat_col[0]

        mod = ols(String, data = Dataframe).fit()

        aov_table = sm.stats.anova_lm(ols(target+ ' ~ ' +cat_col[0], data = Dataframe).fit(), typ=1
)

        anova_fea_imp = zip(cat_col,[round(aov_table['PR(>F)'][0],3)])

        return dict(anova_fea_imp)


num_col_info = anova(list(set(Num_col)-set(custom_cat_col)),target,df)

num_sel = pvalue_significance(num_col_info)

print(num_col_info,'\n')

print("*"*25,"selected columns","*"*25,'\n')

print(num_sel)
```

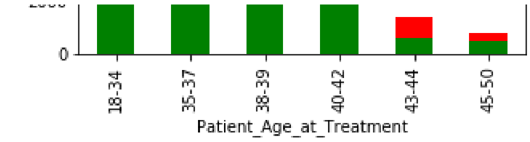
```
{'Embryos_Stored_For_Use_By_Patient': 0.916, 'Fresh_Eggs_Stored': 0.0, 'Eggs_Thawed': 0.405,
'Fresh_Eggs_Collected': 0.0, 'Total_Eggs_Mixed': 0.559, 'Total_Embryos_Created': 0.0,
'Eggs_Microinjected': 0.88, 'Embryos_from_Eggs_Microinjected': 0.084, 'Total_Embryos_Thawed': 0.0,
'Eggs_Mixed_With_Partner_Sperm': 0.077, 'Embryos_Transfered_from_Eggs_Microinjected': 0.002, 'Embr
yos_Transfered': 0.0, 'Eggs_Mixed_With_Donor_sperm': 0.009}

************************* selected columns *************************

['Fresh_Eggs_Stored', 'Fresh_Eggs_Collected', 'Total_Embryos_Created', 'Total_Embryos_Thawed',
'Embryos_Transfered_from_Eggs_Microinjected', 'Embryos_Transfered', 'Eggs_Mixed_With_Donor_sperm']
```

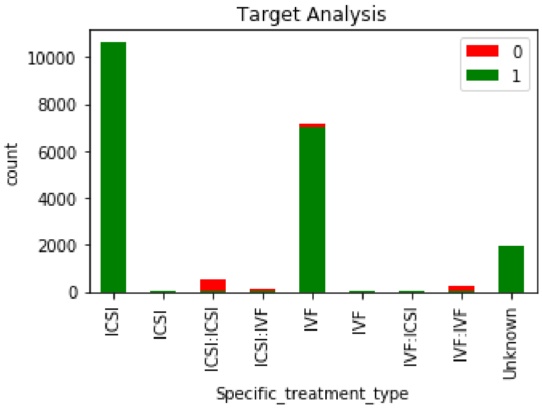
Catogorical column vs Target

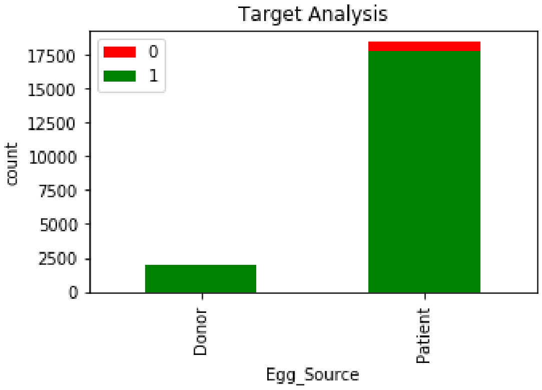
In [51]:

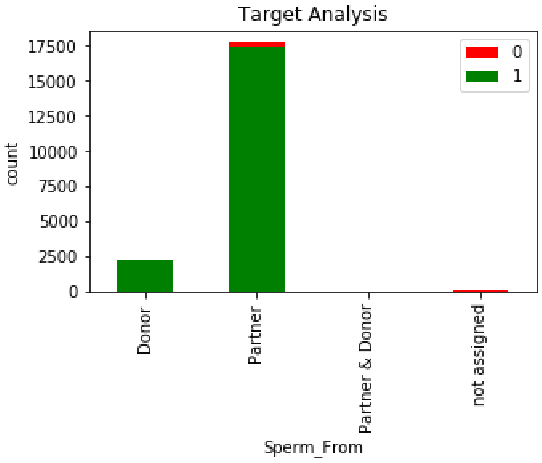
```
def cat_visual(col,target,df):

    plt.figure(figsize=[5,3])

    df_temp = df.groupby([col])[target].apply(lambda x: ((x==0)==True).sum())
    ax = df_temp.plot('bar', rot=90,color = 'red',label='0')

    df_temp = df.groupby([col])[target].apply(lambda x: ((x==1)==True).sum())
    ax=df_temp.plot('bar', rot=90,color = 'g',label='1')


    plt.ylabel('count')
    plt.xlabel(col)
    plt.title('Target Analysis')
    plt.legend()
    plt.show()


for i in cat_sel:

    cat_visual(i,target,df)
```

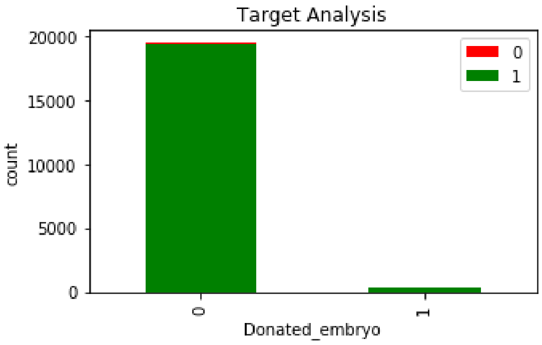



0
18-34
35-37
38-39
40-42
43-44
45-50
Patient_Age_at_Treatment

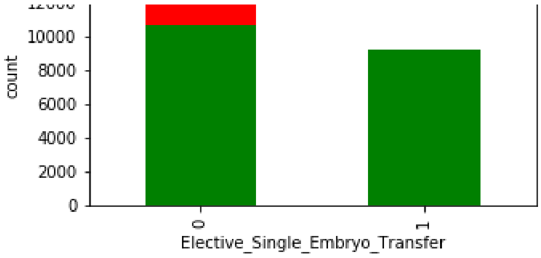

Target Analysis
0
1
count
10000
8000
6000
4000
2000
0
ICSI
ICSI
ICSI:ICSI
ICSI:IVF
IVF
IVF
IVF:ICSI
IVF:IVF
Unknown
Specific_treatment_type


Target Analysis
0
1
count
17500
15000
12500
10000
7500
5000
2500
0
Donor
Patient
Egg_Source


Target Analysis
0
1
count
17500
15000
12500
10000
7500
5000
2500
0
Donor
Partner
Partner & Donor
not assigned
Sperm_From


Target Analysis
0
1
count
20000
15000
10000
5000
0
0
1
Donated_embryo

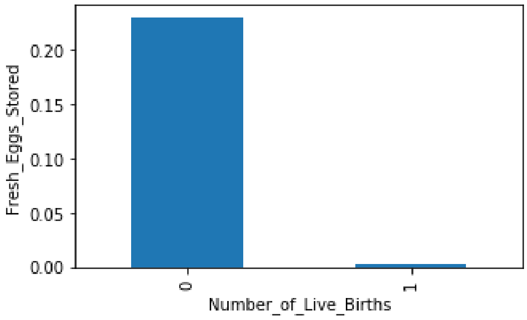
Target Analysis
0
1
14000

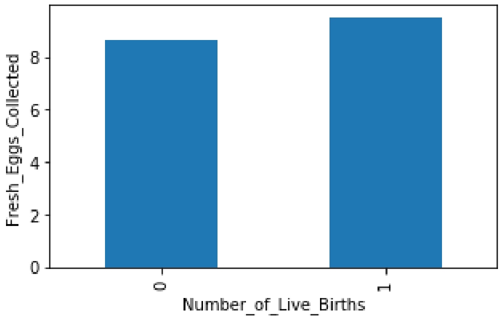


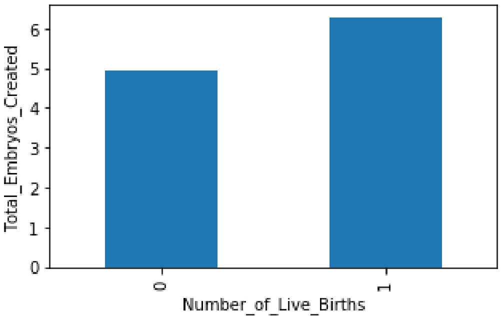
Continous vs Target

In [36]:

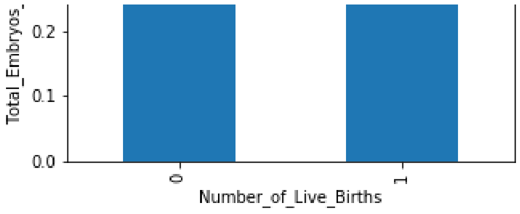
```
for i in num_sel:
    plt.figure(figsize=[5,3])
    df_temp = df.groupby([target])[i].mean()
    df_temp.plot('bar')
    plt.ylabel(i)
    plt.xlabel(target)
    plt.show()
```

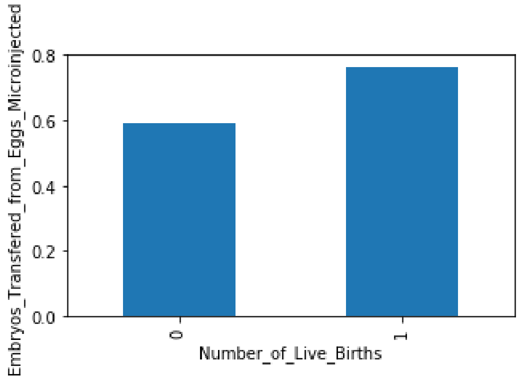


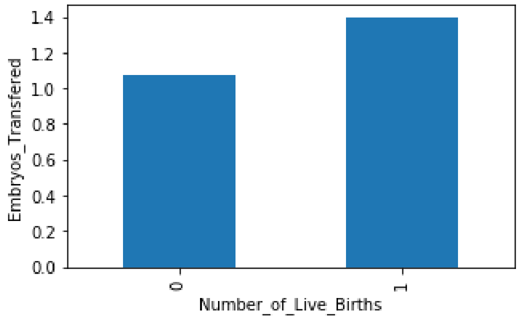


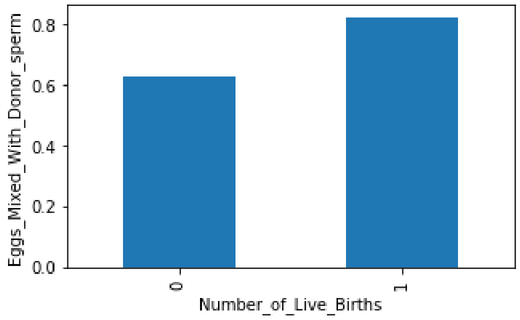


Thawed
0.4
0.3









Applying dummies

In [37]:

```
df_new = df[num_sel+cat_sel+[target]]

df_new = pd.get_dummies(df_new,drop_first = True)
```

Splitting the Data into Train and Test into 70% & 30%

In [38]:

```
X = df_new.drop(target,axis = 1)

Y = df_new[target]

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state = 75)
```

Logistic Regression

In [39]:

```
from sklearn.linear_model import LogisticRegression

log = LogisticRegression().fit(x_train,y_train)

predictions = log.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.70      0.52      0.60      5970
           1       0.61      0.77      0.68      5864

    accuracy                           0.65     11834
   macro avg       0.66      0.65      0.64     11834
weighted avg       0.66      0.65      0.64     11834


[[3122 2848]
 [1333 4531]]
```

DecisionTreeClassifier

In [41]:

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()

tree.fit(x_train,y_train)

predictions = tree.predict(x_test)


predictions = tree.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.63      0.59      0.61      5970
           1       0.61      0.65      0.63      5864

    accuracy                           0.62     11834
   macro avg       0.62      0.62      0.62     11834
weighted avg       0.62      0.62      0.62     11834


[[3525 2445]
 [2028 3836]]
```

RandomForestClassifier

In [42]:

```
from sklearn.ensemble import RandomForestClassifier

tree = RandomForestClassifier()

tree.fit(x_train,y_train)

predictions = tree.predict(x_test)
predictions = tree.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.66      0.56      0.61      5970
           1       0.61      0.71      0.66      5864

    accuracy                           0.63     11834
   macro avg       0.64      0.63      0.63     11834
weighted avg       0.64      0.63      0.63     11834


[[3343 2627]
 [1718 4146]]
```

Xgboost

In [43]:

```
xgcl = xgb.XGBClassifier()

xgcl.fit(x_train, y_train)


predictions = xgcl.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.50      0.61      5970
           1       0.63      0.86      0.73      5864

    accuracy                           0.68     11834
   macro avg       0.71      0.68      0.67     11834
weighted avg       0.71      0.68      0.67     11834


[[3002 2968]
 [ 792 5072]]
```

MultinomialNB

In [44]:

```
from sklearn.naive_bayes import MultinomialNB

Mb_model=MultinomialNB()

Mb_model.fit(x_train,y_train)



predictions = Mb_model.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.69      0.47      0.56      5970
           1       0.59      0.78      0.67      5864

    accuracy                           0.62     11834
```

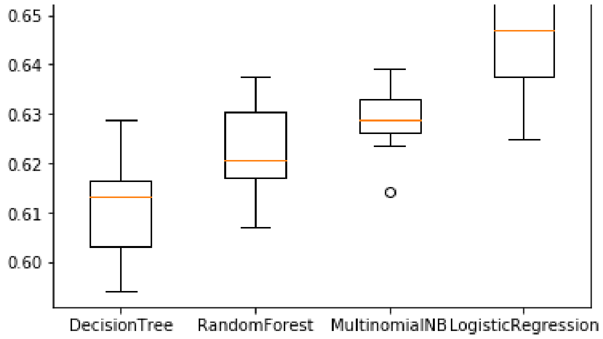
```
   macro avg       0.64      0.63      0.62     11834
weighted avg       0.64      0.62      0.62     11834


[[2813 3157]
 [1287 4577]]
```

Cross_validation

In [28]:

```
from sklearn import model_selection

Dt_model = DecisionTreeClassifier()

Rf_model = RandomForestClassifier()

Mb_model=MultinomialNB()

Lr_model = LogisticRegression()

models = []

models.append(('DecisionTree', Dt_model))

models.append(('RandomForest', Rf_model))

models.append(('MultinomialNB', Mb_model))

models.append(('LogisticRegression', Lr_model))


# evaluate each model in turn

results = []

names = []

scoring = 'accuracy'

for name, model in models:

 kfold = model_selection.KFold(n_splits=10,random_state=2)

 cv_results = model_selection.cross_val_score(model, x_train, y_train, cv=kfold, scoring=scoring)

 results.append(cv_results)

 names.append(name)

 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())

 print(msg)

# boxplot algorithm comparison

fig = plt.figure()

fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
DecisionTree: 0.611365 (0.010288)
RandomForest: 0.622809 (0.009616)
MultinomialNB: 0.628821 (0.006641)
LogisticRegression: 0.645444 (0.010425)
```



Iteration -2 without featureSelection

In [29]:

```
df_new = pd.get_dummies(df,drop_first = True)

X = df_new.drop(target,axis = 1)

Y = df_new[target]

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state = 75)
```

LogisticRegression

In [45]:

```
from sklearn.linear_model import LogisticRegression

log = LogisticRegression().fit(x_train,y_train)

predictions = log.predict(x_test)

print(classification_report(y_test,predictions),'\n')

print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.70      0.52      0.60      5970
           1       0.61      0.77      0.68      5864

    accuracy                           0.65     11834
   macro avg       0.66      0.65      0.64     11834
weighted avg       0.66      0.65      0.64     11834


[[3122 2848]
 [1333 4531]]
```

DecisionTreeClassifier

In [46]:

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()

tree.fit(x_train,y_train)

predictions = tree.predict(x_test)


predictions = tree.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.63      0.59      0.61      5970
           1       0.61      0.65      0.63      5864

    accuracy                           0.62     11834
   macro avg       0.62      0.62      0.62     11834
weighted avg       0.62      0.62      0.62     11834


[[3523 2447]
 [2029 3835]]
```

RandomForestClassifier

In [47]:

```
from sklearn.ensemble import RandomForestClassifier

tree = RandomForestClassifier()

tree.fit(x_train,y_train)

predictions = tree.predict(x_test)


predictions = tree.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.66      0.56      0.60      5970
           1       0.61      0.70      0.65      5864

    accuracy                           0.63     11834
   macro avg       0.63      0.63      0.63     11834
weighted avg       0.63      0.63      0.63     11834


[[3340 2630]
 [1756 4108]]
```

Xgboost

In [48]:

```
xgcl = xgb.XGBClassifier()

xgcl.fit(x_train, y_train)

predictions = tree.predict(x_test)


predictions = xgcl.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.50      0.61      5970
           1       0.63      0.86      0.73      5864

    accuracy                           0.68     11834
   macro avg       0.71      0.68      0.67     11834
```

```
weighted avg       0.71      0.68      0.67     11834


[[3002 2968]
 [ 792 5072]]
```

MultinomialNB

In [49]:

```
from sklearn.naive_bayes import MultinomialNB

Mb_model=MultinomialNB()

Mb_model.fit(x_train,y_train)



predictions = Mb_model.predict(x_test)

print(classification_report(y_test,predictions),'\n')
print(confusion_matrix(y_test,predictions))
```

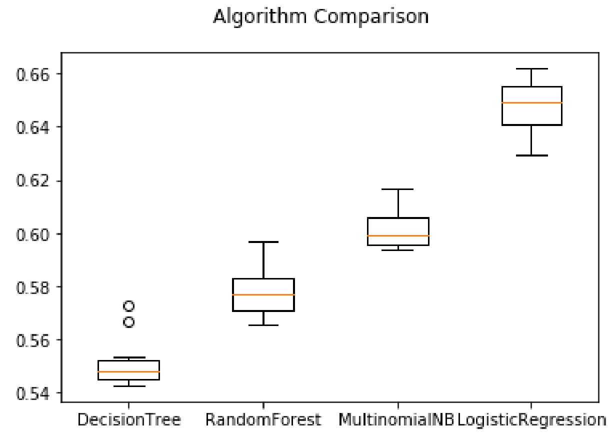
```
              precision    recall  f1-score   support

           0       0.69      0.47      0.56      5970
           1       0.59      0.78      0.67      5864

    accuracy                           0.62     11834
   macro avg       0.64      0.63      0.62     11834
weighted avg       0.64      0.62      0.62     11834


[[2813 3157]
 [1287 4577]]
```

Cross_validation

In [35]:

```
from sklearn import model_selection

Dt_model = DecisionTreeClassifier()

Rf_model = RandomForestClassifier()

Mb_model=MultinomialNB()

Lr_model = LogisticRegression()

models = []

models.append(('DecisionTree', Dt_model))

models.append(('RandomForest', Rf_model))

models.append(('MultinomialNB', Mb_model))

models.append(('LogisticRegression', Lr_model))


# evaluate each model in turn

results = []

names = []

scoring = 'accuracy'

for name, model in models:
 kfold = model_selection.KFold(n_splits=10,random_state=2)

 cv_results = model_selection.cross_val_score(model, x_train, y_train, cv=kfold, scoring=scoring)

 results.append(cv_results)

 names.append(name)

 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())

 print(msg)

# boxplot algorithm comparison

fig = plt.figure()

fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
DecisionTree: 0.551716 (0.009734)
RandomForest: 0.578082 (0.008937)
MultinomialNB: 0.601985 (0.007848)
LogisticRegression: 0.647545 (0.010042)
```



```
In [ ]:
```