In [469]:

```
# In[1]:


#Import environments:

import pandas as pd
import numpy as np
import sklearn
import os
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve
import seaborn as sns
import matplotlib.pyplot as plt
```

In [470]:

```
# In[ ]:


#Load Data:

airport = pd.read_excel(os.path.expanduser("/Users/carlosalvarez/Desktop/NEWYORK
_BOSTON_WEATHER_2018.xlsx" ))
```

In [471]:

```
# In[ ]:


airport.shape
```

Out[471]:

```
(5644, 24)
```

In [472]:

```
# In[ ]:


airport.isnull().values.any()
```

Out[472]:

```
True
```

In [473]:

```
# In[ ]:


airport.isnull().sum()
```

Out[473]:

```
YEAR                    0
QUARTER                 0
MONTH                   0
DAY_OF_MONTH            0
TEMP_DEP                0
FORE_DEP                0
TEMP_ARR                0
FORE_ARR                0
ORIGIN                  0
ORIGIN_CITY_NAME        0
ORIGIN_STATE_ABR        0
DEST                    0
DEST_CITY_NAME          0
DEST_STATE_ABR          0
CRS_DEP_TIME            0
DEP_TIME              185
DEP_DELAY             217
DEP_DEL15             217
CRS_ARR_TIME            0
ARR_TIME              192
ARR_DELAY             212
ARR_DEL15             212
CANCELLED               0
DISTANCE                0
dtype: int64
```

In [474]:

```
# In[ ]:


columns_to_drop = ["ORIGIN_STATE_ABR","DEST_CITY_NAME","DEST_STATE_ABR","ORIGIN_
CITY_NAME"]
```

In [475]:

```
# In[ ]:


airport.fillna(value=0, inplace=True)
airport.isnull().values.any()
```

Out[475]:

False

In [476]:

```
# In[ ]:


import math

for index, row in airport.iterrows():
    airport.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME'] / 100)
airport.head()
```

Out[476]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	TEMP_DEP	FORE_DEP	TEMP_ARR	FORE_A
0	2018	1	1	1	12.5	1	6.75	
1	2018	1	1	1	12.5	1	6.75	
2	2018	1	1	1	12.5	1	6.75	
3	2018	1	1	1	12.5	1	6.75	
4	2018	1	1	1	12.5	1	6.75	

5 rows × 24 columns

In [477]:

```
# In[ ]:


airport = pd.get_dummies(airport, columns=['ORIGIN','DEST'])
airport.head()
```

Out[477]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	TEMP_DEP	FORE_DEP	TEMP_ARR	FORE_A
0	2018	1	1	1	12.5	1	6.75	
1	2018	1	1	1	12.5	1	6.75	
2	2018	1	1	1	12.5	1	6.75	
3	2018	1	1	1	12.5	1	6.75	
4	2018	1	1	1	12.5	1	6.75	

5 rows × 24 columns

In [478]:

```
# In[ ]:


airport.drop(labels=columns_to_drop, axis=1, inplace=True)
train_x, test_x, train_y, test_y = train_test_split(airport.drop('ARR_DEL15', ax
is=1), airport['ARR_DEL15'], test_size=0.2, random_state=101)
```

In [479]:

```
# In[ ]:


train_x.shape
```

Out[479]:

```
(4515, 19)
```

In [480]:

```
# In[ ]:


test_x.shape
```

Out[480]:

```
(1129, 19)
```

In [481]:

```
# In[ ]:

gboost = GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.2, loss='deviance', max_depth=2,
              max_features=2, max_leaf_nodes=None,
              min_impurity_decrease=0.1, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.4, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=100,
              subsample=1.0, tol=0.0001, validation_fraction=0.3,
              verbose=0, warm_start=False)
gboost.fit(train_x, train_y)
```

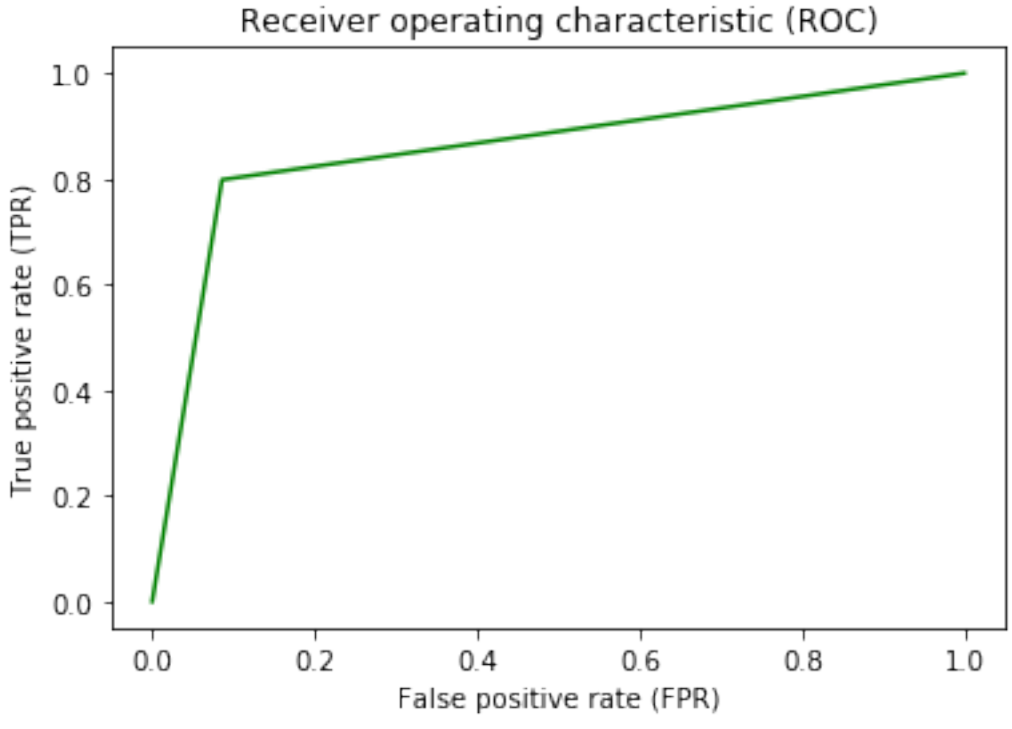
Out[481]:

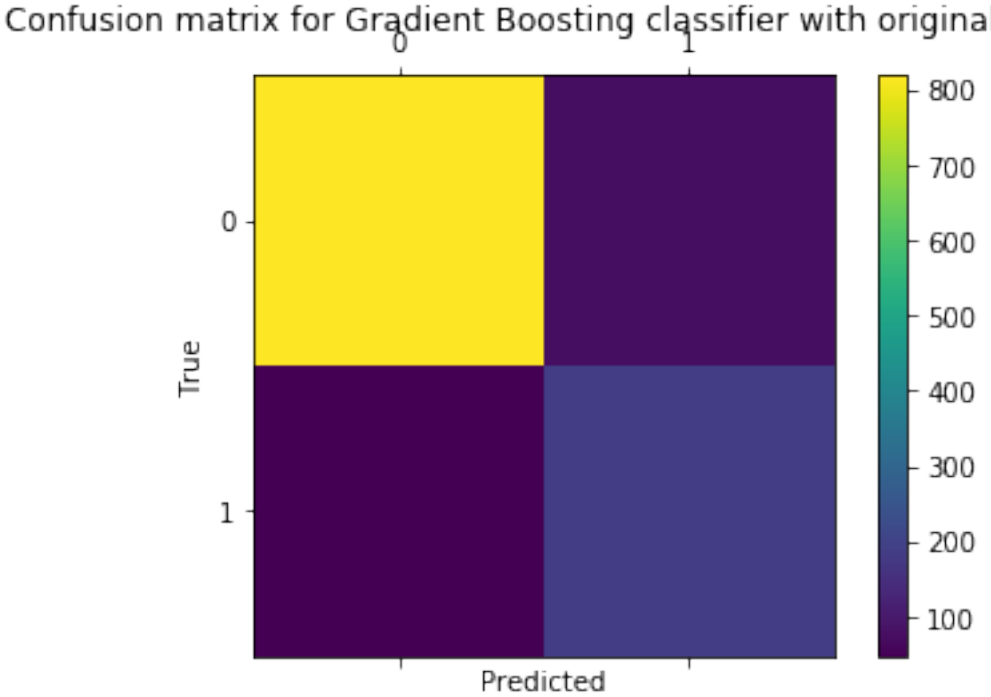
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.2, loss='deviance', max_depth=2,
              max_features=2, max_leaf_nodes=None,
              min_impurity_decrease=0.1, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.4, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=10
0,
              subsample=1.0, tol=0.0001, validation_fraction=0.3,
              verbose=0, warm_start=False)
```

In [482]:

```
# In[ ]:


y_gboost_pred = gboost.predict(test_x)

labels = [0, 1]
cm = confusion_matrix(test_y, y_gboost_pred,labels)

gboost_accuracy = str(np.round(100*float(cm[0][0]+cm[1][1])/float((cm[0][0]+cm[1
][1] + cm[1][0] + cm[0][1])),2))
gboost_recall = str(np.round(100*float((cm[1][1]))/float((cm[1][0]+cm[1][1])),2)
)

print('Accuracy: ' + gboost_accuracy +'%')
print('Recall: ' + gboost_recall +'%')
print('Confusion matrix:')
print(cm)

fpr, tpr, _ = roc_curve(test_y, y_gboost_pred)
auc = np.trapz(fpr,tpr)
print('Area under the ROC curve: ' + str(auc))

fig = plt.figure(1)
plt.plot(fpr,tpr,color='green')
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')
plt.title('Receiver operating characteristic (ROC)')

fig = plt.figure(2)
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix for Gradient Boosting classifier with original data'
)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Accuracy: 89.02%
Recall: 79.83%
Confusion matrix:
[[819  77]
 [ 47 186]]
Area under the ROC curve: 0.14382711909871243
```





In [779]:

```
# In[ ]:


dtree = DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=3, max_leaf_nodes=20,
            min_impurity_decrease=0.00003, min_impurity_split=None,
            min_samples_leaf=400, min_samples_split=9,
            min_weight_fraction_leaf=0.1, presort=False, random_state=None,
            splitter='best')
dtree.fit(train_x, train_y)
```

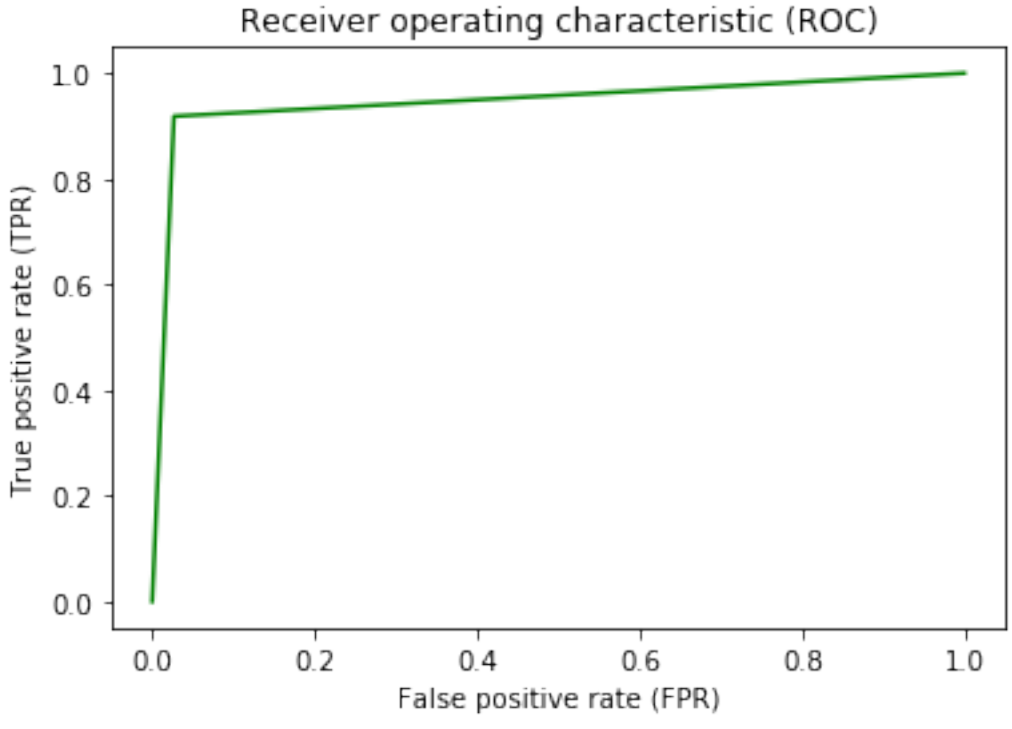
Out[779]:

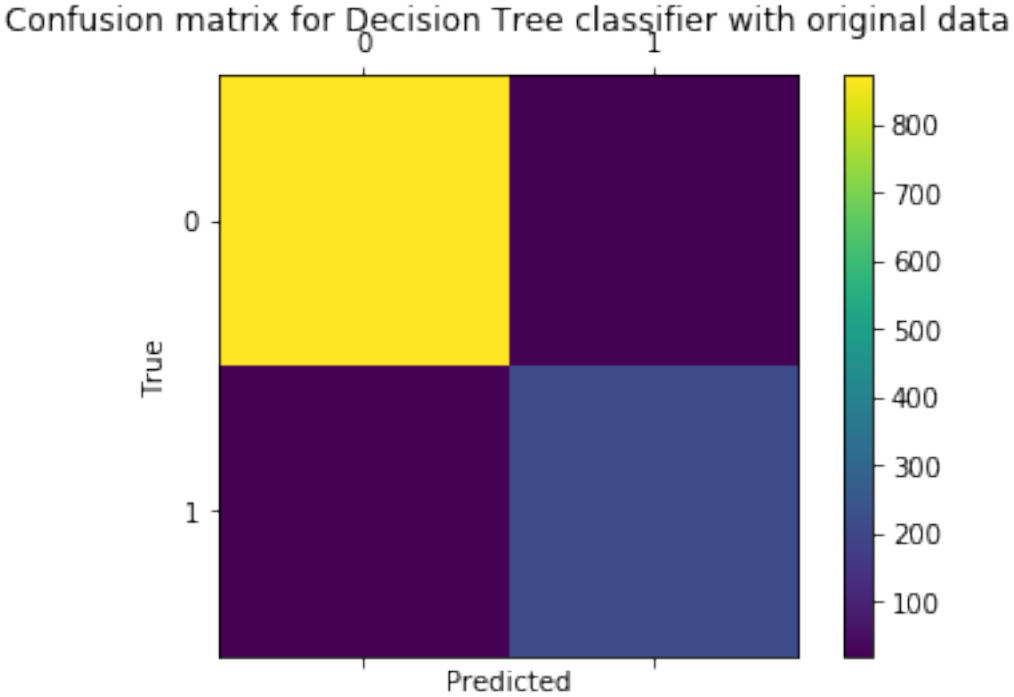
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=2,
            max_features=3, max_leaf_nodes=20, min_impurity_decrease
=3e-05,
            min_impurity_split=None, min_samples_leaf=400,
            min_samples_split=9, min_weight_fraction_leaf=0.1,
            presort=False, random_state=None, splitter='best')
```

In [780]:

```
# In[ ]:


y_dtree_pred = dtree.predict(test_x)

labels = [0, 1]
cm = confusion_matrix(test_y, y_dtree_pred,labels)

dtree_accuracy = str(np.round(100*float(cm[0][0]+cm[1][1])/float((cm[0][0]+cm[1]
[1] + cm[1][0] + cm[0][1])),2))
dtree_recall = str(np.round(100*float((cm[1][1]))/float((cm[1][0]+cm[1][1])),2))

print('Accuracy: ' + dtree_accuracy +'%')
print('Recall: ' + dtree_recall +'%')
print('Confusion matrix:')
print(cm)

fpr, tpr, _ = roc_curve(test_y, y_dtree_pred)
auc = np.trapz(fpr,tpr)
print('Area under the ROC curve: ' + str(auc))

fig = plt.figure(1)
plt.plot(fpr,tpr,color='green')
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')
plt.title('Receiver operating characteristic (ROC)')

fig = plt.figure(2)
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix for Decision Tree classifier with original data')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Accuracy: 96.19%
Recall: 91.85%
Confusion matrix:
[[872  24]
 [ 19 214]]
Area under the ROC curve: 0.05416538933169833
```





In [558]:

```
# In[ ]:


rforest = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=4, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=4,
            min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
rforest.fit(train_x, train_y)
```

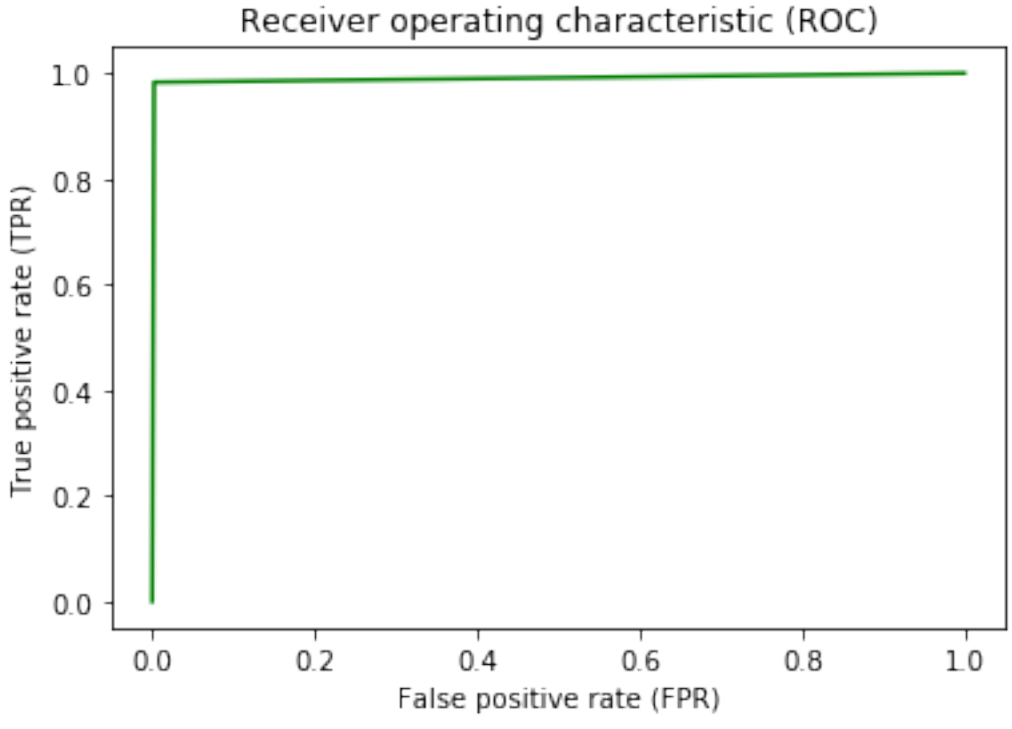
Out[558]:

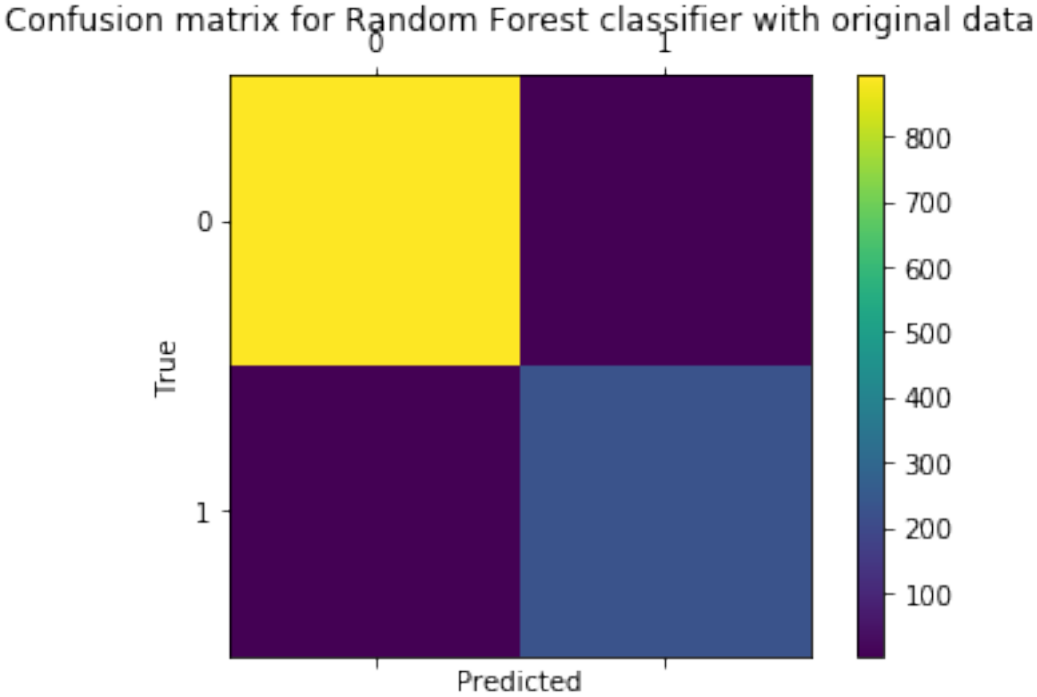
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
'gini',
            max_depth=4, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=4,
            min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=Non
e,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [559]:

```
# In[ ]:


y_rforest_pred = rforest.predict(test_x)

labels = [0, 1]
cm = confusion_matrix(test_y, y_rforest_pred,labels)

rforest_accuracy = str(np.round(100*float(cm[0][0]+cm[1][1])/float((cm[0][0]+cm[
1][1] + cm[1][0] + cm[0][1])),2))
rforest_recall = str(np.round(100*float((cm[1][1]))/float((cm[1][0]+cm[1][1])),2
))

print('Accuracy: ' + rforest_accuracy +'%')
print('Recall: ' + rforest_recall +'%')
print('Confusion matrix:')
print(cm)

fpr, tpr, _ = roc_curve(test_y, y_rforest_pred)
auc = np.trapz(fpr,tpr)
print('Area under the ROC curve: ' + str(auc))

fig = plt.figure(1)
plt.plot(fpr,tpr,color='green')
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')
plt.title('Receiver operating characteristic (ROC)')

fig = plt.figure(2)
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix for Random Forest classifier with original data')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Accuracy: 99.47%
Recall: 98.28%
Confusion matrix:
[[894   2]
 [  4 229]]
Area under the ROC curve: 0.009699762415695899
```





In [631]:

```
reg = linear_model.LogisticRegression(C=1.0, class_weight=None, dual=True, fit_i
ntercept=True,
          intercept_scaling=10, max_iter=20, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=2, solver='warn',
          tol=0.008, verbose=90, warm_start=True)
reg.fit(train_x, train_y)
```

```
[LibLinear]

/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic
.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: Conv
ergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[631]:

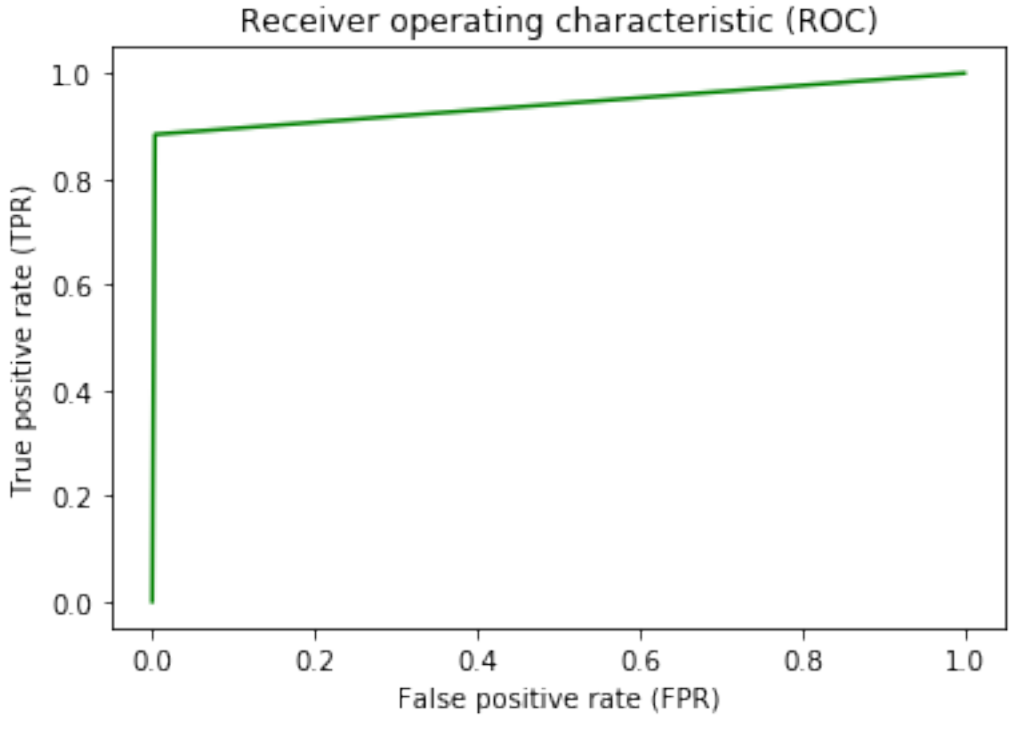
```
LogisticRegression(C=1.0, class_weight=None, dual=True, fit_intercep
t=True,
          intercept_scaling=10, max_iter=20, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=2, solver='warn',
          tol=0.008, verbose=90, warm_start=True)
```

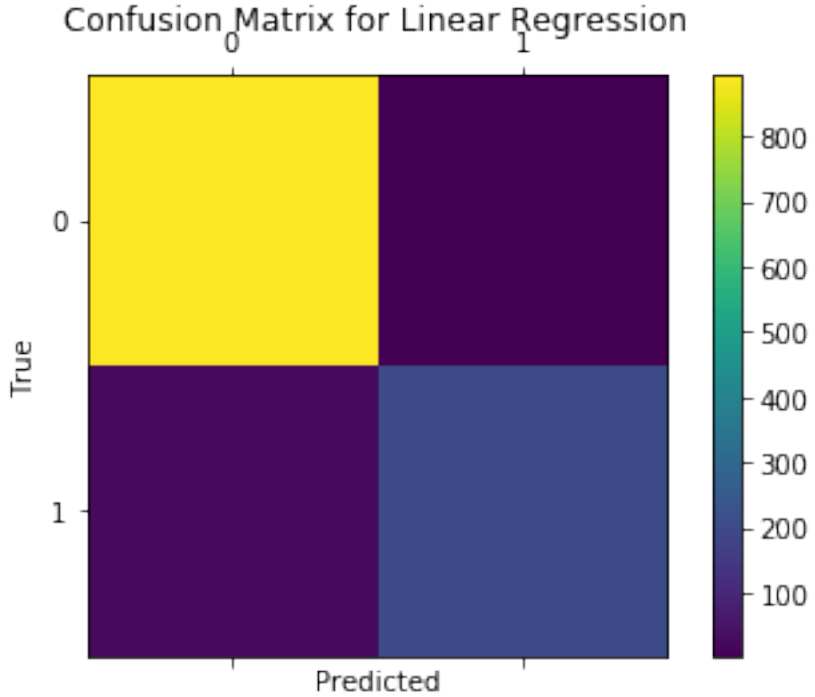
In [632]:

```
y_reg_pred = reg.predict(test_x)

labels = [0, 1]
cm = confusion_matrix(test_y, y_reg_pred,labels)

reg_accuracy = str(np.round(100*float(cm[0][0]+cm[1][1])/float((cm[0][0]+cm[1][1
] + cm[1][0] + cm[0][1])),2))
reg_recall = str(np.round(100*float((cm[1][1]))/float((cm[1][0]+cm[1][1])),2))

print('Accuracy: ' + reg_accuracy +'%')
print('Recall: ' + reg_recall +'%')
print('Confusion matrix:')
print(cm)

fpr, tpr, _ = roc_curve(test_y, y_reg_pred)
auc = np.trapz(fpr,tpr)
print('Area under the ROC curve: ' + str(auc))

fig = plt.figure(1)
plt.plot(fpr,tpr,color='green')
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')
plt.title('Receiver operating characteristic (ROC)')
```

```
fig = plt.figure(2)
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion Matrix for Linear Regression')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Accuracy: 97.34%
Recall: 88.41%
Confusion matrix:
[[893   3]
 [ 27 206]]
Area under the ROC curve: 0.059614021305947 27
```





In [ ]:

In [ ]:

In [ ]: