Importing Libraries

In [67]:

```
# linear algebra and data preprocessing
import numpy as np
import pandas as pd

# plotting and graphs
import seaborn as sns
import matplotlib.pyplot as plt
```

In [69]:

```
# importing dataset

path = 'c:/Users/ADMIN/Documents/audit_data/'
df = pd.read_csv(path + 'trial.csv')
```

In [70]:

```
df.head()
```

Out[70]:

	Sector_score	LOCATION_ID	PARA_A	SCORE_A	PARA_B	SCORE_B	TOTAL	numbers	Marks	Money_Value	MONEY_Ma
0	3.89	23	4.18	6	2.50	2	6.68	5.0	2	3.38	
1	3.89	6	0.00	2	4.83	2	4.83	5.0	2	0.94	
2	3.89	6	0.51	2	0.23	2	0.74	5.0	2	0.00	
3	3.89	6	0.00	2	10.80	6	10.80	6.0	6	11.75	
4	3.89	6	0.00	2	0.08	2	0.08	5.0	2	0.00	

In [71]:

```
print(df.shape)
```

```
(776, 18)
```

In [72]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 776 entries, 0 to 775
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sector_score    776 non-null    float64
 1   LOCATION_ID     776 non-null    object
 2   PARA_A          776 non-null    float64
 3   SCORE_A         776 non-null    int64
 4   PARA_B          776 non-null    float64
 5   SCORE_B         776 non-null    int64
 6   TOTAL           776 non-null    float64
 7   numbers         776 non-null    float64
 8   Marks           776 non-null    int64
 9   Money_Value     775 non-null    float64
 10  MONEY_Marks     776 non-null    int64
 11  District        776 non-null    int64
 12  Loss            776 non-null    int64
 13  LOSS_SCORE      776 non-null    int64
 14  History         776 non-null    int64
```

```
 15   History_score   776 non-null    int64
 16   Score           776 non-null    float64
 17   Risk            776 non-null    int64
dtypes: float64(7), int64(10), object(1)
memory usage: 109.2+ KB
```

In [73]:

```
# displaying number of null values
df.apply(lambda x: sum(x.isnull()))
```

Out[73]:

```
Sector_score     0
LOCATION_ID      0
PARA_A           0
SCORE_A          0
PARA_B           0
SCORE_B          0
TOTAL            0
numbers          0
Marks            0
Money_Value      1
MONEY_Marks      0
District         0
Loss             0
LOSS_SCORE       0
History          0
History_score    0
Score            0
Risk             0
dtype: int64
```

In [74]:

```
## Finding unique values

df.apply(lambda x: len(x.unique()))
```

Out[74]:

```
Sector_score      13
LOCATION_ID       45
PARA_A           363
SCORE_A            3
PARA_B           358
SCORE_B            3
TOTAL            471
numbers            5
Marks              3
Money_Value      329
MONEY_Marks        3
District           3
Loss               3
LOSS_SCORE         3
History            7
History_score      3
Score             17
Risk               2
dtype: int64
```

In [183]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 776 entries, 0 to 775
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sector_score    776 non-null    float64
 1   LOCATION_ID     776 non-null    float64
 2   PARA_A          776 non-null    float64
```

```
 3   SCORE_A         776 non-null    int64  
 4   PARA_B          776 non-null    float64
 5   SCORE_B         776 non-null    int64  
 6   TOTAL           776 non-null    float64
 7   numbers         776 non-null    float64
 8   Marks           776 non-null    int64  
 9   Money_Value     776 non-null    float64
 10  MONEY_Marks     776 non-null    int64  
 11  District        776 non-null    int64  
 12  Loss            776 non-null    int64  
 13  LOSS_SCORE      776 non-null    int64  
 14  History         776 non-null    int64  
 15  History_score   776 non-null    int64  
 16  Score           776 non-null    float64
 17  Risk            776 non-null    int64  
dtypes: float64(8), int64(10)
memory usage: 109.2 KB
```

In [187]:

```python
# Converting LOCATION_ID object type  into float datatype
df['LOCATION_ID'] = df['LOCATION_ID'].astype(np.float)

# filling nan
df['Money_Value'].fillna(df['Money_Value'].mean(),inplace = True)
df['LOCATION_ID'].fillna(df['LOCATION_ID'].mean(),inplace = True)
```

In [188]:

```python
## checking for nulls

df.apply(lambda x: sum(x.isnull()))
```

Out[188]:

```
Sector_score     0
LOCATION_ID      0
PARA_A           0
SCORE_A          0
PARA_B           0
SCORE_B          0
TOTAL            0
numbers          0
Marks            0
Money_Value      0
MONEY_Marks      0
District         0
Loss             0
LOSS_SCORE       0
History          0
History_score    0
Score            0
Risk             0
dtype: int64
```

In [78]:

```python
df.describe()
```

Out[78]:

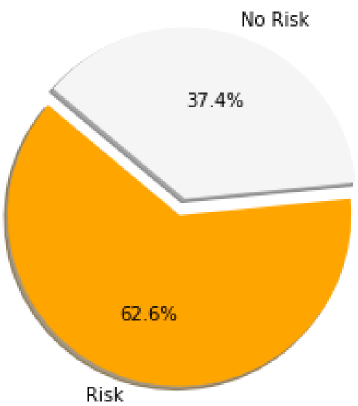
	Sector_score	LOCATION_ID	PARA_A	SCORE_A	PARA_B	SCORE_B	TOTAL	numbers	Marks	M
count	776.000000	776.000000	776.000000	776.000000	776.000000	776.000000	776.000000	776.000000	776.000000	
mean	20.184536	14.856404	2.450194	3.512887	10.799988	3.131443	13.218481	5.067655	2.237113	
std	24.319017	9.872154	5.678870	1.740549	50.083624	1.698042	51.312829	0.264449	0.803517	
min	1.850000	1.000000	0.000000	2.000000	0.000000	2.000000	0.000000	5.000000	2.000000	
25%	2.370000	8.000000	0.210000	2.000000	0.000000	2.000000	0.537500	5.000000	2.000000	
50%	3.890000	13.000000	0.875000	2.000000	0.405000	2.000000	1.370000	5.000000	2.000000	

	Sector_score	LOCATION_ID	PARA_A	SCORE_A	PARA_B	SCORE_B	TOTAL	numbers	Marks	M
75%	55.570000	19.000000	2.480000	6.000000	4.160000	4.000000	7.707500	5.000000	2.000000	
max	59.850000	44.000000	85.000000	6.000000	1264.630000	6.000000	1268.910000	9.000000	6.000000	

EDA

In [79]:

```
## Total Risk
# Data to plot
labels =["Risk","No Risk"]
sizes = df['Risk'].value_counts(sort = "1")
colors = ["orange","whitesmoke"]
explode = (0.1,0)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140,)
plt.axis('equal')
plt.show()
```

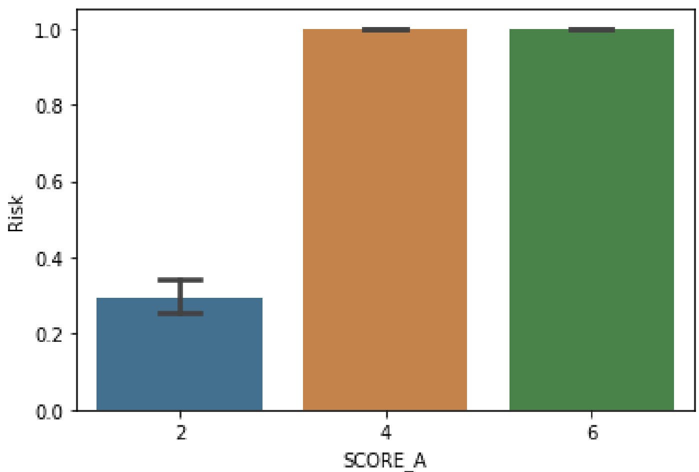


In [80]:

```
fig_dims = (6, 4)

## Barplot for Score_A and Risk

fig, ax = plt.subplots(figsize=fig_dims)
sns.barplot(x=df.SCORE_A, y=df.Risk, capsize = 0.2, saturation=.5)
```

Out[80]:

```
<AxesSubplot:xlabel='SCORE_A', ylabel='Risk'>
```



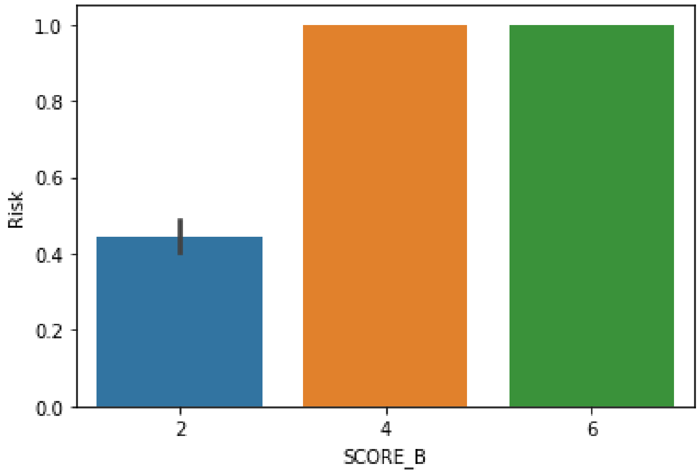
In [81]:

```
## Barplot for Score_B and Risk
```

```
fig, ax = plt.subplots(figsize=fig_dims)
sns.barplot(x=df['SCORE_B'], y=df['Risk'])
```

Out[81]:

```
<AxesSubplot:xlabel='SCORE_B', ylabel='Risk'>
```

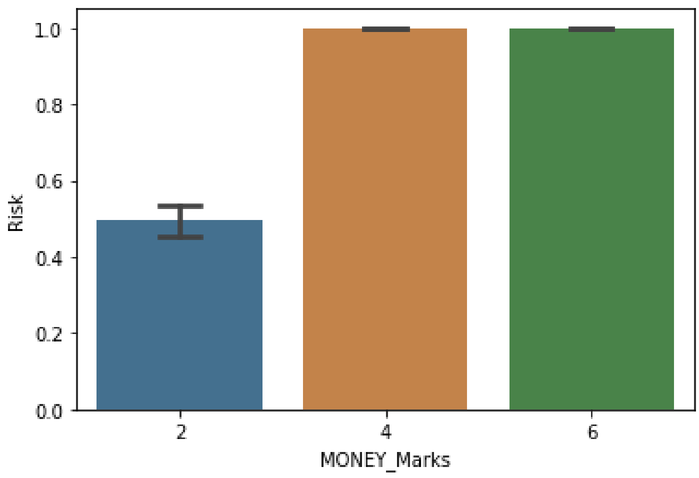


In [82]:

```
## Barplot for Money marks and Risk

fig, ax = plt.subplots(figsize=fig_dims)
sns.barplot(x=df.MONEY_Marks, y=df.Risk, capsize = 0.2, saturation=.5)
```

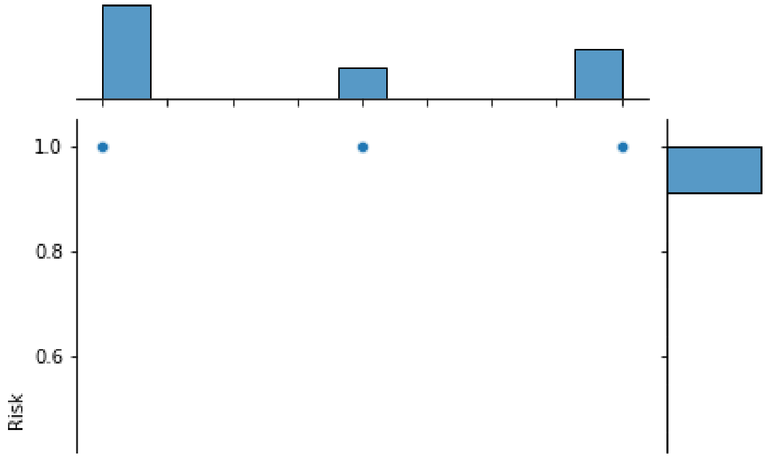
Out[82]:

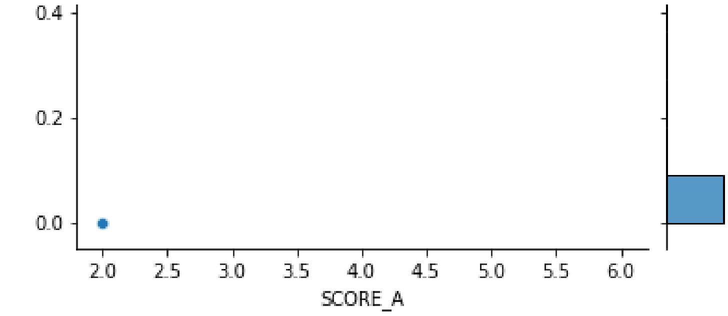
```
<AxesSubplot:xlabel='MONEY_Marks', ylabel='Risk'>
```



In [83]:

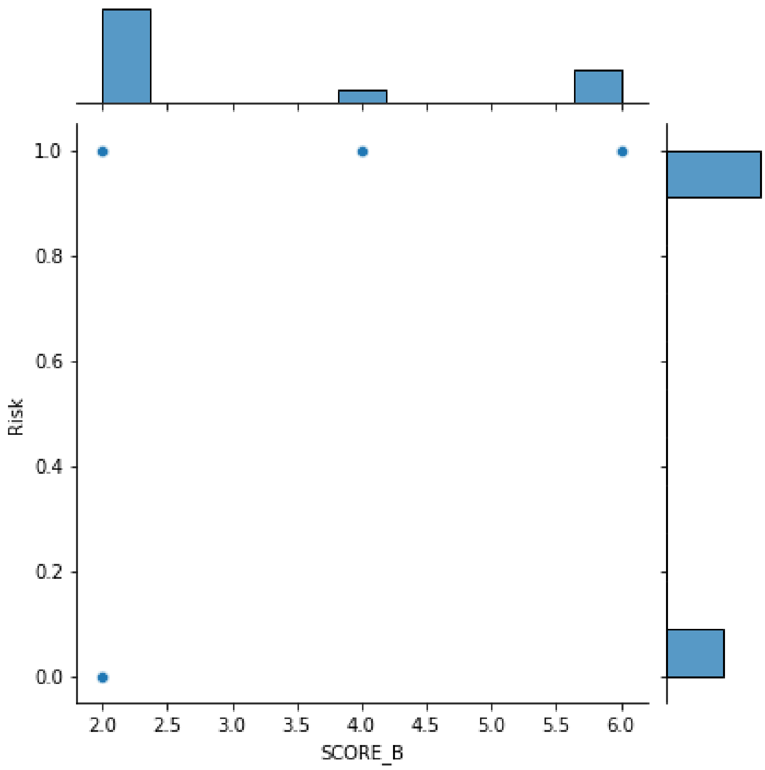
```
## SCORE_A and Risk

sns.jointplot(x=df['SCORE_A'], y=df['Risk']);
```





In [84]:

```
## SCORE_B and Risk

sns.jointplot(x=df['SCORE_B'], y=df['Risk']);
```

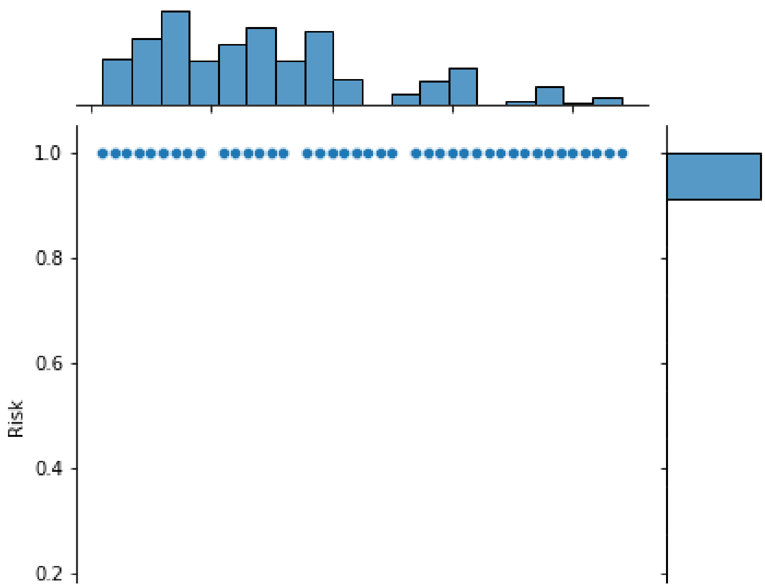


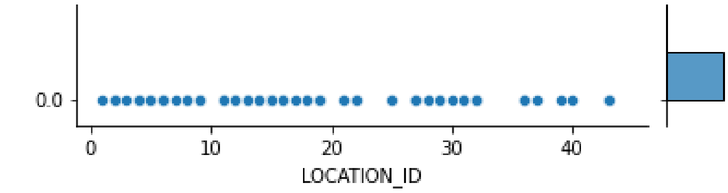
In [85]:

```
## LOCATION_ID and Risk

sns.jointplot(x=df['LOCATION_ID'], y=df['Risk'])
```

Out[85]:

```
<seaborn.axisgrid.JointGrid at 0x1fa34dc2070>
```



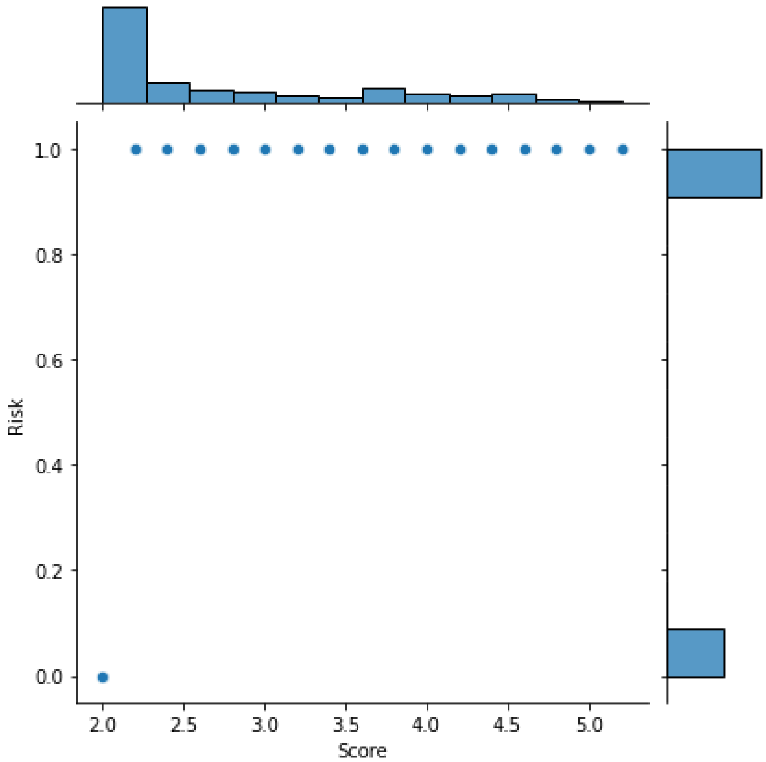


In [86]:

```
## Score and Risk

sns.jointplot(x=df['Score'], y=df['Risk'])
```

Out[86]:

<seaborn.axisgrid.JointGrid at 0x1fa35dc8850>

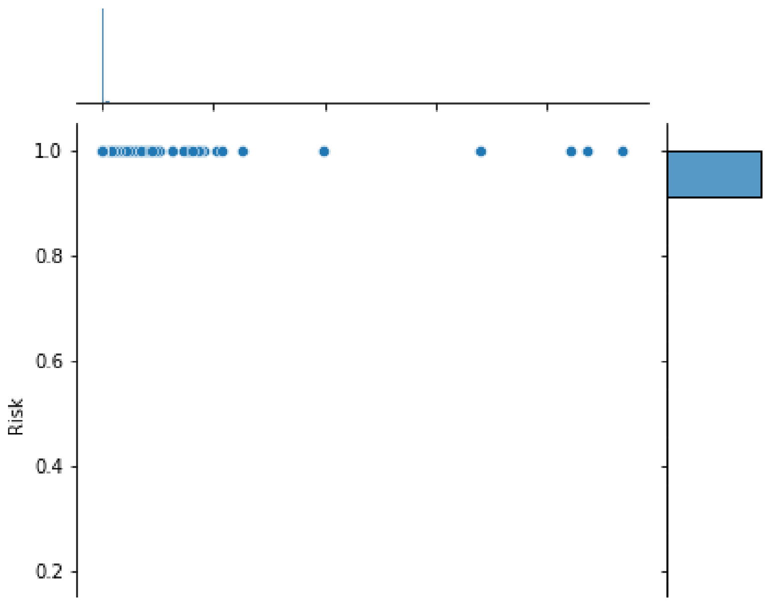


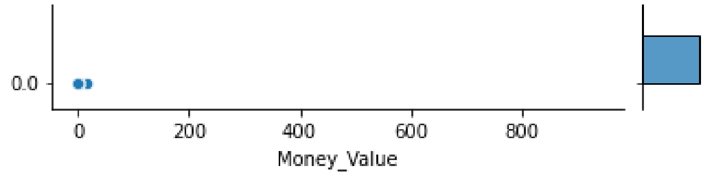
In [87]:

```
## Analyzing Money_value against Risk

sns.jointplot(x=df['Money_Value'], y=df['Risk'])
```

Out[87]:

<seaborn.axisgrid.JointGrid at 0x1fa37400190>



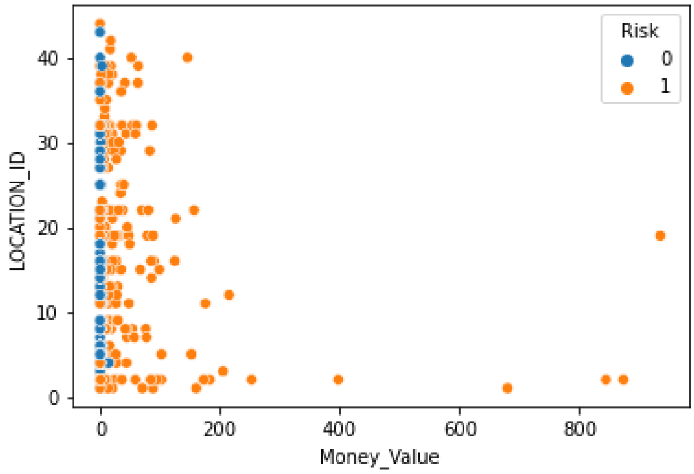


In [88]:

```
fig, ax = plt.subplots(figsize=fig_dims)
sns.scatterplot(x=df['Money_Value'], y=df['LOCATION_ID'], hue = df.Risk)
```

Out[88]:

<AxesSubplot:xlabel='Money_Value', ylabel='LOCATION_ID'>

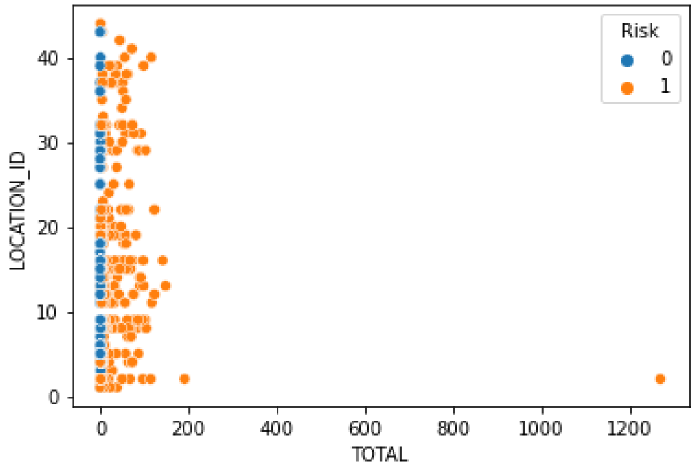


In [89]:

```
fig, ax = plt.subplots(figsize=fig_dims)
sns.scatterplot(x=df['TOTAL'], y=df['LOCATION_ID'], hue = df.Risk)
```

Out[89]:

<AxesSubplot:xlabel='TOTAL', ylabel='LOCATION_ID'>

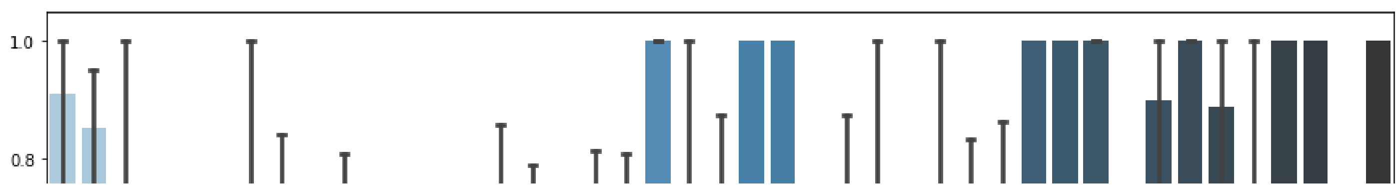


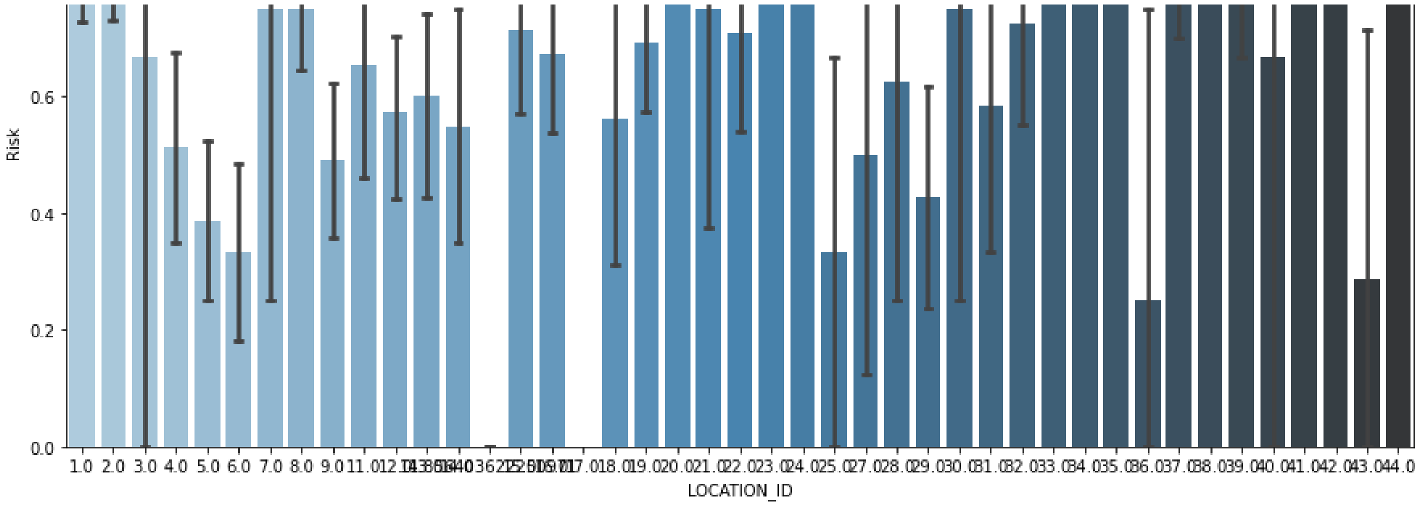
In [90]:

```
fig_dims = (15, 7)
fig, ax = plt.subplots(figsize=fig_dims)
sns.barplot(x=df.LOCATION_ID, y=df.Risk, capsize = 0.2, palette="Blues_d")
```

Out[90]:

<AxesSubplot:xlabel='LOCATION_ID', ylabel='Risk'>

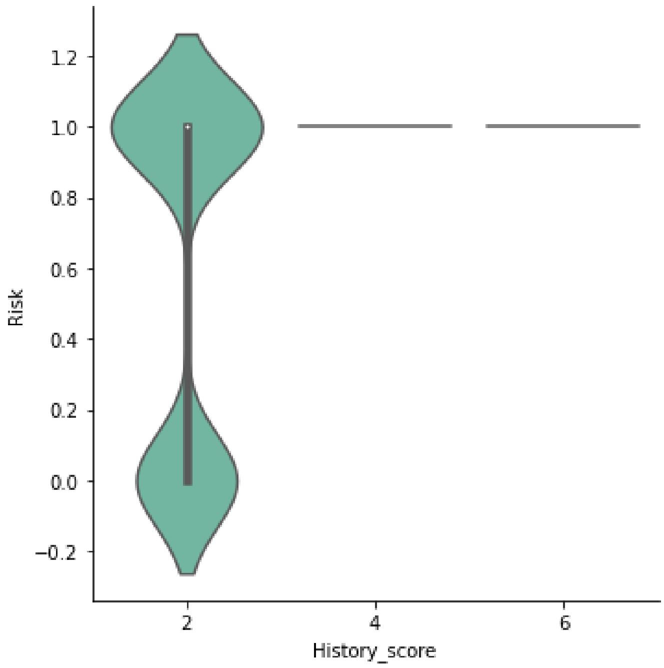




In [91]:

```
ax = sns.factorplot(y="Risk",x="History_score",data=df,kind="violin", palette = "Set2")
```

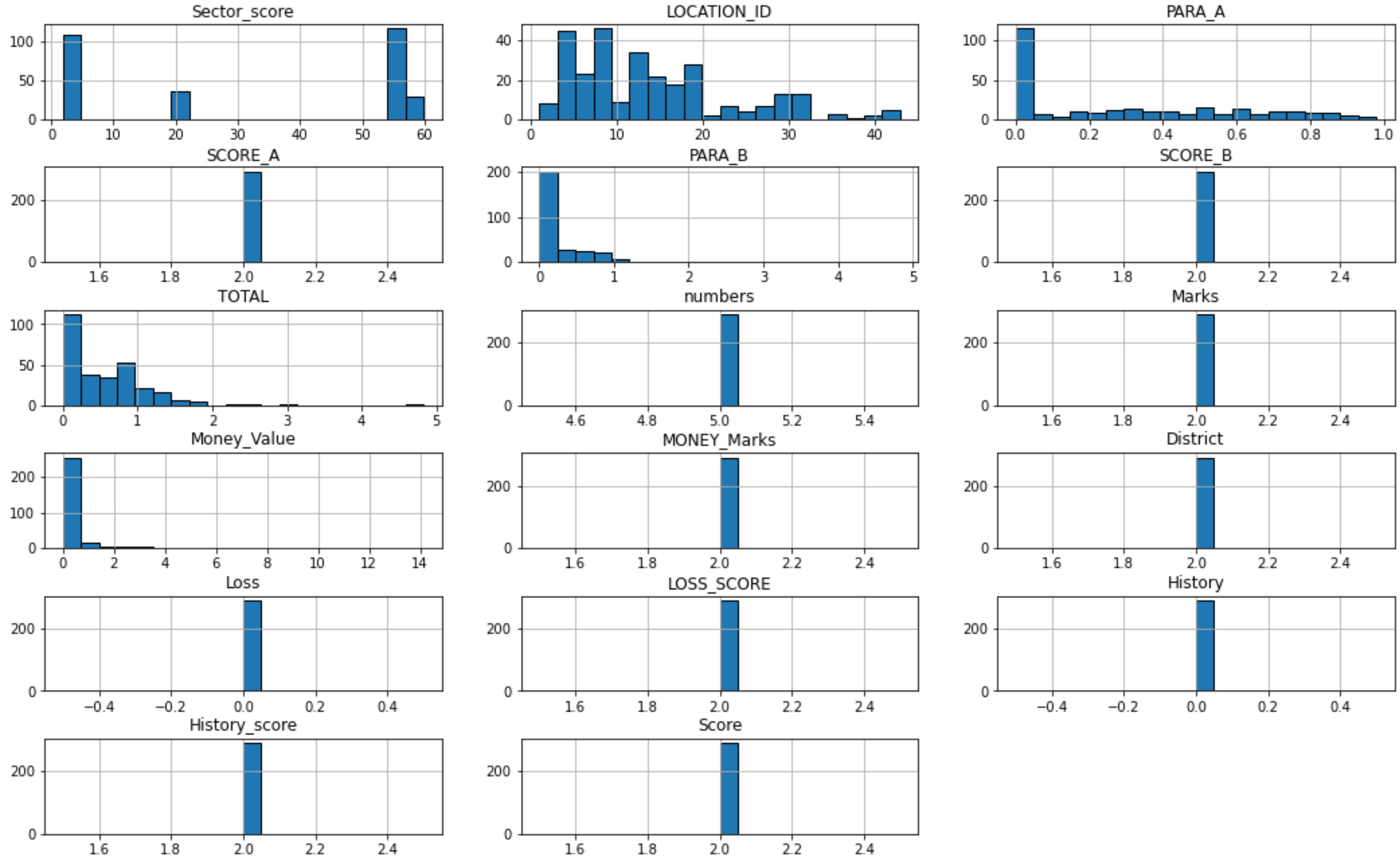
```
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\categorical.py:3704: UserWarning: The
`factorplot` function has been renamed to `catplot`. The original name will be removed in
a future release. Please update your code. Note that the default `kind` in `factorplot` (
`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```



In [92]:

```
### All plots for Risk = 0

import itertools

df1=df[df['Risk']==0]
columns=df1.columns[:17]
plt.subplots(figsize=(18,15))
length=len(columns)
for i,j in itertools.zip_longest(columns,range(length)):
    plt.subplot((length/2),3,j+1)
    plt.subplots_adjust(wspace=0.2,hspace=0.5)
    df1[i].hist(bins=20,edgecolor='black')
    plt.title(i)
plt.show()
```

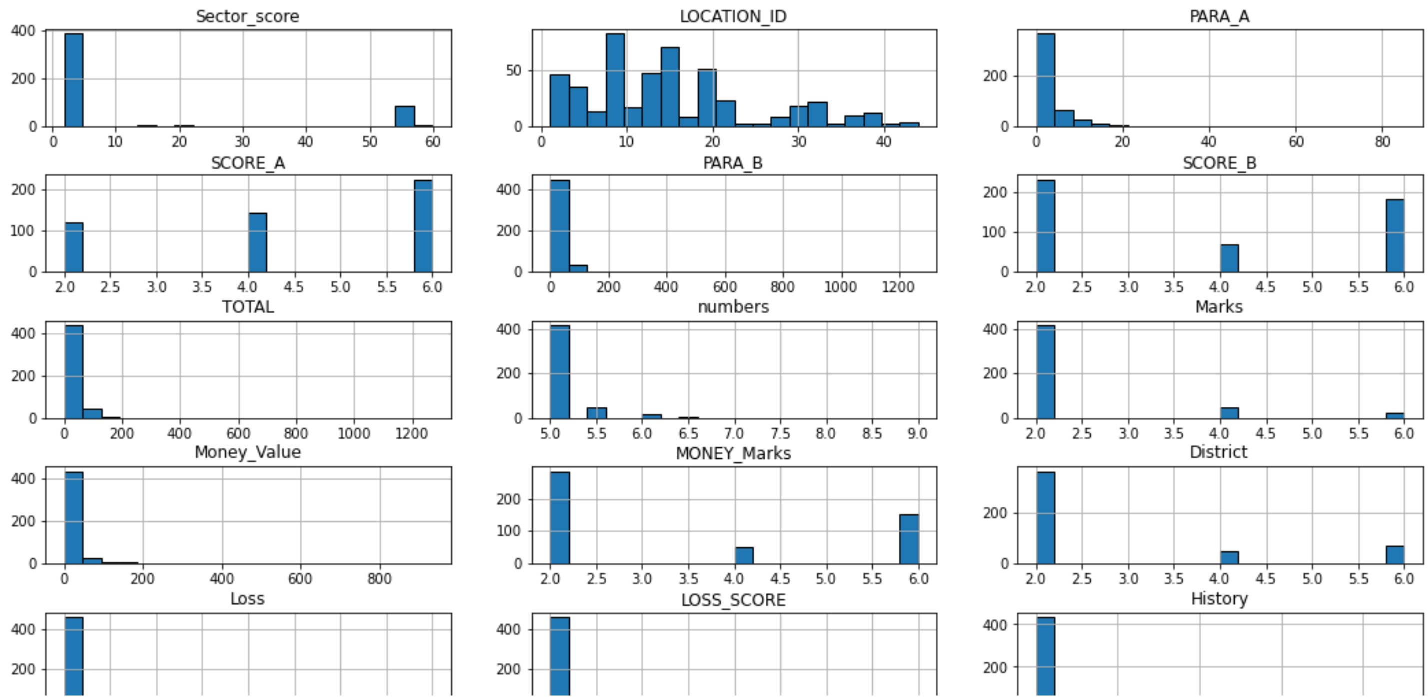
```
<ipython-input-92-5a5f1e0e0c2c>:10: MatplotlibDeprecationWarning: Passing non-integers as
three-element position specification is deprecated since 3.3 and will be removed two mino
r releases later.
  plt.subplot((length/2),3,j+1)
```

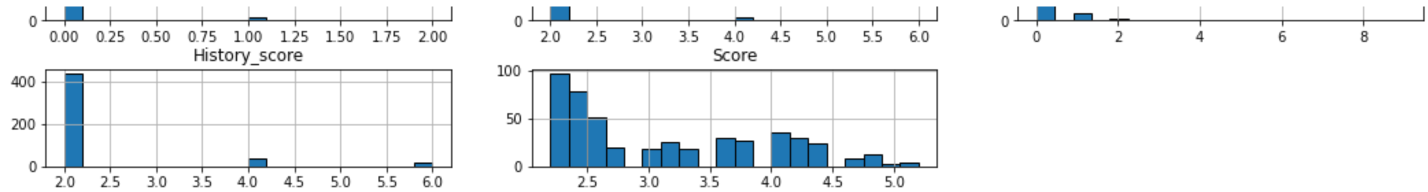


In [93]:

```
## All Plots for risk =1

df1=df[df['Risk']==1]
columns=df1.columns[:17]
plt.subplots(figsize=(18,15))
length=len(columns)
for i,j in itertools.zip_longest(columns,range(length)):
    plt.subplot((length/2),3,j+1)
    plt.subplots_adjust(wspace=0.2,hspace=0.5)
    df1[i].hist(bins=20,edgecolor='black')
    plt.title(i)
plt.show()
```

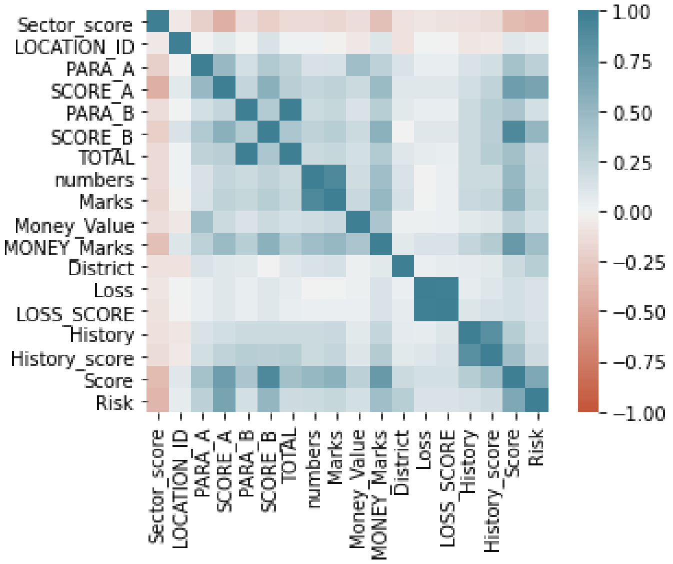
```
<ipython-input-93-010951965789>:8: MatplotlibDeprecationWarning: Passing non-integers as
three-element position specification is deprecated since 3.3 and will be removed two mino
r releases later.
  plt.subplot((length/2),3,j+1)
```





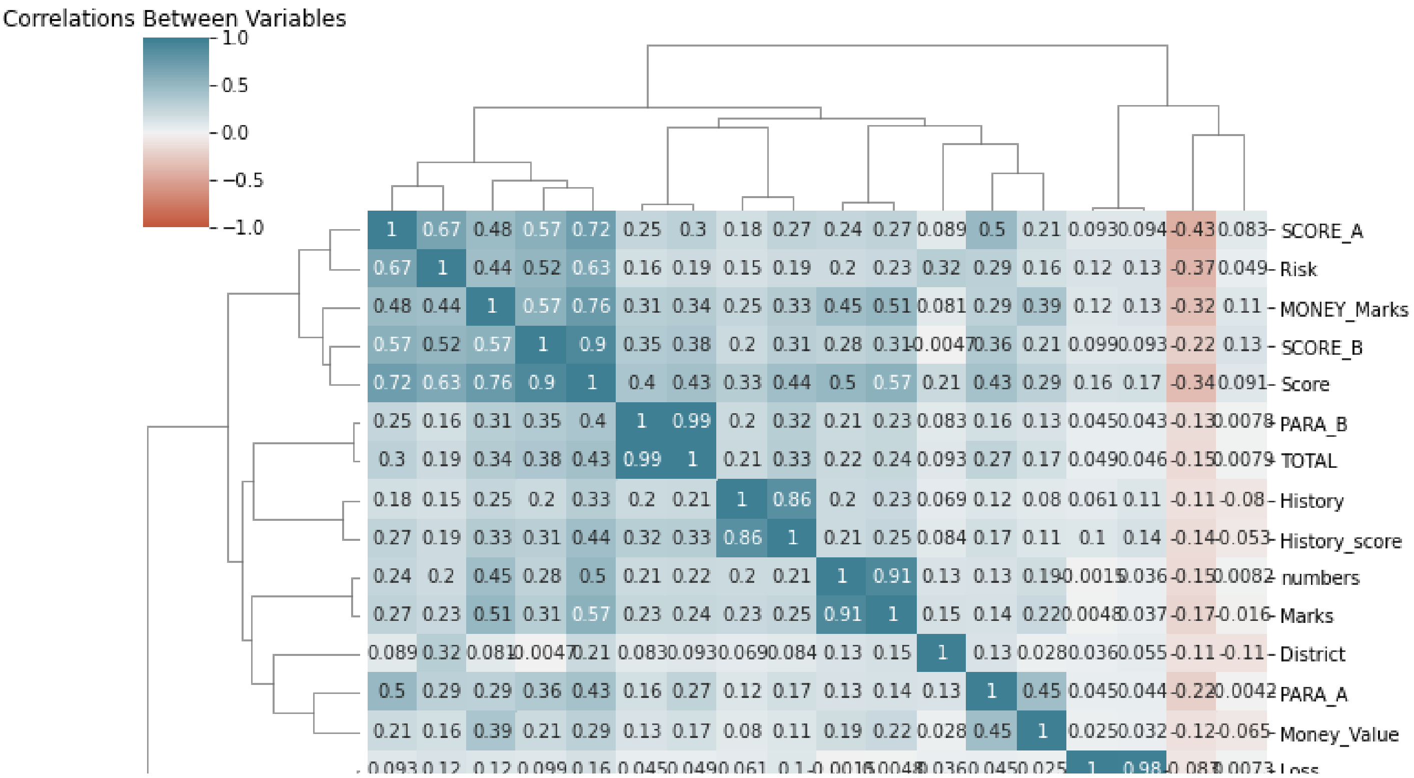
In [94]:

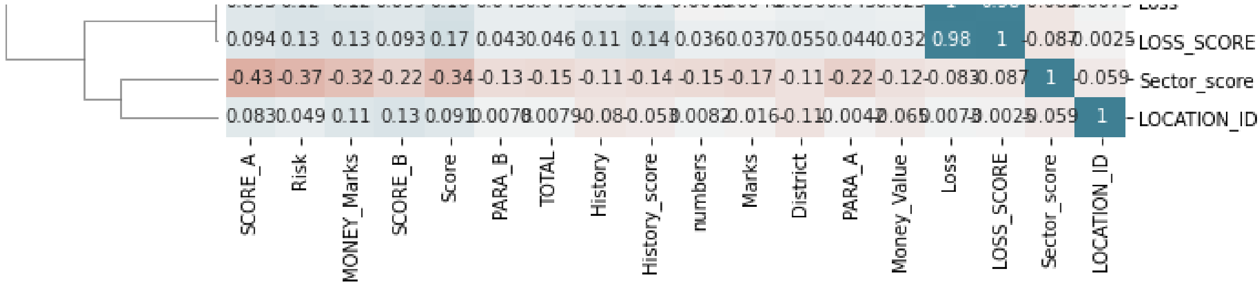
```
corr = df.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
```



In [95]:

```
sns.clustermap(corr, method='ward', cmap=sns.diverging_palette(20, 220, n=200), annot=True,
               vmin=-1, vmax=1, figsize=(10,8))

plt.title("Correlations Between Variables")
#plt.tight_layout()
plt.show()
```





Differentiating into label and features

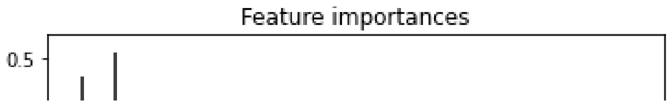
In [96]:

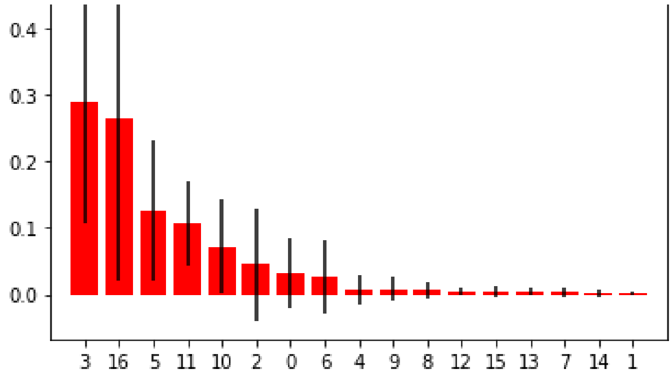
```
y = df.Risk
X = df.drop(['Risk'], 1)
```

In [97]:

```
# Building  a forest and computing the feature importances

from sklearn.ensemble import ExtraTreesClassifier
forest = ExtraTreesClassifier(n_estimators=250, random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

labels = []
for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
   #labels.append(X[f])


# Plotting  the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="red", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 3 (0.289651)
2. feature 16 (0.264210)
3. feature 5 (0.125449)
4. feature 11 (0.106956)
5. feature 10 (0.071360)
6. feature 2 (0.045070)
7. feature 0 (0.032072)
8. feature 6 (0.026229)
9. feature 4 (0.007895)
10. feature 9 (0.007302)
11. feature 8 (0.005674)
12. feature 12 (0.004143)
13. feature 15 (0.004050)
14. feature 13 (0.003474)
15. feature 7 (0.003349)
16. feature 14 (0.002117)
17. feature 1 (0.000999)
```





In [98]:

```
## Selecting the top 10 Important features

features = ['SCORE_A', 'History_score', 'SCORE_B', 'District', 'MONEY_Marks', 'PARA_A', '
Sector_score', 'TOTAL', 'SCORE_A','Money_Value']
X = df[features]
```

Training and Testing

In [99]:

```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size = 0.20, random_
state=1)
```

In [100]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = pd.DataFrame(sc.fit_transform(x_train))
x_test = pd.DataFrame(sc.transform(x_test))
```

Helper functions

In [101]:

```
## Helper function for Metrices

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

def displayMetrics(model, y_true, y_pred):

  #Accuracy Score
  print('Accuracy: ', accuracy_score(y_true, y_pred))

  #Precision Score
  print('Precision Score: ', precision_score(y_true, y_pred, average=None))

  #Recall Score
  print('Recall: ', recall_score(y_true, y_pred, average=None))

  #F1 Score
  print('F1 Score: ', f1_score(y_true, y_pred, average=None))
```

In [102]:

```
## Helper function for recall, precision

from sklearn.metrics import precision_recall_curve, average_precision_score

def plotRecallPrecision(model, testX, testy):
  # predict probabilities
  probs = model.predict_proba(testX)
```

```
  # keep probabilities for the positive outcome only
  probs = probs[:, 1]
  # predict class values
  yhat = model.predict(testX)
  # calculate precision-recall curve
  precision, recall, thresholds = precision_recall_curve(testy, probs)
  # calculate F1 score
  f1 = f1_score(testy, yhat)
  # calculate precision-recall AUC
  auc = auc(recall, precision)
  # calculate average precision score
  ap = average_precision_score(testy, probs)
  print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
  # plot no skill
  plt.plot([0, 1], [0.5, 0.5], linestyle='--')
  # plot the precision-recall curve for the model
  plt.plot(recall, precision, marker='.')
  # show the plot
  plt.xlabel('Recall', fontsize=12)
  plt.ylabel('Precision', fontsize=12)
  plt.title('Precision-Recall Curve', fontsize=12)
  plt.show()
```

In [103]:

```
## Helper function for

# from sklearn.metrics import precision_recall_curve, average_precision_score

# def plotRecallPrecisionSVM(model, testX, testy):
#   # predict probabilities
#   probs = model.decision_function(testX)
#   # keep probabilities for the positive outcome only
#   probs = probs[:, 1]
#   # predict class values
#   yhat = model.predict(testX)
#   # calculate precision-recall curve
#   precision, recall, thresholds = precision_recall_curve(testy, probs)
#   # calculate F1 score
#   f1 = f1_score(testy, yhat)
#   # calculate precision-recall AUC
#   auc = auc(recall, precision)
#   # calculate average precision score
#   ap = average_precision_score(testy, probs)
#   print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
#   # plot no skill
#   plt.plot([0, 1], [0.5, 0.5], linestyle='--')
#   # plot the precision-recall curve for the model
#   plt.plot(recall, precision, marker='.')
#   # show the plot
#   plt.xlabel('Recall', fontsize=12)
#   plt.ylabel('Precision', fontsize=12)
#   plt.title('Precision-Recall Curve', fontsize=12)
#   plt.show()
```

In [104]:

```
from sklearn.metrics import mean_squared_error, r2_score
def printLinearModels(model, x_train, x_test, y_train, y_test):

  y_predicted = model.predict(x_test)
  rmse = mean_squared_error(y_test, y_predicted)
  r2 = r2_score(y_test, y_predicted)

  # printing values
#   print('Slope:' ,model.coef_)
#   print('Intercept:', model.intercept_)
  print('Root mean squared error: ', rmse)
  print('R2 score: ', r2)
```

In [105]:

```
## Helper function for Confusion matrix plot

from sklearn.metrics import confusion_matrix
from sklearn import metrics

def plotConfusionMatrix(y_true, y_pred):

    cnf_matrix = metrics.confusion_matrix(y_true, y_pred)
    class_names=[0,1] # name  of classes
    fig, ax = plt.subplots(figsize=(6,4)) # resize the size of cnfsn matrix
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names)
    plt.yticks(tick_marks, class_names)

    # create heatmap
    sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

In [106]:

```
## Helper function for ROC curve plot

def plotROC(y_true, y_pred):

    fpr, tpr, _ = metrics.roc_curve(y_true,  y_pred)
    auc = metrics.roc_auc_score(y_true, y_pred)
    plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % auc)
    plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.rcParams['font.size'] = 12
    plt.title('ROC curve for treatment classifier')
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.legend(loc="lower right")
    plt.show()
```

In [ ]:

```

```

In [107]:

```
## Helper function for the Model comparison graph

from sklearn import model_selection
seed = 7
scoring = 'accuracy'

def modelComparison(models, x_train, y_train):
  results = []
  names = []
  for model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, x_train, y_train, cv=kfold)
    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.figure(figsize=(8,6))
    plt.boxplot(results)
#     ax.set_xticklabels(names)
```

```
    plt.show()
```

In [108]:

```
## cross validation

from sklearn import model_selection
from sklearn.model_selection import cross_val_score

seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureW
arning: Setting a random_state has no effect since shuffle is False. This will raise an e
rror in 0.24. You should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
```

Models

In [109]:

```
results = []
```

Linear Regression

In [110]:

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
printLinearModels(reg, x_train, x_test, y_train, y_test)
```

```
Root mean squared error:  0.19364422314868343
R2 score:  0.2097055484577629
```

In [111]:

```
cv_results = model_selection.cross_val_score(reg, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('LinearRegression', cv_results.mean(), cv_results.std())
print(msg)
```

```
LinearRegression: 0.116881 (0.903867)
```

Logistic Regression

In [112]:

```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression(solver='newton-cg', multi_class='auto', max_iter=1000)
logReg.fit(x_train,y_train)
```

Out[112]:

```
LogisticRegression(max_iter=1000, solver='newton-cg')
```

In [113]:

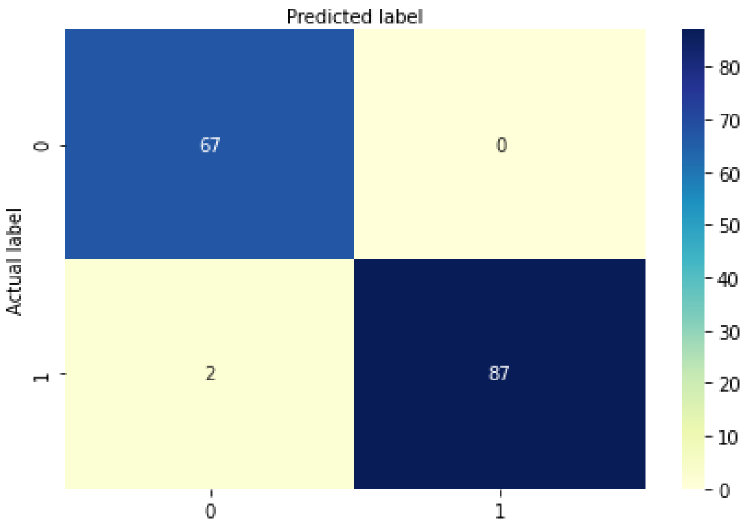
```
y_pred = logReg.predict(x_test)
displayMetrics(logReg, y_test, y_pred)
```

```
Accuracy:  0.9871794871794872
Precision Score:  [0.97101449 1.        ]
Recall:  [1.         0.97752809]
F1 Score:  [0.98529412 0.98863636]
```

In [114]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

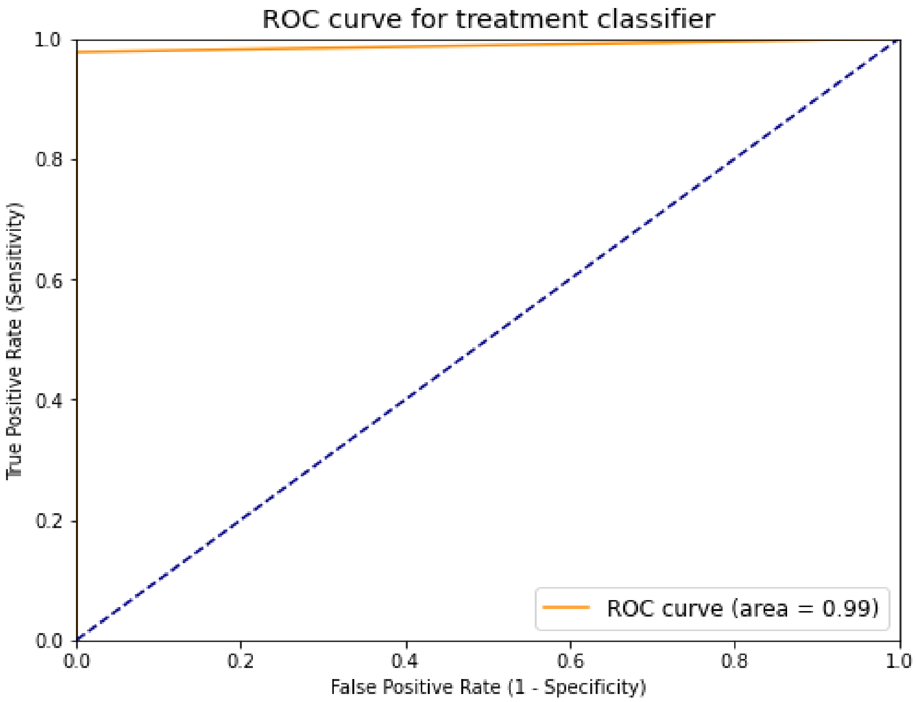
```
 Confusion matrix 
```



In [115]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve 
```



In [116]:

```
cv_results = model_selection.cross_val_score(logReg, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('LogisticRegression', cv_results.mean(), cv_results.std())
print(msg)
```

```
LogisticRegression: 0.980417 (0.029933)
```

SGD Classifier

In [117]:

```
from sklearn.linear_model import SGDClassifier
```

```
sgdCls = SGDClassifier(max_iter=1000)
sgdCls.fit(x_train, y_train)
```

Out[117]:

```
SGDClassifier()
```

In [118]:

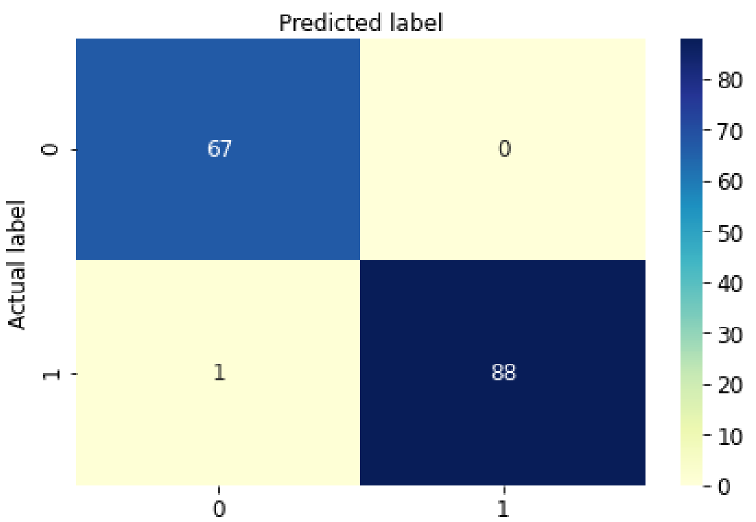
```
y_pred = sgdCls.predict(x_test)
displayMetrics(sgdCls, y_test, y_pred)
```

```
Accuracy:  0.9935897435897436
Precision Score:  [0.98529412 1.        ]
Recall:  [1.         0.98876404]
F1 Score:  [0.99259259 0.99435028]
```

In [119]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

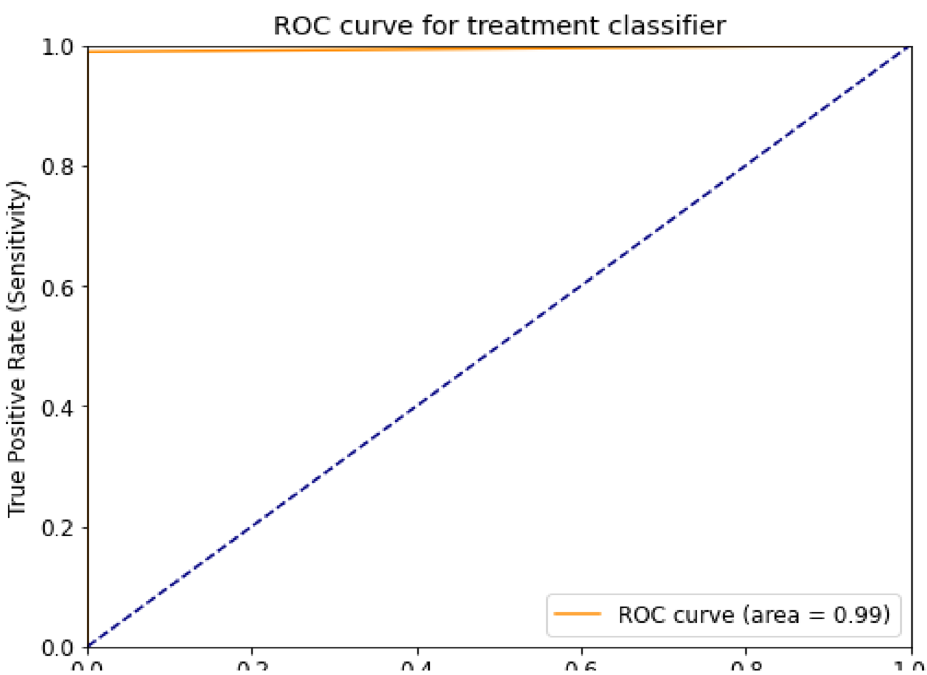
```
 Confusion matrix
```



In [120]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



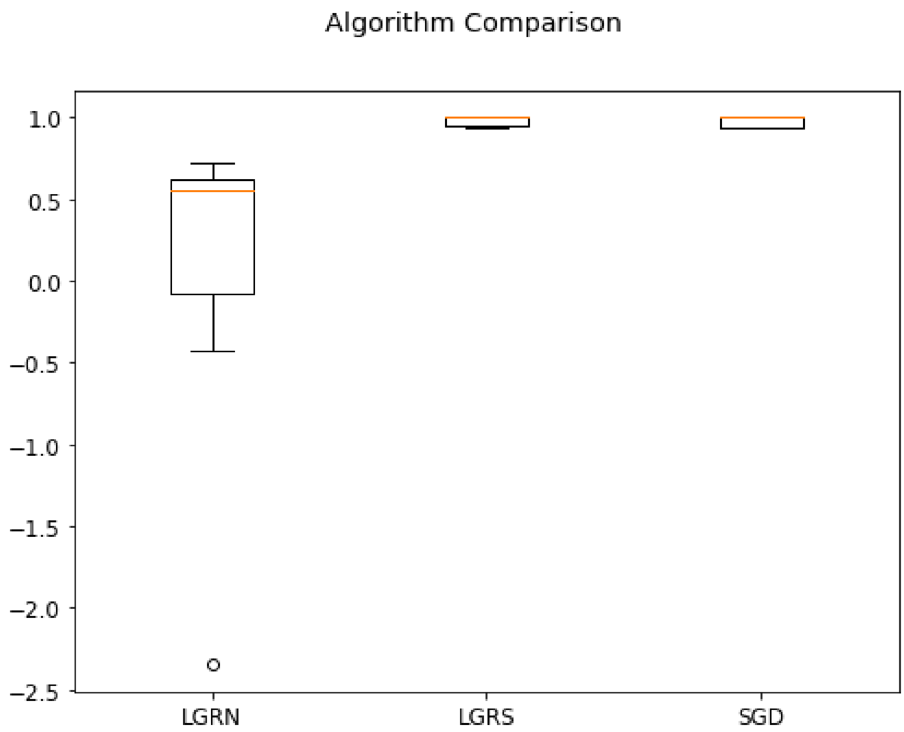
In [121]:

```
cv_results = model_selection.cross_val_score(sgdCls, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('SGDClassifier', cv_results.mean(), cv_results.std())
print(msg)
```

```
SGDClassifier: 0.973750 (0.032170)
```

In [122]:

```
names=['LGRN','LGRS','SGD']

fig = plt.figure(figsize = (8,6) )
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Geometric Models

In [123]:

```
results = []
```

Support Vector Machines

In [124]:

```
from sklearn.svm import SVC
svcModel = SVC(gamma='auto', probability=True)
svcModel.fit(x_train,y_train)
```

Out[124]:

```
SVC(gamma='auto', probability=True)
```

In [125]:

```
y_pred = svcModel.predict(x_test)
```

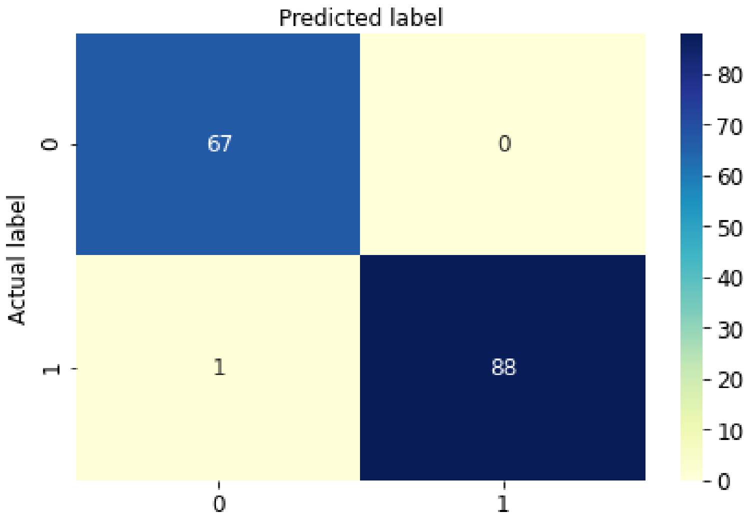
```
displayMetrics(svcModel, y_test, y_pred)
```

```
Accuracy:  0.9935897435897436
Precision Score:  [0.98529412 1.        ]
Recall:  [1.         0.98876404]
F1 Score:  [0.99259259 0.99435028]
```

In [126]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

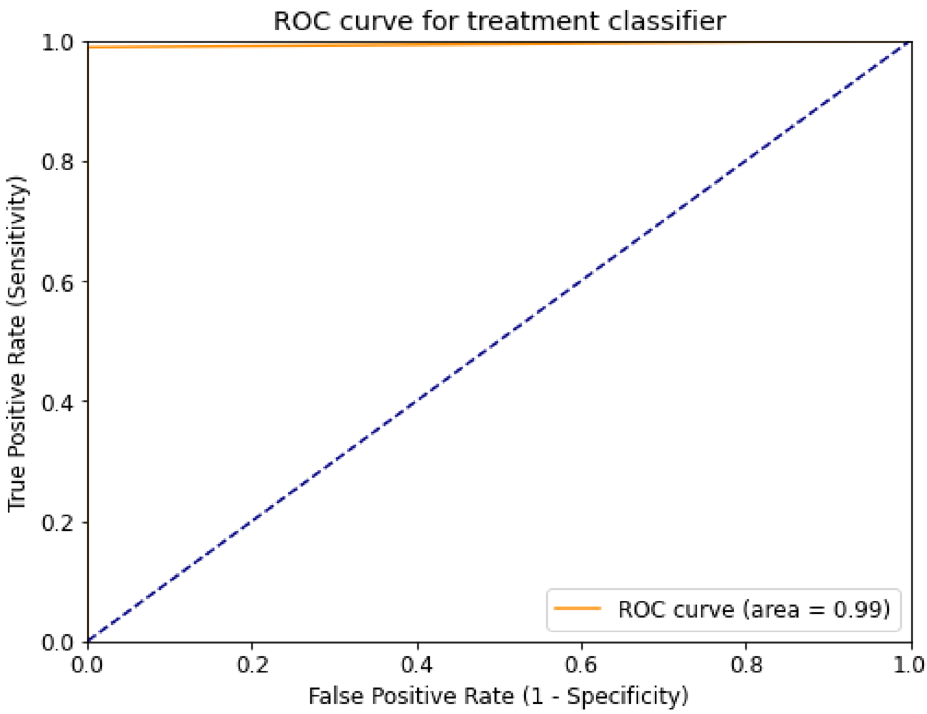
```
 Confusion matrix
```



In [127]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



In [128]:

```
cv_results = model_selection.cross_val_score(svcModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('SVC', cv_results.mean(), cv_results.std())
print(msg)
```

```
SVC: 0.987083 (0.025850)
```

KNN using manhattan distance

In [129]:

```
from sklearn.neighbors import KNeighborsClassifier

knnModel = KNeighborsClassifier(p=1, n_neighbors=8)
knnModel.fit(x_train,y_train)
```

Out[129]:

```
KNeighborsClassifier(n_neighbors=8, p=1)
```

In [130]:

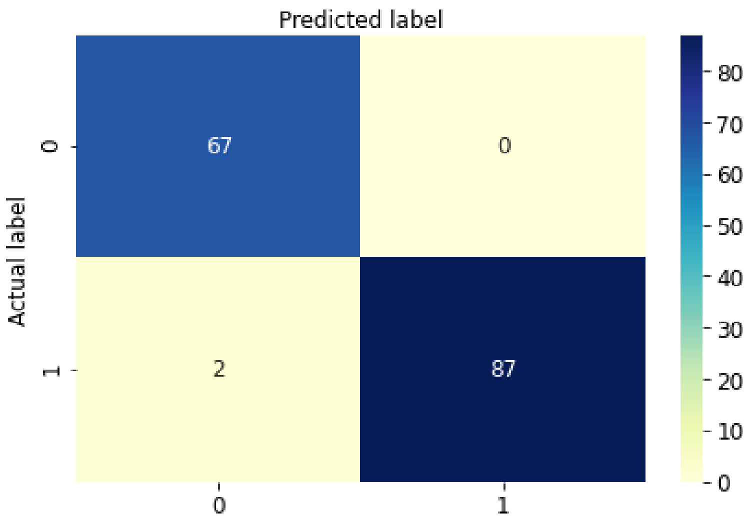
```
y_pred = knnModel.predict(x_test)
displayMetrics(knnModel, y_test, y_pred)
```

```
Accuracy:  0.9871794871794872
Precision Score:  [0.97101449 1.        ]
Recall:  [1.         0.97752809]
F1 Score:  [0.98529412 0.98863636]
```

In [131]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

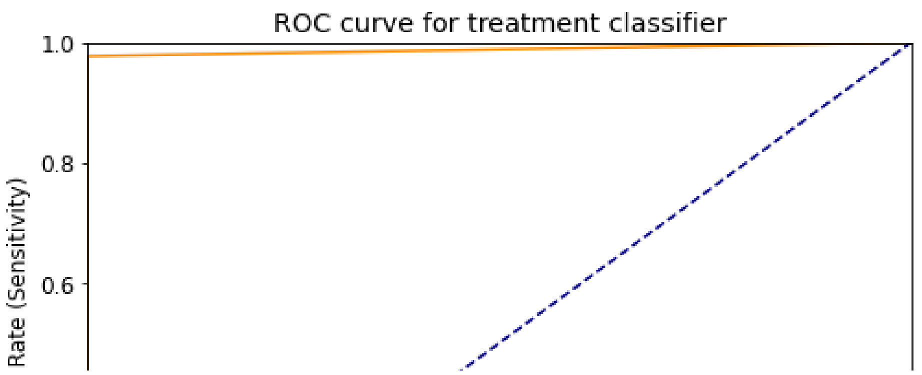
```
 Confusion matrix
```

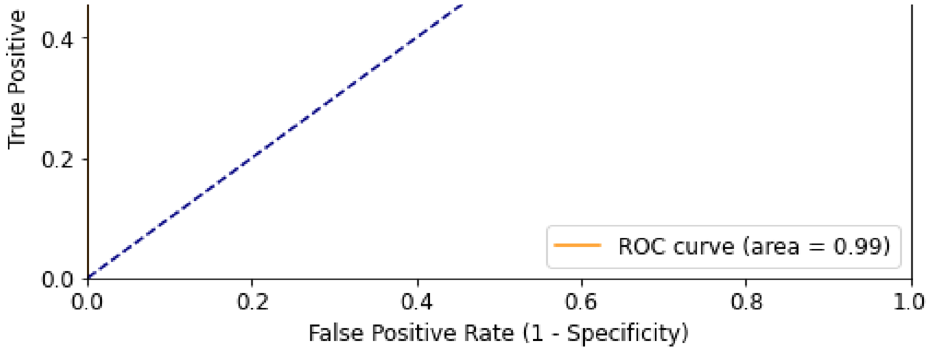


In [132]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```





In [133]:

```
cv_results = model_selection.cross_val_score(knnModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('KNN-M', cv_results.mean(), cv_results.std())
print(msg)
```

```
KNN-M: 0.904167 (0.051269)
```

KNN using euclidean_distance

In [134]:

```
knnModelE = KNeighborsClassifier(p=2, n_neighbors=8)
knnModelE.fit(x_train,y_train)
```

Out[134]:

```
KNeighborsClassifier(n_neighbors=8)
```

In [135]:

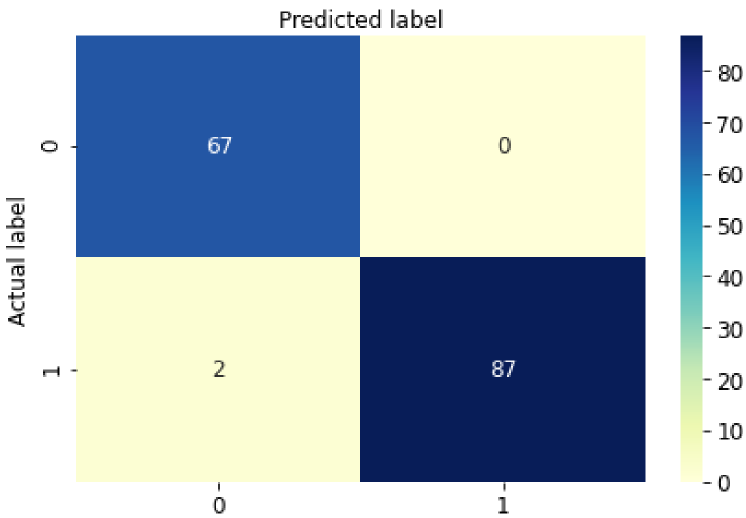
```
y_pred = knnModelE.predict(x_test)
displayMetrics(knnModelE, y_test, y_pred)
```

```
Accuracy:  0.9871794871794872
Precision Score:  [0.97101449 1.        ]
Recall:  [1.         0.97752809]
F1 Score:  [0.98529412 0.98863636]
```

In [136]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

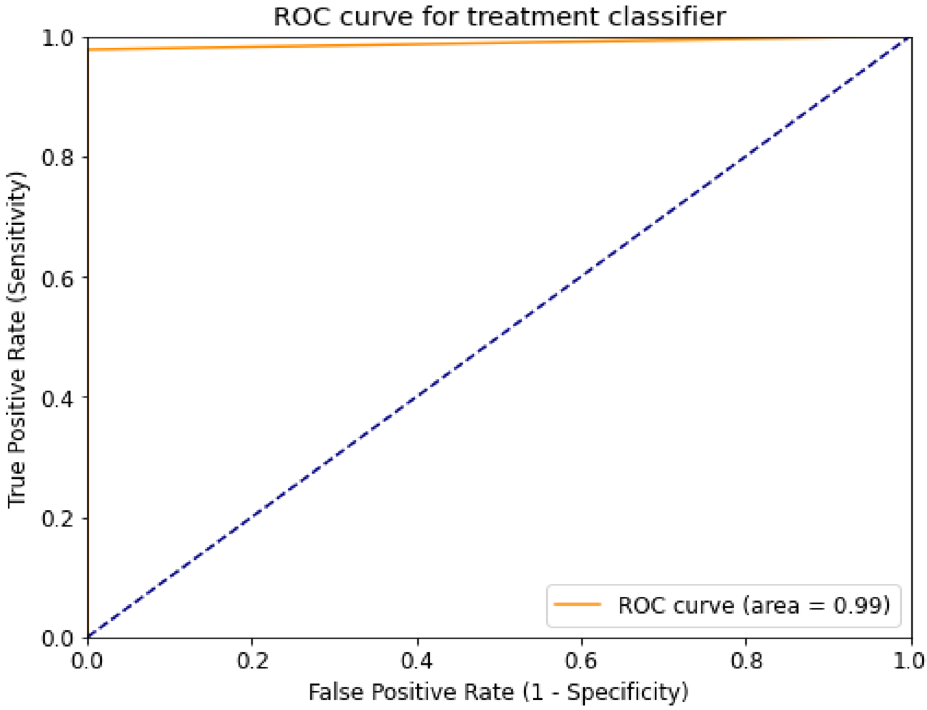
```
 Confusion matrix
```



In [137]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



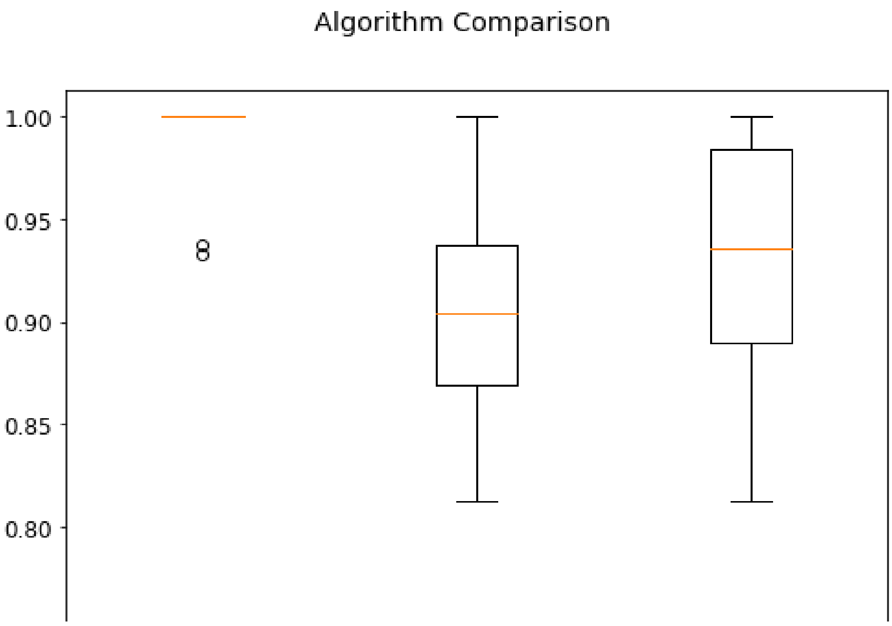
In [138]:

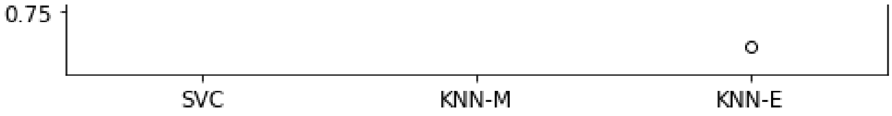
```
cv_results = model_selection.cross_val_score(knnModelE, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('KNN-E', cv_results.mean(), cv_results.std())
print(msg)
```

```
KNN-E: 0.916250 (0.082736)
```

In [139]:

```
## Comparison of various Geomatric models

names=['SVC', 'KNN-M','KNN-E']

fig = plt.figure(figsize = (8,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```





Probabilistic model

In [140]:

```
results = []
```

In [141]:

```
from sklearn.naive_bayes import GaussianNB
gnbModel = GaussianNB()
gnbModel.fit(x_train,y_train)
```

Out[141]:

```
GaussianNB()
```

In [142]:

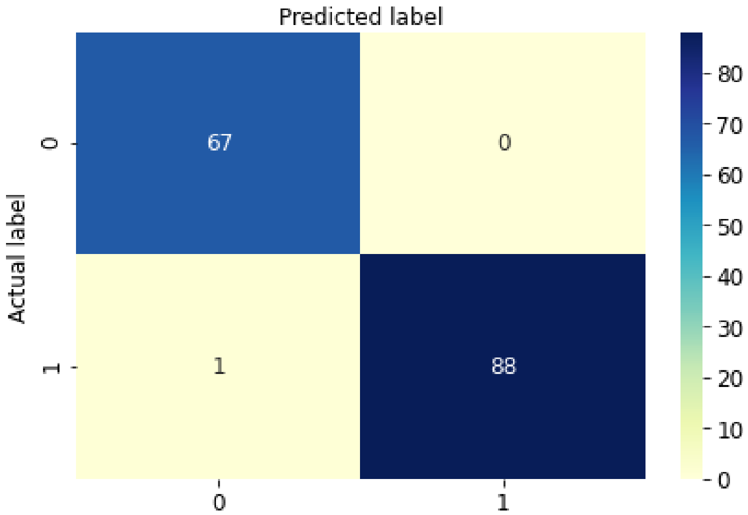
```
y_pred = gnbModel.predict(x_test)
displayMetrics(gnbModel, y_test, y_pred)
```

```
Accuracy:  0.9935897435897436
Precision Score:  [0.98529412 1.        ]
Recall:  [1.         0.98876404]
F1 Score:  [0.99259259 0.99435028]
```

In [143]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

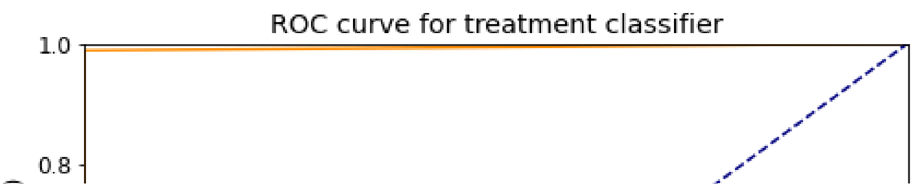
```
 Confusion matrix
```

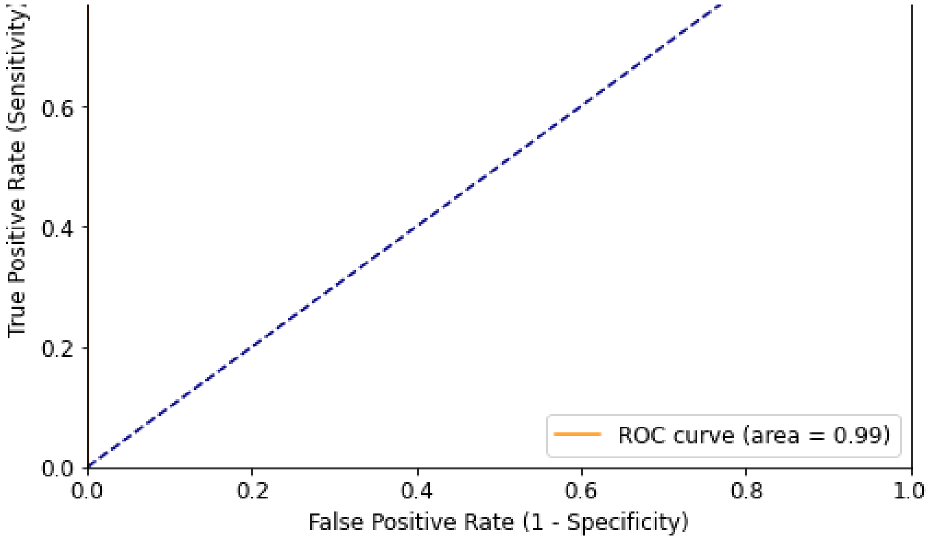


In [144]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```





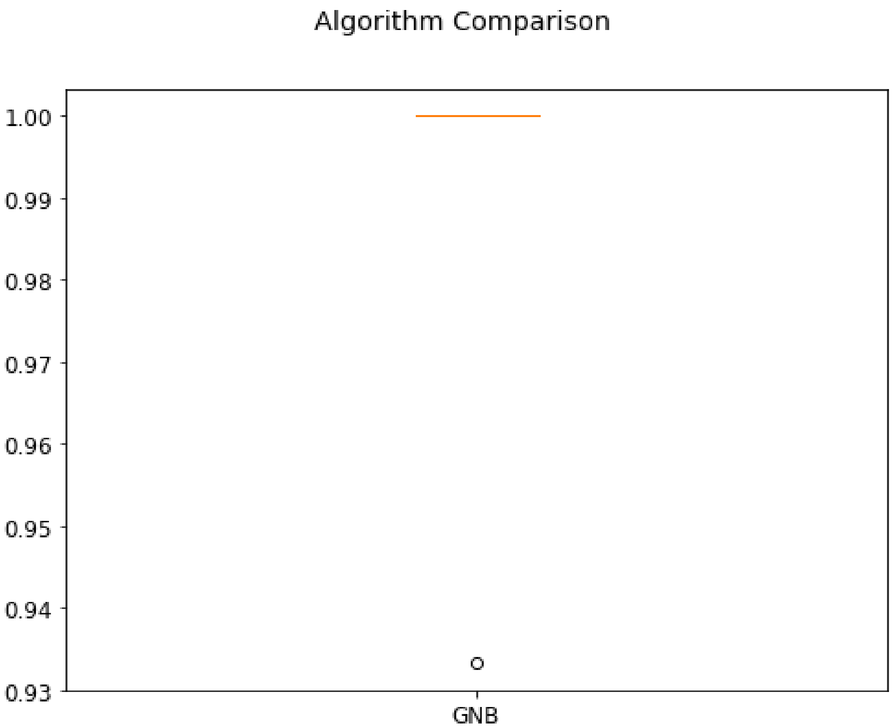
In [145]:

```
cv_results = model_selection.cross_val_score(gnbModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('GNB', cv_results.mean(), cv_results.std())
print(msg)
```

GNB: 0.993333 (0.020000)

In [146]:

```
names=['GNB']

fig = plt.figure(figsize = (8,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Tree Based Model

In [147]:

```
from sklearn import tree
treeModel = tree.DecisionTreeClassifier(max_depth=8, max_features='auto', min_samples_sp
```

```
lit = 4)
treeModel = treeModel.fit(x_train,y_train)
```

In [148]:

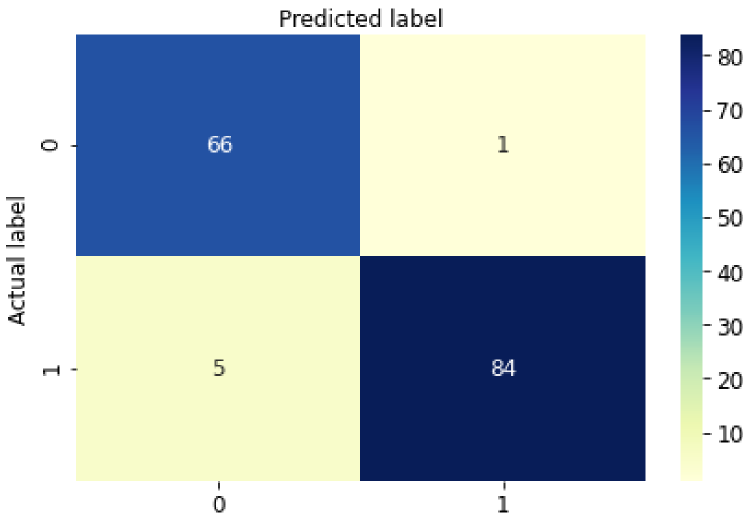
```
y_pred = treeModel.predict(x_test)
displayMetrics(treeModel, y_test, y_pred)
```

```
Accuracy:  0.9615384615384616
Precision Score:  [0.92957746 0.98823529]
Recall:  [0.98507463 0.94382022]
F1 Score:  [0.95652174 0.96551724]
```

In [149]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

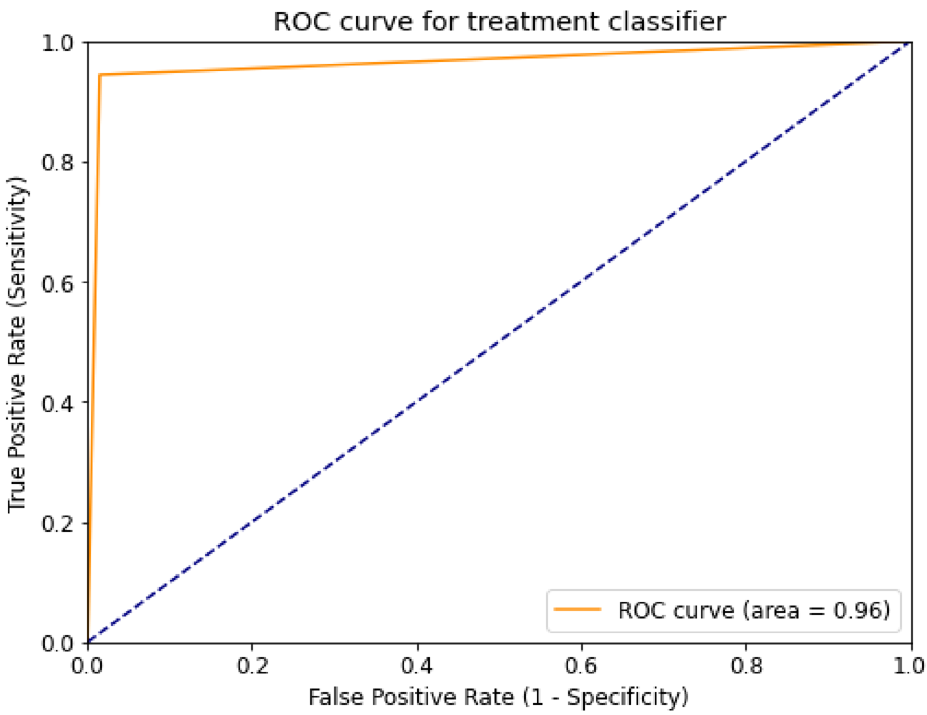
```
 Confusion matrix
```



In [153]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



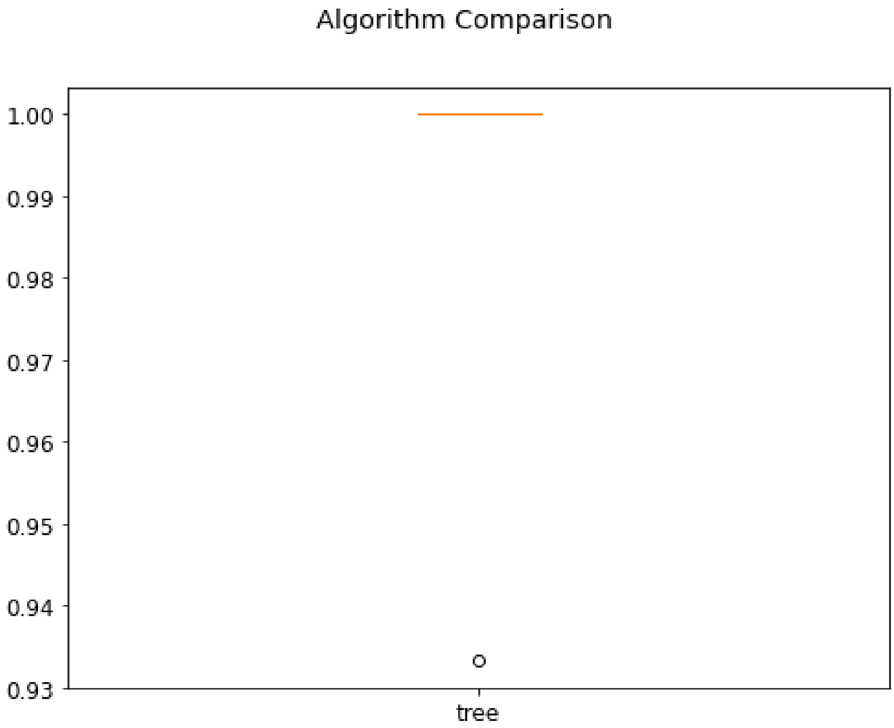
In [193]:

```
import graphviz
dot_data = tree.export_graphviz(treeModel, out_file=None,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
#graph.render("Restaurant risk indicator")
```

```
  File "<ipython-input-193-60e289e93d7c>", line 1
    pip install graphviz
              ^
SyntaxError: invalid syntax
```

In [155]:

```
names=['tree']

fig = plt.figure(figsize = (8,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Ensembler

In [156]:

```
results = []
```

Voting

In [157]:

```
from sklearn import ensemble
from sklearn.linear_model import RidgeClassifier
ridgeCls = RidgeClassifier()
votModel = ensemble.VotingClassifier(estimators=[('RC', ridgeCls), ('KNN', knnModel), ('
TR', treeModel)])
votModel.fit(x_train, y_train)
```

Out[157]:

```
VotingClassifier(estimators=[('RC', RidgeClassifier()),
                             ('KNN', KNeighborsClassifier(n_neighbors=8, p=1)),
                             ('TR',
                              DecisionTreeClassifier(max_depth=8,
                                                     max_features='auto',
                                                     min_samples_split=4))])
```

In [158]:

```
y_pred = votModel.predict(x_test)
displayMetrics(votModel, y_test, y_pred)
```

```
Accuracy:  0.9807692307692307
Precision Score:  [0.95714286 1.        ]
Recall:  [1.         0.96629213]
F1 Score:  [0.97810219 0.98285714]
```

In [159]:

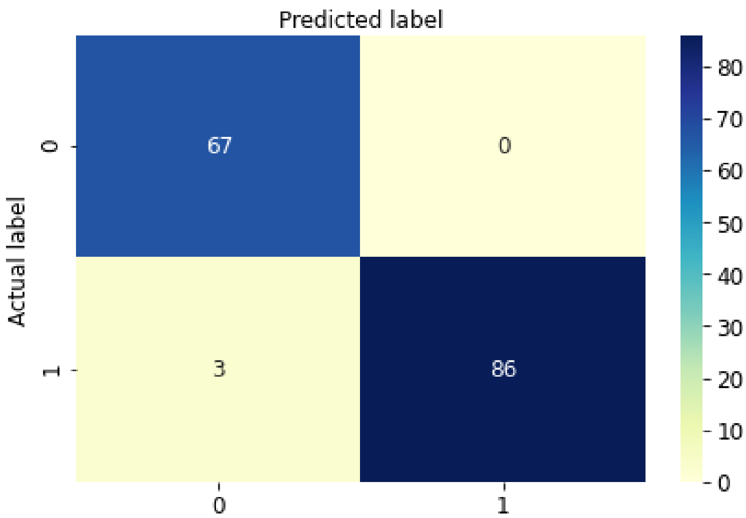
```
printLinearModels(votModel, x_train, x_test, y_train, y_test)
```

```
Root mean squared error:  0.019230769230769232
R2 score:  0.9215160154284756
```

In [160]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

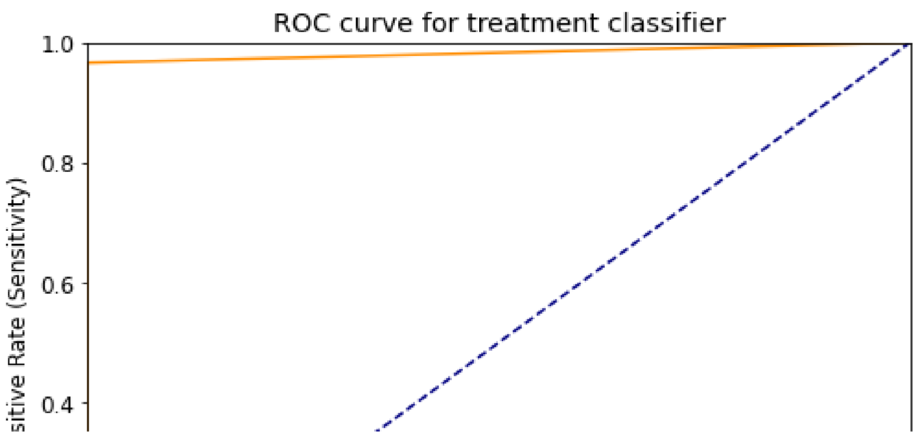
```
 Confusion matrix
```

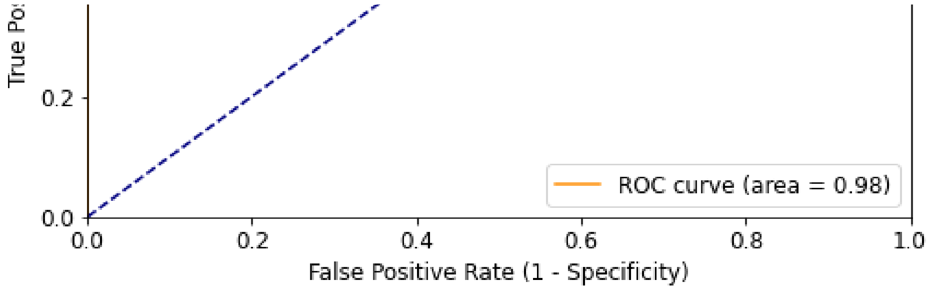


In [161]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```





In [162]:

```
cv_results = model_selection.cross_val_score(votModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('Voting', cv_results.mean(), cv_results.std())
print(msg)
```

```
Voting: 0.917083 (0.056687)
```

Bagging

In [163]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
baggingModel = BaggingClassifier(KNeighborsClassifier(p=1, n_neighbors=8))
baggingModel.fit(x_train,y_train)
```

Out[163]:

```
BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=8, p=1))
```

In [164]:

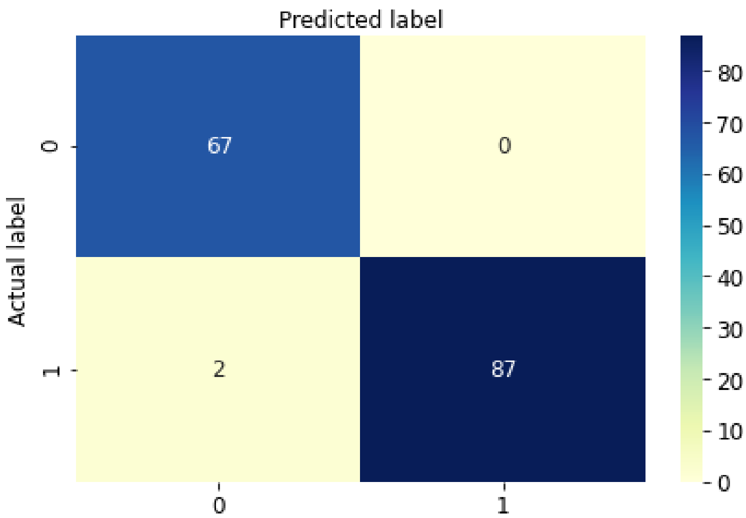
```
y_pred = baggingModel.predict(x_test)
displayMetrics(baggingModel, y_test, y_pred)
```

```
Accuracy:  0.9871794871794872
Precision Score:  [0.97101449 1.        ]
Recall:  [1.         0.97752809]
F1 Score:  [0.98529412 0.98863636]
```

In [165]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

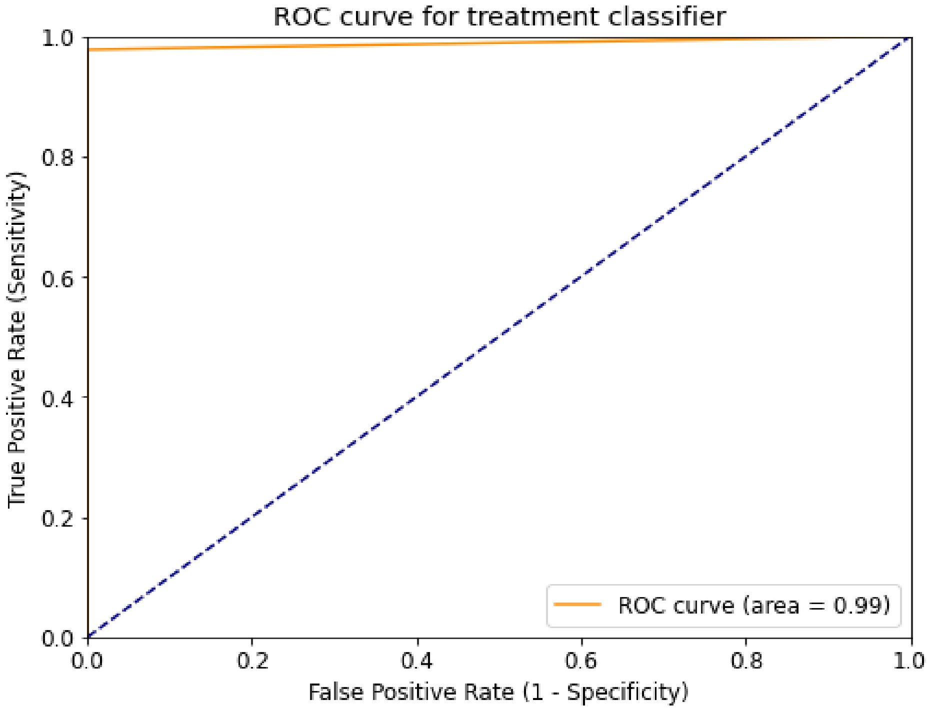
```
 Confusion matrix
```



In [166]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



In [167]:

```
cv_results = model_selection.cross_val_score(baggingModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('BC', cv_results.mean(), cv_results.std())
print(msg)
```

```
BC: 0.923750 (0.054232)
```

RandomForestClassifier

In [168]:

```
from sklearn.ensemble import RandomForestClassifier
rfcModel = RandomForestClassifier(n_estimators=2, max_depth=6)
rfcModel.fit(x_train,y_train)
```

Out[168]:

```
RandomForestClassifier(max_depth=6, n_estimators=2)
```

In [169]:

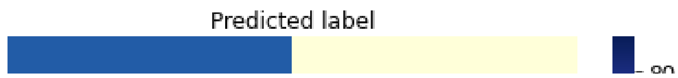
```
y_pred = rfcModel.predict(x_test)
displayMetrics(rfcModel, y_test, y_pred)
```

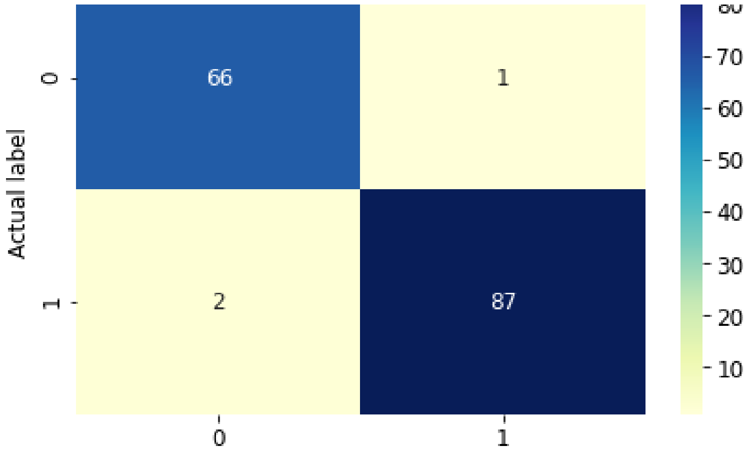
```
Accuracy:  0.9807692307692307
Precision Score:  [0.97058824 0.98863636]
Recall:  [0.98507463 0.97752809]
F1 Score:  [0.97777778 0.98305085]
```

In [170]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

```
 Confusion matrix
```

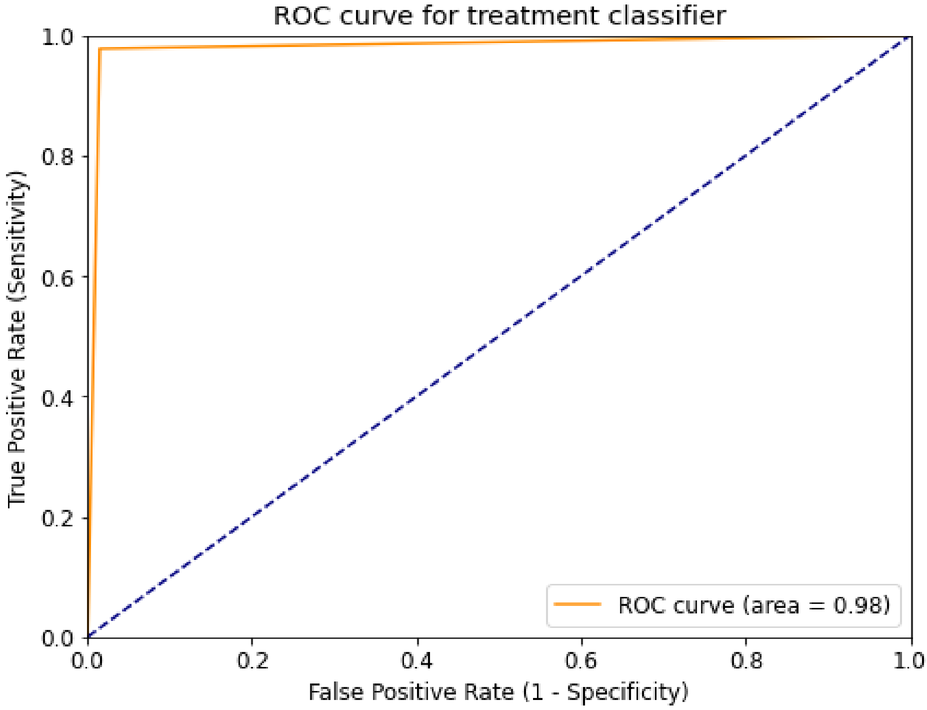




In [171]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

 ROC curve



In [172]:

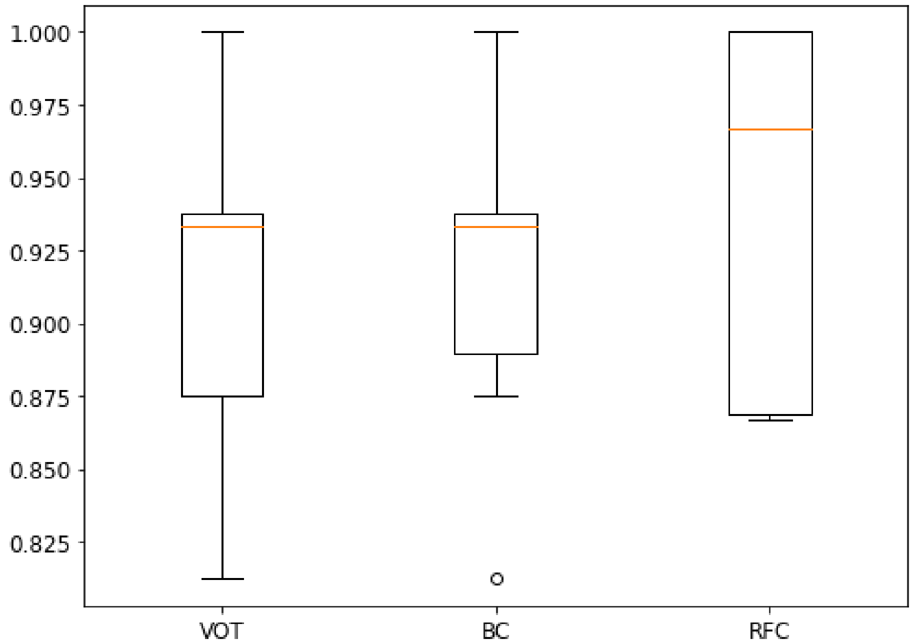
```
cv_results = model_selection.cross_val_score(rfcModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('RFC', cv_results.mean(), cv_results.std())
print(msg)
```

```
RFC: 0.940833 (0.061964)
```

In [173]:

```
## Comparing Ensemblers

names=['VOT','BC','RFC']

fig = plt.figure(figsize = (8,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Algorithm Comparison



Neural Network

In [174]:

```
results =[]
```

In [175]:

```
from sklearn.neural_network import MLPClassifier
nn_mclf = MLPClassifier(max_iter=50,solver='sgd',verbose='true',validation_fraction=0.0)
nn_mclf.fit(x_train, y_train)
```

```
Iteration 1, loss = 0.73557524
Iteration 2, loss = 0.72720831
Iteration 3, loss = 0.71464562
Iteration 4, loss = 0.69963193
Iteration 5, loss = 0.68402832
Iteration 6, loss = 0.66822773
Iteration 7, loss = 0.65253549
Iteration 8, loss = 0.63711285
Iteration 9, loss = 0.62263912
Iteration 10, loss = 0.60910005
Iteration 11, loss = 0.59624801
Iteration 12, loss = 0.58390965
Iteration 13, loss = 0.57227670
Iteration 14, loss = 0.56124721
Iteration 15, loss = 0.55068165
Iteration 16, loss = 0.54054543
Iteration 17, loss = 0.53105281
Iteration 18, loss = 0.52196267
Iteration 19, loss = 0.51319726
Iteration 20, loss = 0.50485898
Iteration 21, loss = 0.49693938
Iteration 22, loss = 0.48935554
Iteration 23, loss = 0.48213615
Iteration 24, loss = 0.47517653
Iteration 25, loss = 0.46861303
Iteration 26, loss = 0.46239545
Iteration 27, loss = 0.45643521
Iteration 28, loss = 0.45074101
Iteration 29, loss = 0.44530268
Iteration 30, loss = 0.44003791
Iteration 31, loss = 0.43512170
Iteration 32, loss = 0.43024302
Iteration 33, loss = 0.42561346
Iteration 34, loss = 0.42103109
Iteration 35, loss = 0.41664525
```

```
Iteration 36, loss = 0.41237113
Iteration 37, loss = 0.40823111
Iteration 38, loss = 0.40421602
Iteration 39, loss = 0.40026993
Iteration 40, loss = 0.39638908
Iteration 41, loss = 0.39263123
Iteration 42, loss = 0.38897225
Iteration 43, loss = 0.38543277
Iteration 44, loss = 0.38204218
Iteration 45, loss = 0.37870222
Iteration 46, loss = 0.37545702
Iteration 47, loss = 0.37223788
Iteration 48, loss = 0.36904411
Iteration 49, loss = 0.36594831
Iteration 50, loss = 0.36297403
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.
py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and the
optimization hasn't converged yet.
  warnings.warn(
```

Out[175]:

```
MLPClassifier(max_iter=50, solver='sgd', validation_fraction=0.0,
              verbose='true')
```

In [176]:

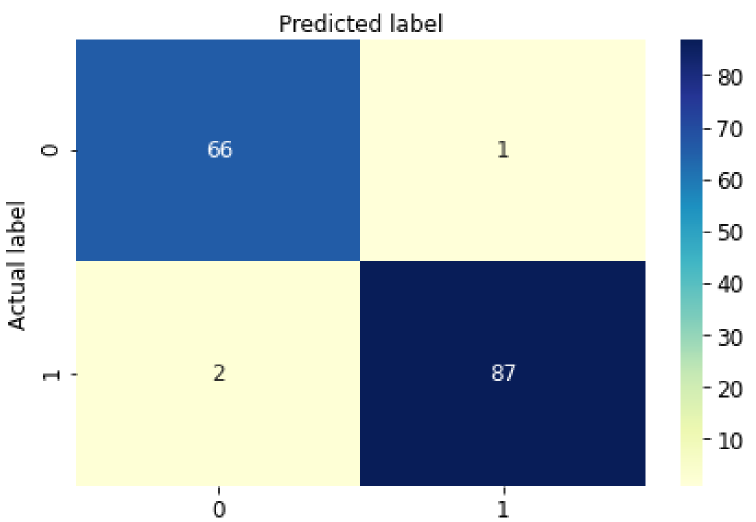
```
y_pred = rfcModel.predict(x_test)
displayMetrics(rfcModel, y_test, y_pred)
```

```
Accuracy:  0.9807692307692307
Precision Score:  [0.97058824 0.98863636]
Recall:  [0.98507463 0.97752809]
F1 Score:  [0.97777778 0.98305085]
```

In [177]:

```
print('\n Confusion matrix \n')
plotConfusionMatrix(y_test, y_pred)
```

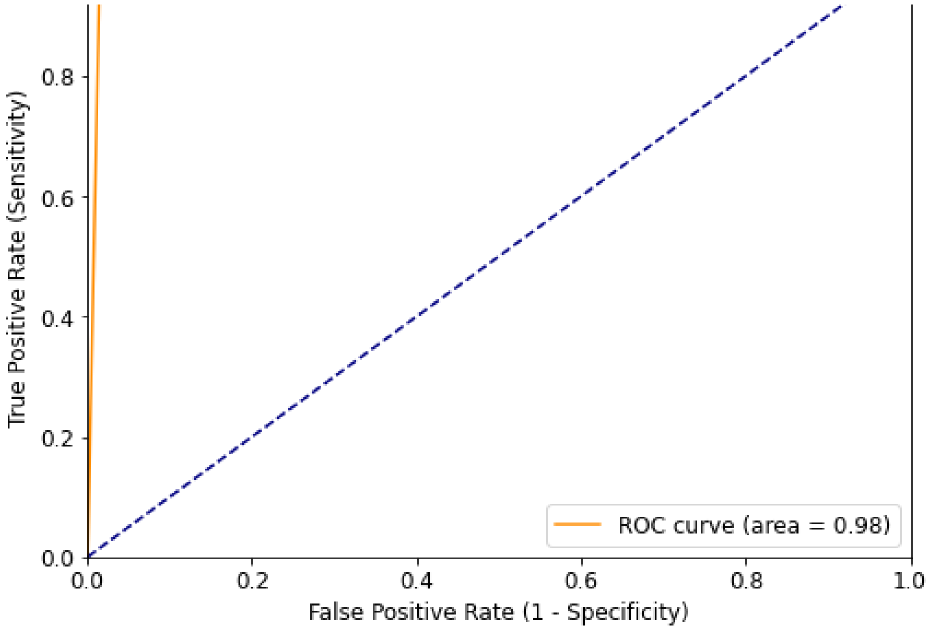
```
 Confusion matrix
```



In [178]:

```
print("\n ROC curve \n")
plotROC(y_test, y_pred)
```

```
 ROC curve
```



In [179]:

```
cv_results = model_selection.cross_val_score(rfcModel, x_test, y_test, cv=kfold)
results.append(cv_results)
msg = "%s: %f (%f)" % ('MLP', cv_results.mean(), cv_results.std())
print(msg)
```

```
MLP: 0.922917 (0.038290)
```

In [180]:

```
names=['MLP']

fig = plt.figure(figsize = (8,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```