Dantinne_Alex MSDS_422 Assignment #1

January 24, 2021

[54]:
```
# What do you think this does?
    # It outputs more than one output
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[55]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import os
import pickle
import pandas_profiling
from pandas_profiling import ProfileReport
```

[56]:
```
os.listdir()
```

[56]:
```
['.ipynb_checkpoints',
 '2021 Utility Contract Schedule.pdf',
 'assign-1-radon-data.pickle',
 'Assignment-1-Guide-v5.pdf',
 'Dantinne_Alex MSDS_422 Assignment #1.ipynb',
 'Sync-1-MSDS-422-58-Fall-2020-v1.ipynb',
 'Untitled.ipynb']
```

[57]:
```
radonDF=pd.read_pickle('assign-1-radon-data.pickle')
radonDF.head()
radonDF.tail()
```

[57]:
```
    Code State            County  Lung Cancer Mortality  Radon  Obesity  \
0   1001    AL    Autauga_County                97.0293    1.5     31.3
1  13103    GA  Effingham_County                94.4043    0.5     31.1
2  13217    GA     Newton_County                91.8648    1.3     32.1
3  13225    GA      Peach_County                93.6161    1.6     30.1
4  21077    KY   Gallatin_County               141.4099    0.6     30.9
```

```
   Age Over 65  Currently Smoke  Ever Smoke  Median HH Income  Mort Rank  \
0         10.2             26.4       48.60             56.58       2523
1          8.0             26.6       49.65             63.26       2409
2          9.9             27.4       49.95             51.18       2281
3          9.8             27.5       47.55             41.73       2366
4         10.3             27.9       54.60             47.68       2878

   Radon Rank
0      1113.5
1       270.5
2       974.5
3      1184.0
4       364.5
```

[57]:
```
       Code State          County  Lung Cancer Mortality  Radon  Obesity  \
2876  37135    NC   Orange_County                77.8595    2.0     22.2
2877  42029    PA  Chester_County                71.4891    9.9     22.1
2878  49027    UT  Millard_County                27.1582    0.7     20.9
2879  49029    UT   Morgan_County                32.9497    3.7     21.2
2880  49051    UT  Wasatch_County                30.8043    3.6     22.5

      Age Over 65  Currently Smoke  Ever Smoke  Median HH Income  Mort Rank  \
2876          8.4             15.4       41.80             61.57       1346
2877         11.7             17.6       43.10             90.56        932
2878         12.4             17.4       40.30             52.21          9
2879          8.7             12.2       31.50             81.36         23
2880          8.4             15.6       38.45             75.11         16

      Radon Rank
2876      1410.5
2877      2781.0
2878       460.5
2879      2079.5
2880      2051.0
```

[58]:
```
radonDF.columns
radonDF.dtypes
radonDF.shape
```

[58]:
```
Index(['Code', 'State', 'County', 'Lung Cancer Mortality', 'Radon', 'Obesity',
       'Age Over 65', 'Currently Smoke', 'Ever Smoke', 'Median HH Income',
       'Mort Rank', 'Radon Rank'],
      dtype='object')
```

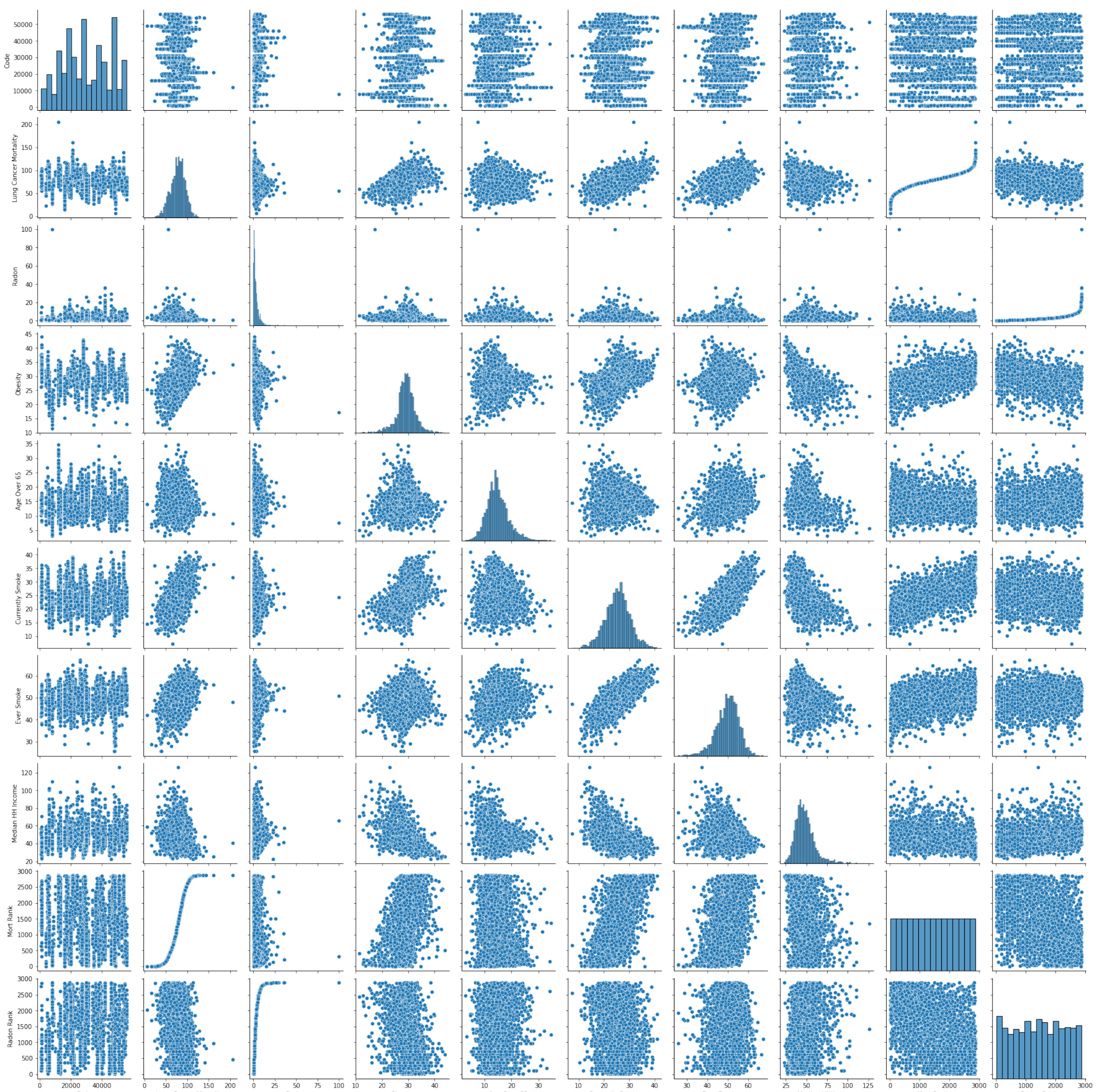
[58]:
```
Code                        int64
State                      object
County                     object
```

```
Lung Cancer Mortality    float64
Radon                    float64
Obesity                  float64
Age Over 65              float64
Currently Smoke          float64
Ever Smoke               float64
Median HH Income         float64
Mort Rank                  int64
Radon Rank               float64
dtype: object
```

[58]:
```
(2881, 12)
```

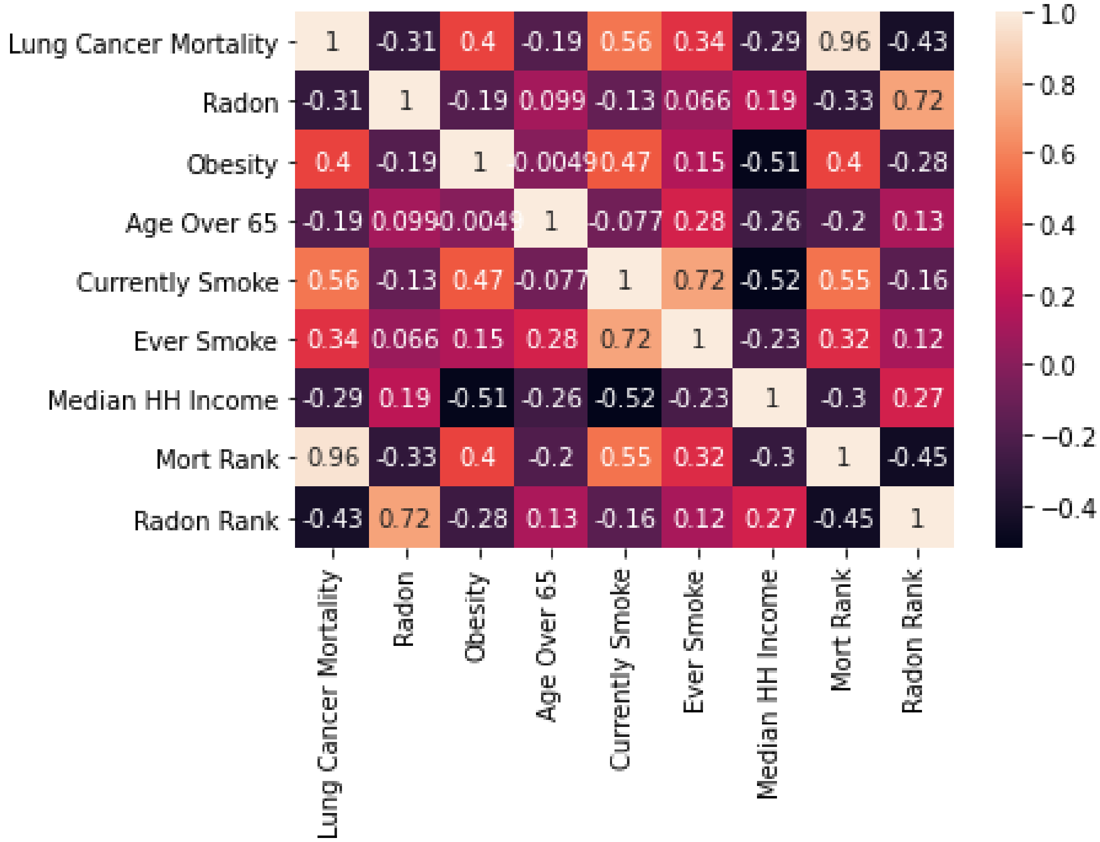
[59]:
```
sns.pairplot(radonDF)
```

[59]:
```
<seaborn.axisgrid.PairGrid at 0x254e3613a08>
```



[77]:
```python
corrMatrix = radonDF.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```

[77]: <AxesSubplot:>



[60]:
```python
radonProfile=ProfileReport(radonDF,'radon data',explorative=True)
```

[61]:
```python
radonProfile.to_notebook_iframe()
```

```
HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=20.0), HTML(value=
```

```
HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0, max=1.0), HTMI
```

```
HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0), HTML(value='')))
```

```
<IPython.core.display.HTML object>
```

[62]:
```
radonDF.Code=radonDF.Code.astype('str',copy=True)
radonDF.dtypes
```

[62]:
```
Code                       object
State                      object
County                     object
Lung Cancer Mortality     float64
Radon                     float64
Obesity                   float64
Age Over 65               float64
Currently Smoke           float64
Ever Smoke                float64
Median HH Income          float64
Mort Rank                   int64
Radon Rank                float64
dtype: object
```

[63]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

[64]:
```
y=radonDF["Lung Cancer Mortality"].to_numpy(copy=True)
X=radonDF.iloc[:,5].to_numpy(copy=True).reshape(-1,1)

y.shape
X.shape
```

[64]:
```
(2881,)
```

[64]:
```
(2881, 1)
```

[65]:
```
np.random.seed(66)
```

[66]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.20,␣
↪random_state=77)
```

[67]:
```
regMod=LinearRegression()
regMod.fit(Xtrain, ytrain)
```

[67]:
```
LinearRegression()
```

[68]:
```
print(f'Model intercept estimate: {regMod.intercept_:5.3f}')
print(f'Model estimated coefficients: {regMod.coef_[0]:5.3f}')
```

```
Model intercept estimate: 19.898
Model estimated coefficients: 2.008
```

[69]:
```
ytestPred = regMod.predict(Xtest)
```

[70]:
```
trainR2=regMod.score(Xtrain,ytrain)
testR2=r2_score(ytest,ytestPred)
```

[71]:
```
print(f'Training R\u00b2 = {trainR2}, Test R\u00b2 = {testR2:0.2f}')
```

```
Training R² = 0.17618491147003945, Test R² = 0.09
```

[87]:
```
from sklearn.linear_model import Ridge
regRid = Ridge()
regRid.fit(Xtrain, ytrain)
```

[87]:
```
Ridge()
```

[88]:
```
print(f'Model intercept estimate: {regRid.intercept_:5.3f}')
print(f'Model estimated coefficients: {regRid.coef_[0]:5.3f}')
```

```
Model intercept estimate: 19.900
Model estimated coefficients: 2.008
```

[91]:
```
ytestPred = regRid.predict(Xtest)
trainR3=regRid.score(Xtrain,ytrain)
testR3=r2_score(ytest,ytestPred)
print(f'Training R\u00b2 = {trainR3}, Test R\u00b2 = {testR3:0.2f}')
```

```
Training R² = 0.17618491129693892, Test R² = 0.09
```

[95]:
```
from sklearn.linear_model import Lasso
regLas = Lasso()
regLas.fit(Xtrain, ytrain)
```

[95]:
```
Lasso()
```

[96]:
```
print(f'Model intercept estimate: {regLas.intercept_:5.3f}')
print(f'Model estimated coefficients: {regLas.coef_[0]:5.3f}')
```

```
Model intercept estimate: 21.990
Model estimated coefficients: 1.935
```

[97]:
```
ytestPred = regLas.predict(Xtest)
trainR4=regLas.score(Xtrain,ytrain)
testR4=r2_score(ytest,ytestPred)
print(f'Training R\u00b2 = {trainR4}, Test R\u00b2 = {testR4:0.2f}')
```

```
Training R² = 0.17595691492805732, Test R² = 0.10
```

[98]:
```
from sklearn.linear_model import ElasticNet
regEla = ElasticNet()
regEla.fit(Xtrain, ytrain)
```

[98]:
```
ElasticNet()
```

[100]:
```
print(f'Model intercept estimate: {regEla.intercept_:5.3f}')
print(f'Model estimated coefficients: {regEla.coef_[0]:5.3f}')
```

```
Model intercept estimate: 22.934
Model estimated coefficients: 1.903
```

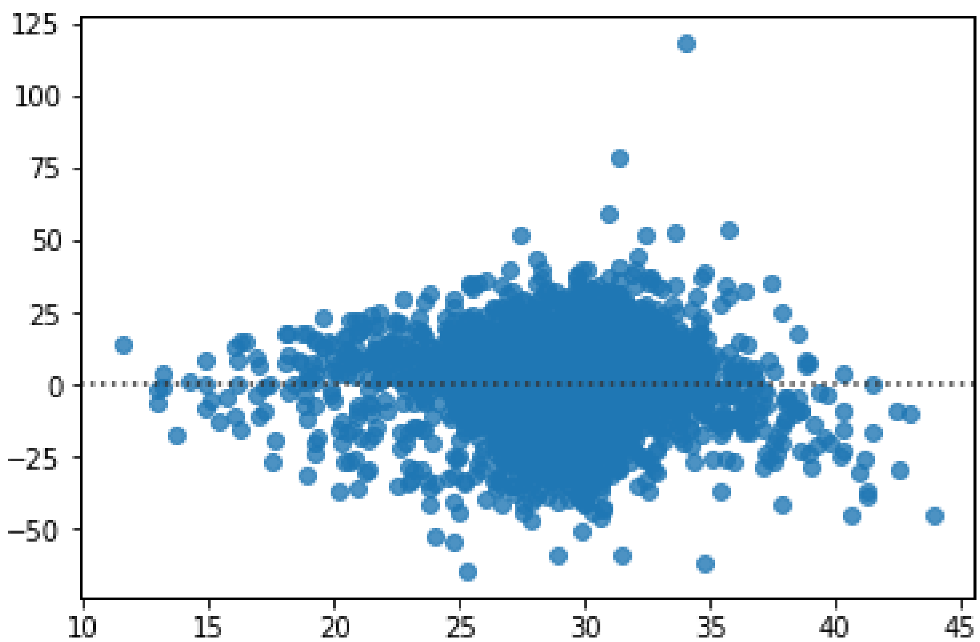
[101]:
```
ytestPred = regEla.predict(Xtest)
trainR5=regEla.score(Xtrain,ytrain)
testR5=r2_score(ytest,ytestPred)
print(f'Training R\u00b2 = {trainR5}, Test R\u00b2 = {testR5:0.2f}')
```

```
Training R² = 0.1757046231720456, Test R² = 0.10
```

The training R^2 and test r^2 values of each regression are similar In each, the training R^2 and test R^2 differ.

[92]:
```
import seaborn as sns
sns.residplot(X,y)
```

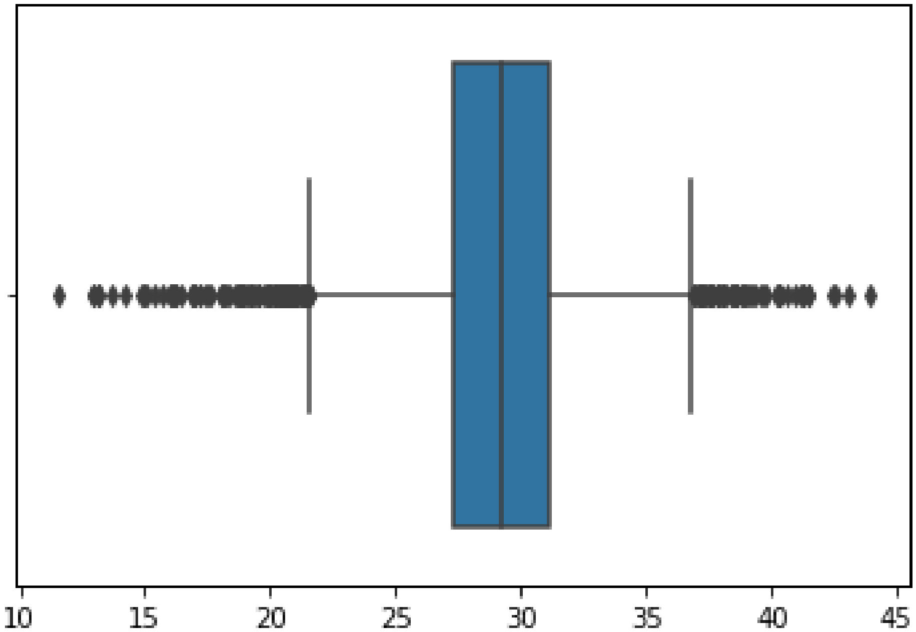
[92]:
```
<AxesSubplot:>
```



[73]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

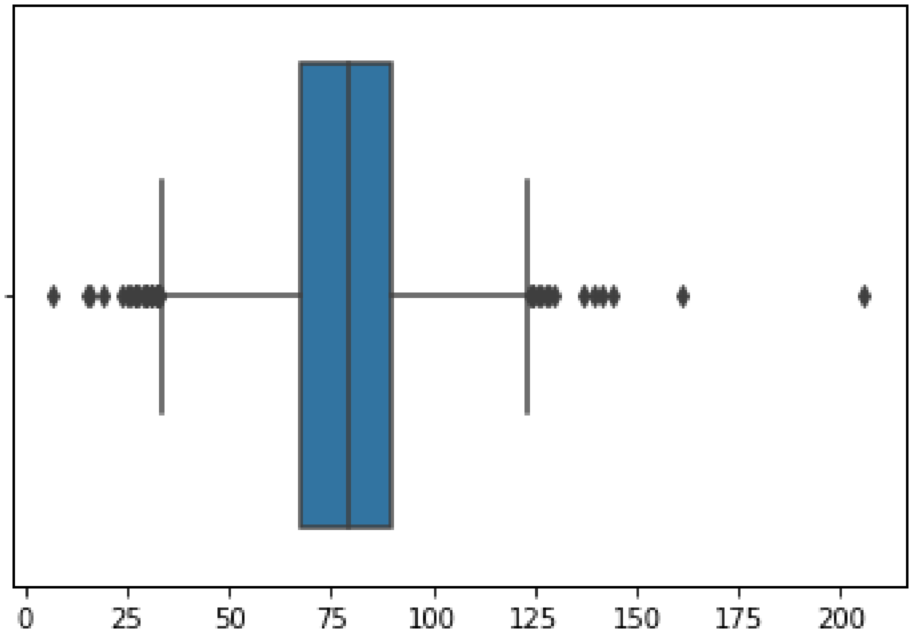
[74]:
```
sns.boxplot(X)
```

[74]: <AxesSubplot:>



[75]:
```
sns.boxplot(y)
```

[75]: <AxesSubplot:>



[25]:
```
#There appear to be outliers in x & y which may effect the linear regression.
```

[26]:
```
radonDF['HiRisk']=np.where(radonDF['Lung Cancer Mortality']>79.47,1,0)
```

[27]:
```
radonDF.columns
```

[27]:
```
Index(['Code', 'State', 'County', 'Lung Cancer Mortality', 'Radon', 'Obesity',
       'Age Over 65', 'Currently Smoke', 'Ever Smoke', 'Median HH Income',
       'Mort Rank', 'Radon Rank', 'HiRisk'],
      dtype='object')
```

[28]:
```
radonDF2=radonDF.iloc[:,:12].copy()
radonDF2['HiRisk']=radonDF.HiRisk.copy()
radonDF2.columns
```

[28]:
```
Index(['Code', 'State', 'County', 'Lung Cancer Mortality', 'Radon', 'Obesity',
       'Age Over 65', 'Currently Smoke', 'Ever Smoke', 'Median HH Income',
       'Mort Rank', 'Radon Rank', 'HiRisk'],
      dtype='object')
```

[29]:
```
misser=radonDF2.isna().sum(axis=1)
radonDF2=radonDF2.loc[misser==0,:].copy()
radonDF2.columns
```

[29]:
```
Index(['Code', 'State', 'County', 'Lung Cancer Mortality', 'Radon', 'Obesity',
       'Age Over 65', 'Currently Smoke', 'Ever Smoke', 'Median HH Income',
```

```
       'Mort Rank', 'Radon Rank', 'HiRisk'],
      dtype='object')
```

[30]:
```
y=radonDF2.HiRisk.to_numpy(copy=True)
X=radonDF2[['Obesity','Ever Smoke']].to_numpy(copy=True)
```

[31]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score,␣
 ↪precision_score, recall_score
from sklearn.base import clone
```

[32]:
```
kf=StratifiedKFold(n_splits=10, shuffle=True, random_state=55)
```

[33]:
```
from sklearn.linear_model import LogisticRegression
logReg=LogisticRegression(penalty="l2",n_jobs=1,verbose=0)

resList=[]
for train, test in kf.split(X,y):
    logR=clone(logReg).fit(X[train],y[train])
    trainPred=logR.predict(X[train])
    trainProb=logR.predict_proba(X[train])[:,1]
    testPred = logR.predict(X[test])
    testProb=logR.predict_proba(X[test])[:,1]

    accTrain=accuracy_score(trainPred, y[train])
    accTest=accuracy_score(testPred,y[test])
    aucTrain=roc_auc_score(y[train],trainProb)
    aucTest=roc_auc_score(y[test],testProb)

    resList.append(
    {'trainAcc':accTrain,
    'testAcc':accTest,
    'trainAUC':aucTrain,
    'testAUC':aucTest})
```

[34]:
```
logResDF=pd.DataFrame(resList)
```

[35]:
```
print('Summary Across Folds')
logResDF.describe()
```

```
Summary Across Folds
```

[35]:

	trainAcc	testAcc	trainAUC	testAUC
count	10.000000	10.000000	10.000000	10.000000
mean	0.665818	0.665278	0.740317	0.740457
std	0.002733	0.024067	0.002869	0.025939
min	0.662809	0.628472	0.735070	0.704009
25%	0.663291	0.647569	0.738712	0.723368

```
50%      0.665123   0.661458   0.740437   0.737929
75%      0.668306   0.687500   0.742564   0.754184
max      0.669753   0.701389   0.744158   0.788395
```

[36]:
```python
from sklearn.model_selection import cross_validate
from sklearn.metrics import SCORERS
```

[37]:
```python
SCORERS.keys()
```

[37]:
```
dict_keys(['explained_variance', 'r2', 'max_error', 'neg_median_absolute_error',
'neg_mean_absolute_error', 'neg_mean_absolute_percentage_error',
'neg_mean_squared_error', 'neg_mean_squared_log_error',
'neg_root_mean_squared_error', 'neg_mean_poisson_deviance',
'neg_mean_gamma_deviance', 'accuracy', 'top_k_accuracy', 'roc_auc',
'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted',
'balanced_accuracy', 'average_precision', 'neg_log_loss', 'neg_brier_score',
'adjusted_rand_score', 'rand_score', 'homogeneity_score', 'completeness_score',
'v_measure_score', 'mutual_info_score', 'adjusted_mutual_info_score',
'normalized_mutual_info_score', 'fowlkes_mallows_score', 'precision',
'precision_macro', 'precision_micro', 'precision_samples', 'precision_weighted',
'recall', 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted',
'f1', 'f1_macro', 'f1_micro', 'f1_samples', 'f1_weighted', 'jaccard',
'jaccard_macro', 'jaccard_micro', 'jaccard_samples', 'jaccard_weighted'])
```

[38]:
```python
logRegCV=cross_validate(clone(logReg),X,y,cv=10,return_train_score=True,scoring␣
↪=('accuracy','roc_auc'))
```

[39]:
```python
print(f'Summary Across Folds')
pd.DataFrame(logRegCV).iloc[:,2:].describe()
```

```
Summary Across Folds
```

[39]:

	test_accuracy	train_accuracy	test_roc_auc	train_roc_auc
count	10.000000	10.000000	10.000000	10.000000
mean	0.650694	0.670795	0.774647	0.742327
std	0.159590	0.020091	0.228576	0.023078
min	0.458333	0.632330	0.348881	0.700144
25%	0.526042	0.662133	0.672198	0.730016
50%	0.621528	0.669753	0.842249	0.737690
75%	0.748264	0.684703	0.950137	0.757384
max	0.979167	0.703704	0.996527	0.778845

[40]:
```python
from sklearn.model_selection import GridSearchCV

parms = {"C": [0.0001, 0.001, 0.01, 0.1, 1.0]}
```

[41]:
```python
logReg=LogisticRegression()
```

```python
logParmCV = GridSearchCV(logReg,n_jobs=1,verbose=0,
                         refit=True,
                         cv = 10,
                         scoring="accuracy",
                         return_train_score=True,
                         param_grid=parms)
logParmCV
```

[41]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(), n_jobs=1,
             param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1.0]},
             return_train_score=True, scoring='accuracy')
```

[42]:
```python
logParmFit = logParmCV.fit(X,y)
print(sorted(logParmFit.cv_results_.keys()))
```

```
['mean_fit_time', 'mean_score_time', 'mean_test_score', 'mean_train_score',
'param_C', 'params', 'rank_test_score', 'split0_test_score',
'split0_train_score', 'split1_test_score', 'split1_train_score',
'split2_test_score', 'split2_train_score', 'split3_test_score',
'split3_train_score', 'split4_test_score', 'split4_train_score',
'split5_test_score', 'split5_train_score', 'split6_test_score',
'split6_train_score', 'split7_test_score', 'split7_train_score',
'split8_test_score', 'split8_train_score', 'split9_test_score',
'split9_train_score', 'std_fit_time', 'std_score_time', 'std_test_score',
'std_train_score']
```

[43]:
```python
print(f'Summary of Grid Search')
cols=['param_C','mean_train_score','std_train_score',
      'mean_test_score','std_test_score']

pd.DataFrame(logParmFit.cv_results_)[cols].
 ↪sort_values(by='mean_test_score',ascending=False)
```

```
Summary of Grid Search
```

[43]:

	param_C	mean_train_score	std_train_score	mean_test_score	std_test_score
3	0.1000	0.670718	0.019077	0.650694	0.151400
4	1.0000	0.670795	0.019060	0.650694	0.151400
2	0.0100	0.670448	0.018487	0.650000	0.151540
1	0.0010	0.667670	0.018027	0.646875	0.155050
0	0.0001	0.657176	0.016260	0.632986	0.165877

[44]:
```python
logParmFit.best_params_
```

[44]:
```
{'C': 0.1}
```

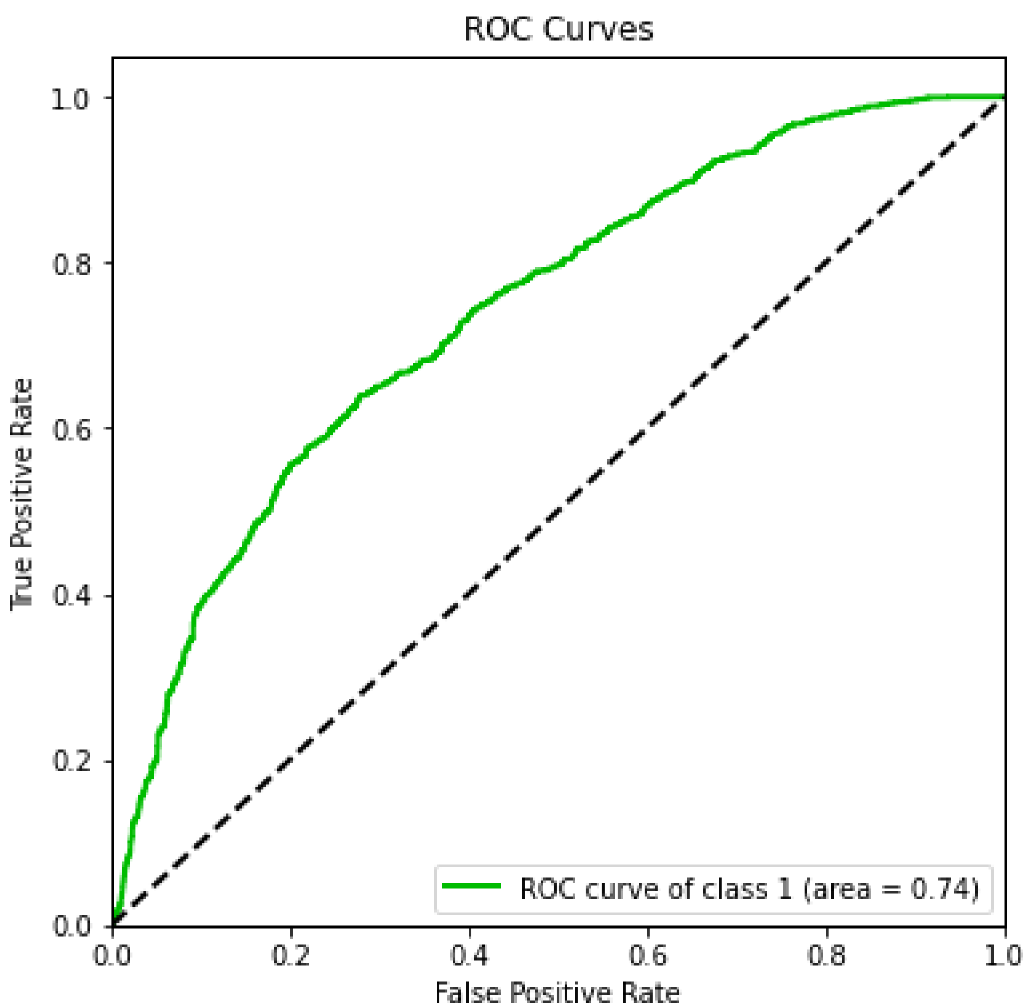
[45]:
```
import scikitplot as skplt
matplotlib.rc('axes',lw=1,edgecolor='black')
```

[46]:
```
yPredProb=logParmFit.predict_proba(X)
```

[47]:
```
yPredProb.shape
```

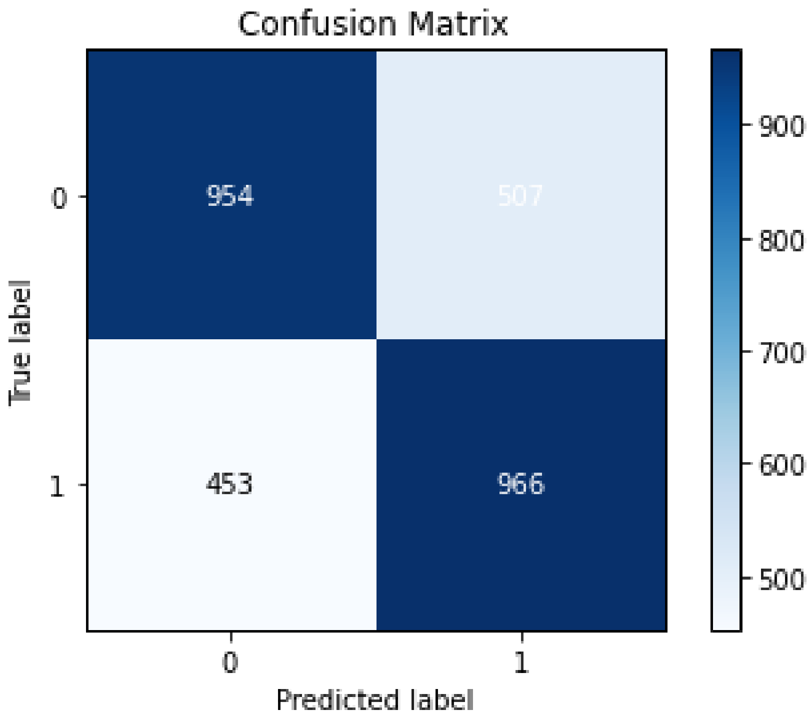
[47]:
```
(2880, 2)
```

[48]:
```
skplt.metrics.plot_roc(y, yPredProb,
                       classes_to_plot=1,
                      plot_micro=False,
                      plot_macro=False,
                      figsize=(6,6)
                      )
plt.show();
```



[49]:
```
yPred=logParmFit.predict(X)
```

[50]:
```
skplt.metrics.plot_confusion_matrix(y,yPred);
```



1. Models are valiadted using methods like hold out, leave one out cross-validation, and k fold cross-validation to ensure that the models are preforming correctly by reviewing the trained model with testing data set.
2. After review, I don't think that any of my models are "over-fit". If I did have a model over-fit, I would up size the data I trained with. Using more data could add more noise, buy may produce a better signal. Another way would be to cross-validate which would test the effectiveness of the model.
3. R^2 is calculated by the summation of squared difference of the average of the actual values by actual values. The mean squared error is calculated by the summation of the squared difference between the predicited and observed then dividing by the number of data points. The mean absolute error is calculated by the summation of the absolute difference between the predicted and true values and then dividing by the total number of data points.
4. Precision is a measurement that shows the proportion of positives that were truely positive. Precision = TP/(TP+FP). Recall is a measurement that shows the proportion of positives that were rightly positive. Recall = TP/(TP+FN). The F1 statistic is an accuracy measurement of a model. F1= (2x(precisionxrecall)/precision+recall))
5. Bias/variance trade-off is a balance between controling bias and variance errors. One way to lower variance is to increase the bias and one way to lower the bias is to increase the variance

when a model is applied to new data that hasn't been trained.

[51]:
```
import os
os.getcwd()
```

[51]:
```
'C:\\Users\\alex.dantinne\\Documents\\Northwestern\\Winter
2021\\MSDS_422\\Assignment 1'
```

[ ]:
```

```