In [1]:

```
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 [2]:

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

In [3]:

```
df.shape
```

Out[3]:

```
(80488, 26)
```

In [4]:

```
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

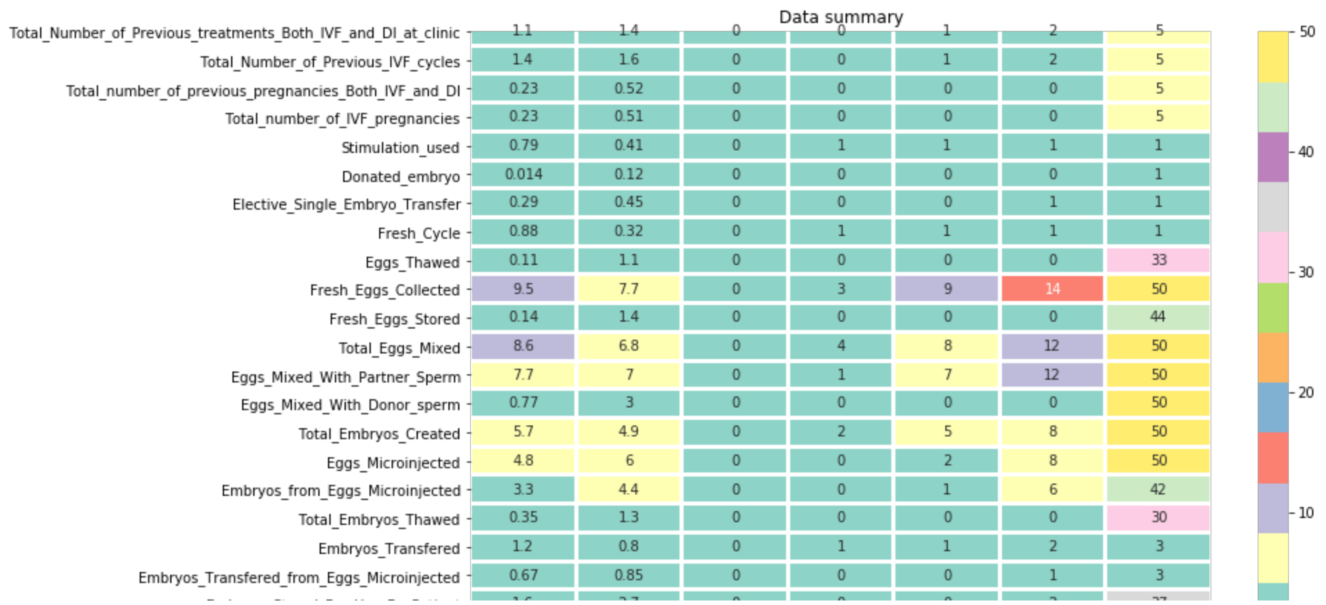
In [5]:

```
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
```

In [6]:

```
plt.figure(figsize=(12,8))
sns.heatmap(df.describe()[1:].transpose(),
            annot=True,linecolor="w",
            linewidth=2,cmap=sns.color_palette("Set3"))
plt.title("Data summary")
plt.show()
```



Target Fitness

In [7]:

```
# 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 [8]:

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

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

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

# df.reset_index(inplace = True)

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

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

# df_two = df[df[target]==2].head(low)

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

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

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


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

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 76322 entries, 0 to 79620
Data columns (total 26 columns):
Patient_Age_at_Treatment                                       76322 non-null object
Total_Number_of_Previous_treatments_Both_IVF_and_DI_at_clinic  76322 non-null int64
Total_Number_of_Previous_IVF_cycles                            76322 non-null int64
Total_number_of_previous_pregnancies_Both_IVF_and_DI           76322 non-null int64
Total_number_of_IVF_pregnancies                                76322 non-null int64
Stimulation_used                                               76322 non-null int64
Donated_embryo                                                 76322 non-null float64
Specific_treatment_type                                        76322 non-null object
Elective_Single_Embryo_Transfer                                76322 non-null float64
Egg_Source                                                     76322 non-null object
Sperm_From                                                     76322 non-null object
Fresh_Cycle                                                    76322 non-null float64
Eggs_Thawed                                                    76322 non-null float64
Fresh_Eggs_Collected                                           76322 non-null float64
```

```
Fresh_Eggs_Stored                                   76322 non-null float64
Total_Eggs_Mixed                                    76322 non-null float64
Eggs_Mixed_With_Partner_Sperm                       76322 non-null float64
Eggs_Mixed_With_Donor_sperm                         76322 non-null float64
Total_Embryos_Created                               76322 non-null float64
Eggs_Microinjected                                  76322 non-null float64
Embryos_from_Eggs_Microinjected                     76322 non-null float64
Total_Embryos_Thawed                                76322 non-null float64
Embryos_Transfered                                  76322 non-null float64
Embryos_Transfered_from_Eggs_Microinjected          76322 non-null float64
Embryos_Stored_For_Use_By_Patient                   76322 non-null float64
Number_of_Live_Births                               76322 non-null int64
dtypes: float64(16), int64(6), object(4)
memory usage: 15.7+ MB
```

In [10]:

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

Out[10]:

```
0    56599
1    19723
Name: Number_of_Live_Births, dtype: int64
```

In [11]:

```
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

    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 [12]:

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

Datatype Conversion

In [13]:

```
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[13]:

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

Statistical Analysis

1.Chi2 Analysis

In [14]:

```
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[t
arget]))

        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("*"*25,"selected columns","*"*25,'\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': 0.0, 'Donated_embryo': 0.0, 'Elective_Single_Embryo_Transfer': 0.0, 'Fresh_Cycle': 0.0}

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

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

2.Anova Analysis

In [15]:

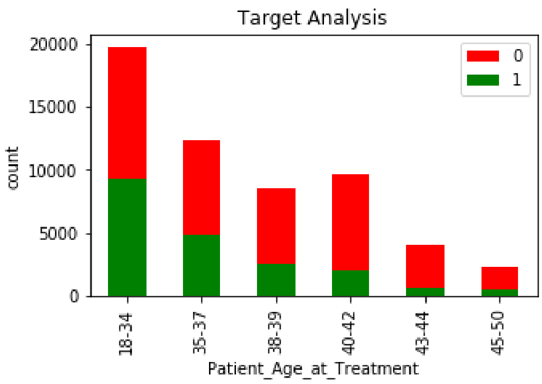
```
def anova(cat_col,target,Dataframe):

    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("*"*45,"selected columns","*"*45,'\n')

print(num_sel)
```

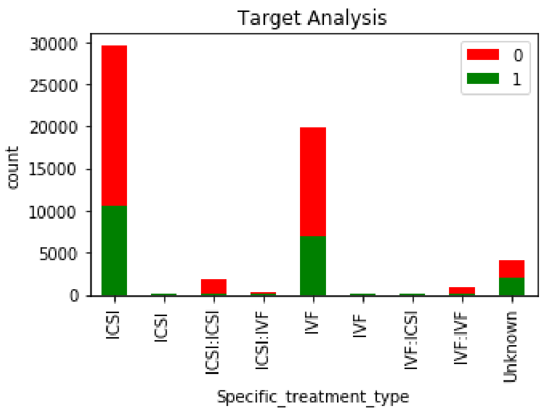
```
{'Eggs_Mixed_With_Donor_sperm': 0.205, 'Embryos_from_Eggs_Microinjected': 0.025, 'Eggs_Mixed_With_Partner_Sperm': 0.046, 'Total_Eggs_Mixed': 0.048, 'Total_Embryos_Created': 0.0, 'Embryos_Transfered_from_Eggs_Microinjected': 0.0, 'Embryos_Stored_For_Use_By_Patient': 0.058, 'Fresh_Eggs_Stored': 0.002, 'Eggs_Microinjected': 0.307, 'Embryos_Transfered': 0.0, 'Eggs_Thawed': 0.0, 'Total_Embryos_Thawed': 0.0, 'Fresh_Eggs_Collected': 0.0}

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

['Embryos_from_Eggs_Microinjected', 'Eggs_Mixed_With_Partner_Sperm', 'Total_Eggs_Mixed', 'Total_Embryos_Created', 'Embryos_Transfered_from_Eggs_Microinjected', 'Fresh_Eggs_Stored', 'Embryos_Transfered', 'Eggs_Thawed', 'Total_Embryos_Thawed', 'Fresh_Eggs_Collected']
```

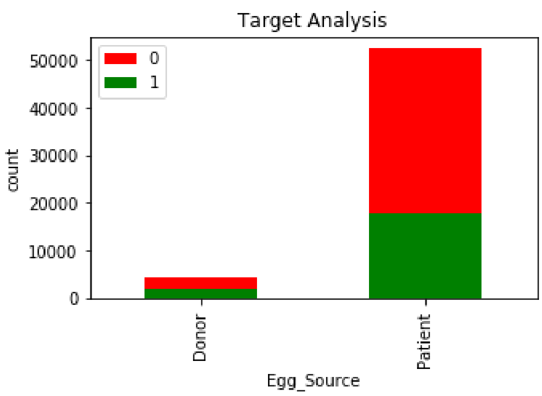
Catogorical column vs Target

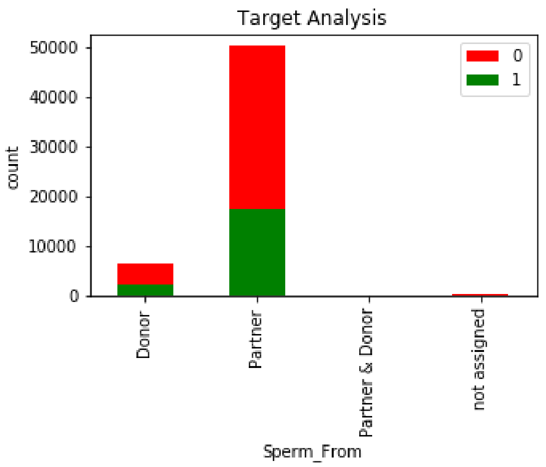
In [16]:

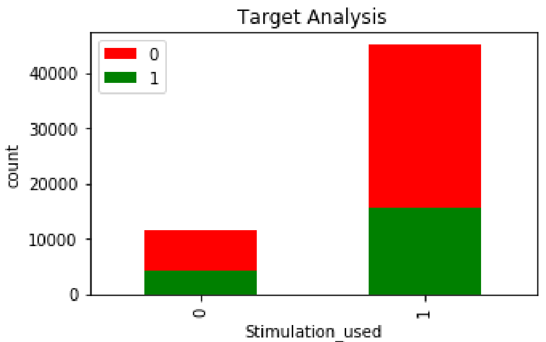
```
def cat_visual(col,target,df):

    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)
```

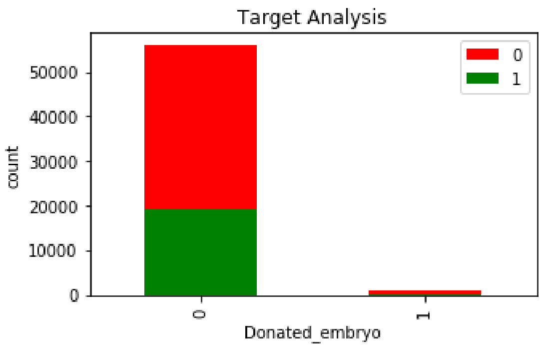


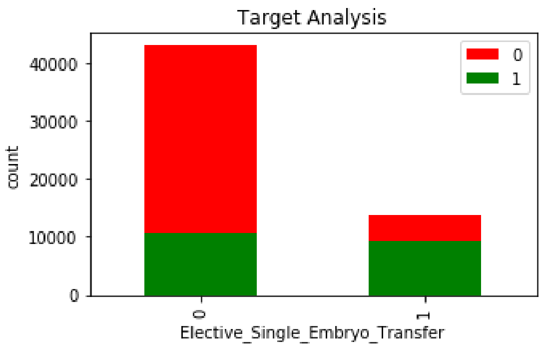


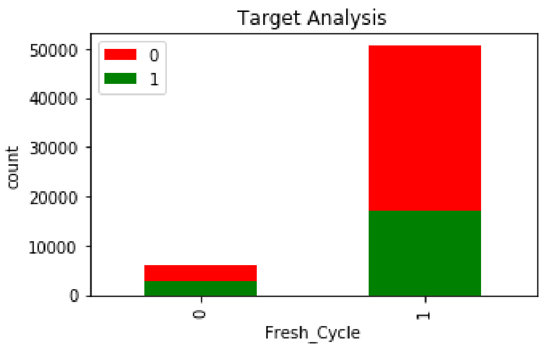



Target Analysis
count
0
1
Donor
Partner
Partner & Donor
not assigned
Sperm_From


Target Analysis
count
0
1
Stimulation_used


Target Analysis
count
0
1
Donated_embryo

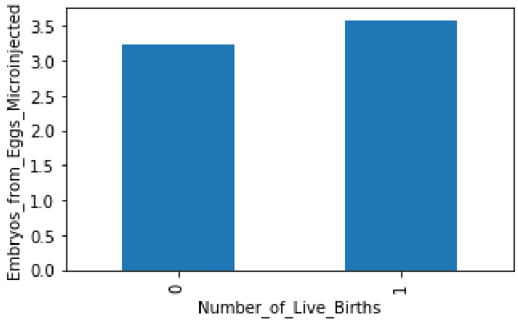

Target Analysis
count
0
1
Elective_Single_Embryo_Transfer

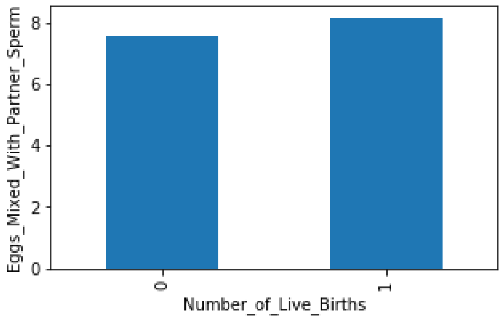

Target Analysis
count
0
1
Fresh_Cycle

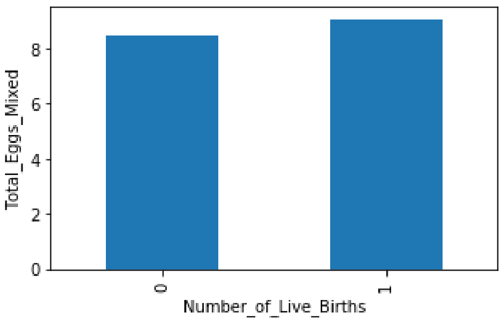
Continous vs Target

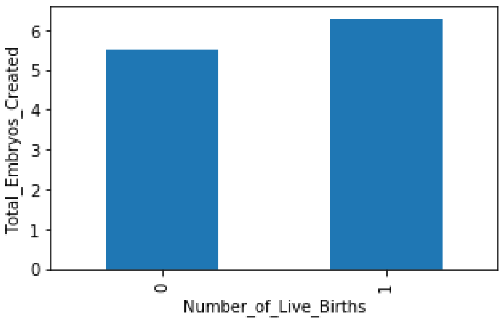
In [45]:

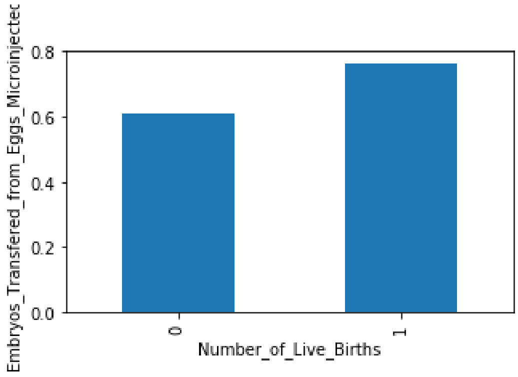
```
for i in num_sel:

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

    df_temp = df.groupby([target])[i].mean()

    df_temp.plot('bar')

    plt.ylabel(i)

    plt.xlabel(target)

    plt.show()
```

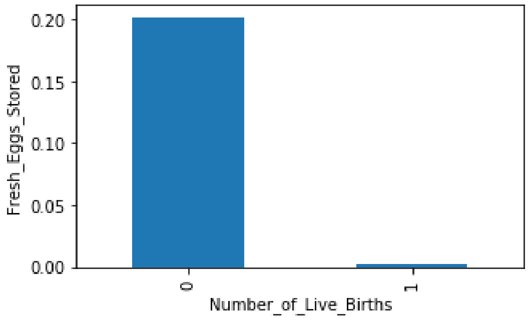


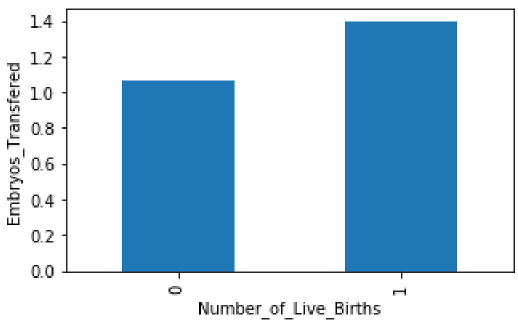


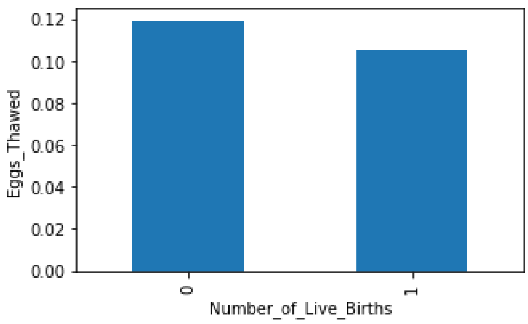


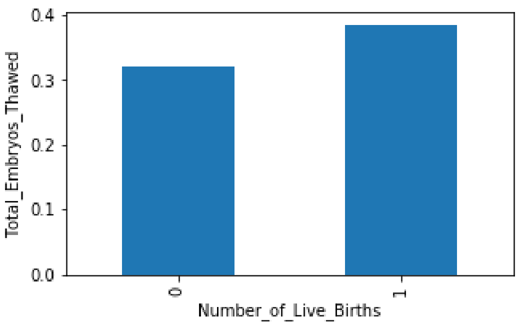


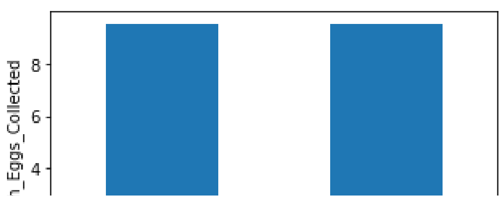

Embryos_Transfered_from_Eggs_Microinjected
0.0
0.2
0.4
0.6
0.8
0
1
Number_of_Live_Births

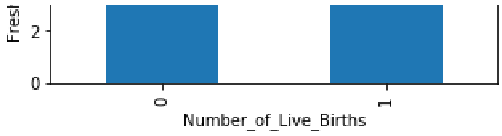

Fresh_Eggs_Stored
0.00
0.05
0.10
0.15
0.20
0
1
Number_of_Live_Births


Embryos_Transfered
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1
Number_of_Live_Births


Eggs_Thawed
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
1
Number_of_Live_Births


Total_Embryos_Thawed
0.0
0.1
0.2
0.3
0.4
0
1
Number_of_Live_Births


4
6
8



In [18]:

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

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

In [19]:

```
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 [21]:

```
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')
```

```
              precision    recall  f1-score   support

           0       0.74      0.98      0.84     16929
           1       0.40      0.04      0.08      5968

    accuracy                           0.73     22897
   macro avg       0.57      0.51      0.46     22897
weighted avg       0.65      0.73      0.64     22897
```

In [23]:

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

```
[[16534   395]
 [ 5704   264]]
```

DecisionTreeClassifier

In [24]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.75      0.87      0.81     16929
           1       0.35      0.20      0.25      5968

    accuracy                           0.69     22897
   macro avg       0.55      0.53      0.53     22897
weighted avg       0.65      0.69      0.66     22897


[[14710  2219]
 [ 4792  1176]]
```

In [46]:

```
from sklearn.model_selection import GridSearchCV

params={'max_depth':np.arange(1,30)}

DT=DecisionTreeClassifier()

GS=GridSearchCV(DT,params,cv=5)

GS.fit(x_train,y_train)

k=GS.best_params_

k=k['max_depth']

print(' best max_depth value: ',k)

from sklearn.tree import DecisionTreeClassifier

model=DecisionTreeClassifier(criterion='entropy',max_depth=k)

model.fit(x_train,y_train)

print('--------------------------------test_data------------------------------')

from sklearn import metrics

import numpy as np


predictions = tree.predict(x_test)

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

```
 best max_depth value:  1
--------------------------------test_data------------------------------
              precision    recall  f1-score   support

           0       0.74      0.82      0.78     16929
           1       0.26      0.18      0.21      5968

    accuracy                           0.65     22897
   macro avg       0.50      0.50      0.49     22897
weighted avg       0.61      0.65      0.63     22897


[[13837  3092]
 [ 4903  1065]]
```

RandomForestClassifier

In [25]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.85      0.80     16929
           1       0.35      0.23      0.27      5968

    accuracy                           0.69     22897
   macro avg       0.55      0.54      0.54     22897
weighted avg       0.65      0.69      0.67     22897


[[14449  2480]
 [ 4624  1344]]
```

Xgboost

In [27]:

```
xgcl = xgb.XGBClassifier()

xgcl.fit(x_train, y_train)


predictions = xgcl.predict(x_test)

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

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85     16929
           1       0.44      0.01      0.02      5968

    accuracy                           0.74     22897
   macro avg       0.59      0.50      0.43     22897
weighted avg       0.66      0.74      0.63     22897


[[16863    66]
 [ 5917    51]]
```

MultinomialNB

In [28]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support
```

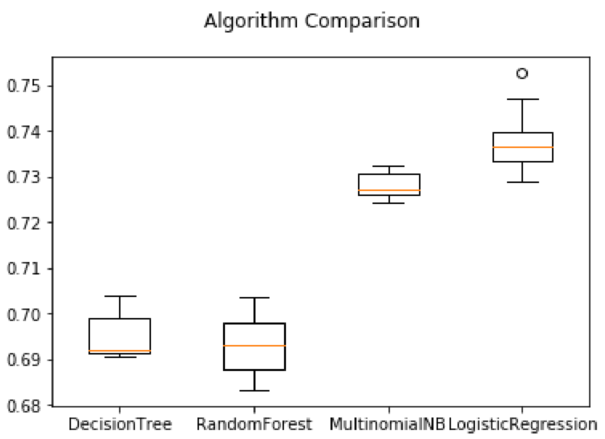
```
             0       0.75      0.94      0.83     16929
             1       0.39      0.11      0.17      5968

      accuracy                           0.72     22897
     macro avg       0.57      0.52      0.50     22897
  weighted avg       0.66      0.72      0.66     22897


[[15900  1029]
 [ 5314   654]]
```

Cross_validation

In [30]:

```
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: 0.694843 (0.004646)
RandomForest: 0.693140 (0.006478)
MultinomialNB: 0.728049 (0.002835)
LogisticRegression: 0.738081 (0.006857)
```



Iteration -2 without featureSelection

In [31]:

```
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 [33]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.75      0.97      0.84     16929
           1       0.42      0.07      0.11      5968

    accuracy                           0.73     22897
   macro avg       0.58      0.52      0.48     22897
weighted avg       0.66      0.73      0.65     22897


[[16396   533]
 [ 5577   391]]
```

DecisionTreeClassifier

In [34]:

```
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')
```

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74     16929
           1       0.27      0.27      0.27      5968

    accuracy                           0.62     22897
   macro avg       0.51      0.51      0.51     22897
weighted avg       0.62      0.62      0.62     22897
```

RandomForestClassifier

In [36]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.74      0.82      0.78     16929
           1       0.26      0.18      0.21      5968

    accuracy                           0.65     22897
   macro avg       0.50      0.50      0.49     22897
weighted avg       0.61      0.65      0.63     22897


[[13837  3092]
 [ 4903  1065]]
```

Xgboost

In [37]:

```
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')
cmat=print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.74      0.98      0.85     16929
           1       0.46      0.04      0.07      5968

    accuracy                           0.74     22897
```

```
   macro avg       0.60      0.51      0.46     22897
weighted avg       0.67      0.74      0.64     22897


[[16674   255]
 [ 5748   220]]
```

MultinomialNB

In [38]:

```
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')
```

```
              precision    recall  f1-score   support

           0       0.77      0.84      0.80     16929
           1       0.39      0.30      0.34      5968

    accuracy                           0.70     22897
   macro avg       0.58      0.57      0.57     22897
weighted avg       0.67      0.70      0.68     22897
```

Cross_validation

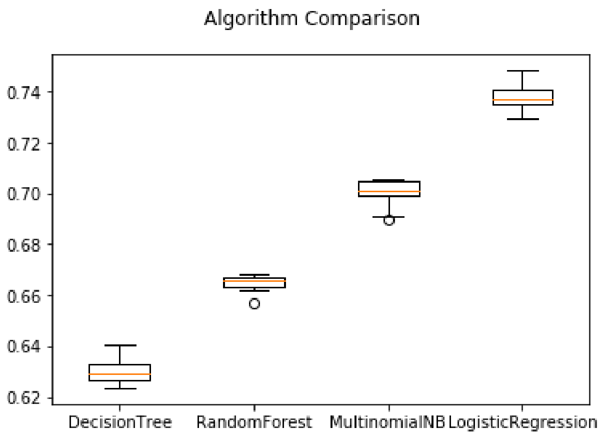
That k-fold cross validation is a procedure used to estimate the skill of the model on new data. There are common tactics that you can use to select the value of k for your dataset. There are commonly used variations on cross-validation such as stratified and repeated that are available in scikit-learn.

In [39]:

```
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: 0.630435 (0.004975)
RandomForest: 0.664670 (0.003328)
MultinomialNB: 0.700215 (0.005427)
LogisticRegression: 0.737876 (0.005350)
```



In [ ]: