Bank Note Analysis

Data Set Information:

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

Attribute Information:

1. variance of Wavelet Transformed image (continuous)
2. skewness of Wavelet Transformed image (continuous)
3. curtosis of Wavelet Transformed image (continuous)
4. entropy of image (continuous)
5. class (integer)

Dataset: https://www.kaggle.com/ritesaluja/bank-note-authentication-uci-data (https://www.kaggle.com/ritesaluja/bank-note-authentication-uci-data)

UCI: http://archive.ics.uci.edu/ml/datasets/banknote+authentication (http://archive.ics.uci.edu/ml/datasets/banknote+authentication)

Can we classify banknote as fake or genuine?

In [65]:
```
from pyforest import*
```

In [66]:
```
lazy_imports()
```

Out[66]:
```
['import os',
 'import plotly.express as px',
 'import altair as alt',
 'import plotly.graph_objs as go',
 'import keras',
 'import bokeh',
 'import datetime as dt',
 'from sklearn.preprocessing import OneHotEncoder',
 'from pathlib import Path',
 'import sklearn',
 'import numpy as np',
 'from sklearn.manifold import TSNE',
 'from sklearn.ensemble import GradientBoostingRegressor',
 'import re',
 'import plotly as py',
 'from sklearn.model_selection import train_test_split',
 'import statistics',
 'from sklearn.ensemble import RandomForestClassifier',
 'import nltk',
 'from sklearn.ensemble import RandomForestRegressor',
 'import tensorflow as tf',
 'from sklearn import svm',
 'import pickle',
 'import dash',
 'import matplotlib as mpl',
 'from dask import dataframe as dd',
 'from sklearn.feature_extraction.text import TfidfVectorizer',
 'import tqdm',
 'from pyspark import SparkContext',
 'from sklearn.ensemble import GradientBoostingClassifier',
 'import spacy',
 'import gensim',
 'import pydot',
 'from openpyxl import load_workbook',
 'import glob',
 'import sys']
```

In [67]:
```
df=pd.read_csv('BankNote_Authentication.csv')
```

In [68]:
```
df.head()
```

Out[68]:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [69]:
```
df.shape
```

Out[69]:
```
(1372, 5)
```

In [70]:
```
df['class'].value_counts()
```

Out[70]:
```
0    762
1    610
Name: class, dtype: int64
```

In [71]:
```
df.isna().sum()
```

Out[71]:
```
variance    0
skewness    0
curtosis    0
entropy     0
class       0
dtype: int64
```

In [72]:
```
df.dtypes
```

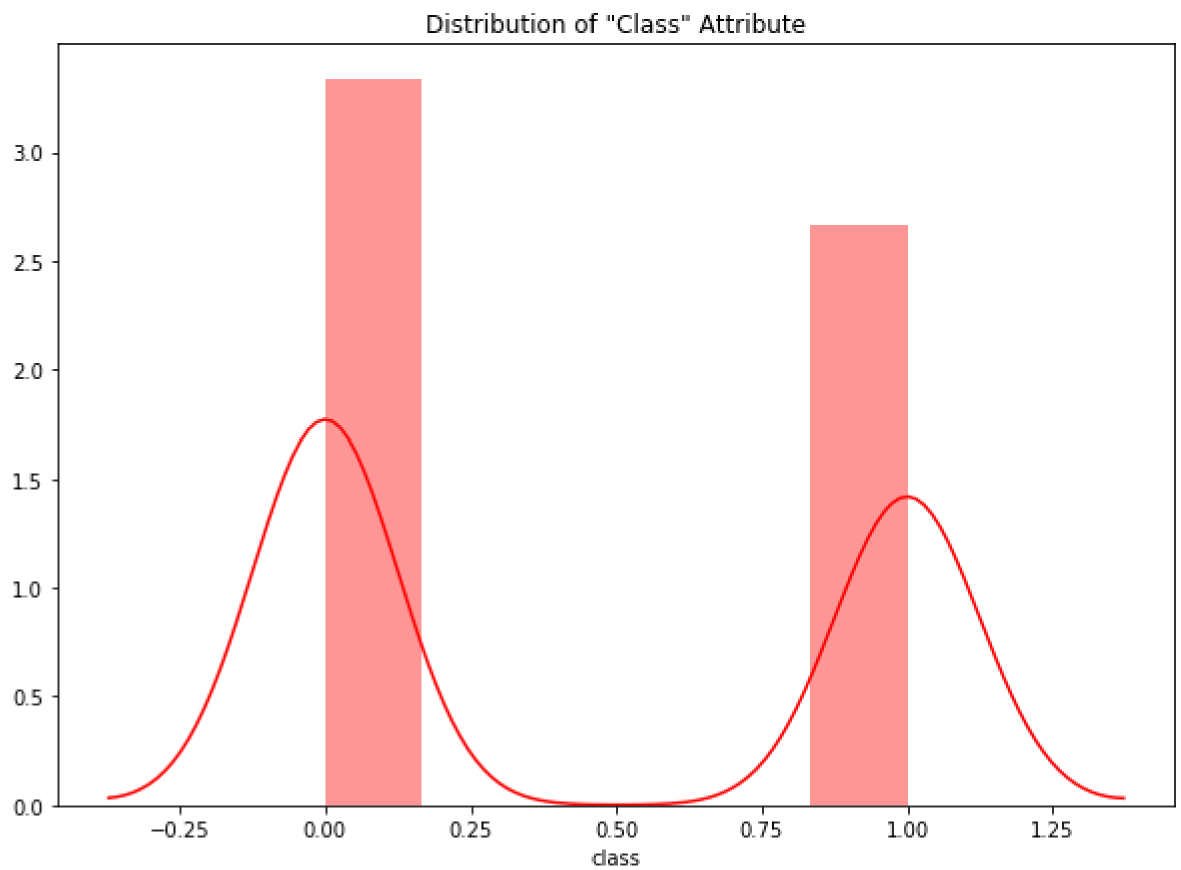
Out[72]:
```
variance    float64
skewness    float64
curtosis    float64
entropy     float64
class         int64
dtype: object
```

Visualizations

Univariate Data Analysis

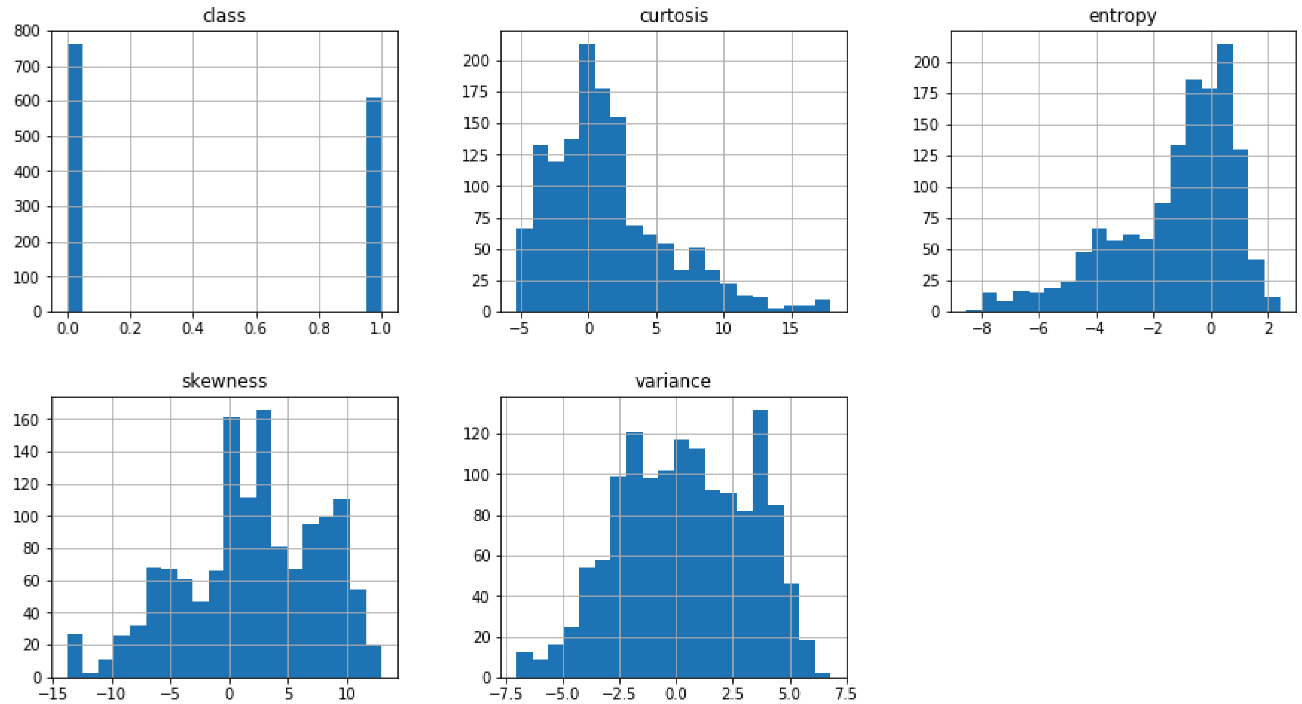
In [73]:
```
plt.figure(figsize=(10,7))
plt.title('Distribution of "Class" Attribute')

sns.distplot(df['class'],color='red')
```

Out[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x125c39358>
```



Multivariate Data Analysis

In [74]:
```
df.hist(bins=20, figsize=(15,8),layout=(2,3)); #Histogram of all the attributes
```

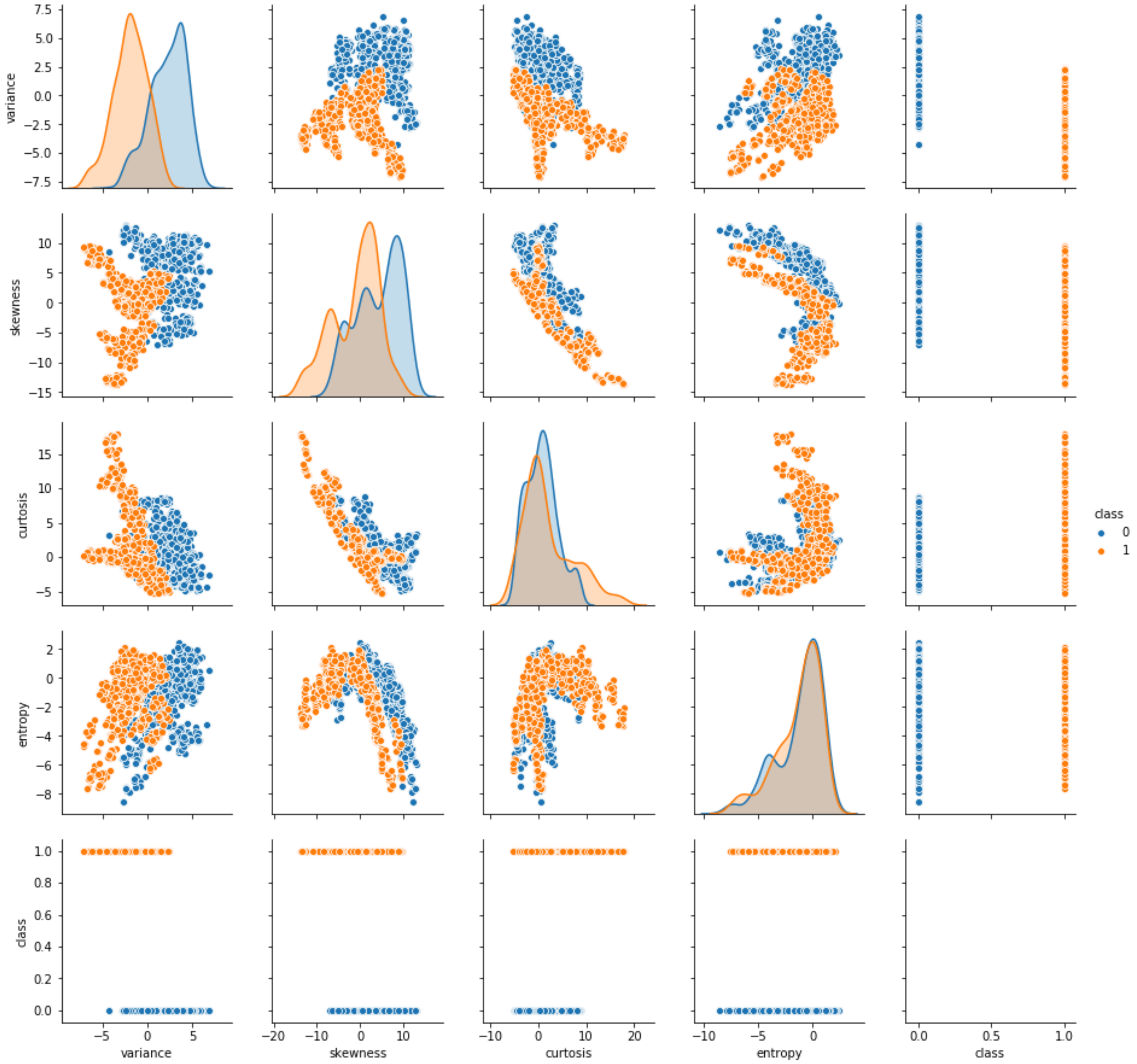


In [75]:
```
# col_names = df.drop('class', axis = 1).columns.tolist()

# plt.figure(figsize = (10,3))
# i = 0
# for col in col_names:
#     plt.subplot(1,4,i+1)
#     plt.grid(True, alpha =0.5)
#     sns.kdeplot(df[col][df['class'] ==0], label = 'Fake note')
#     sns.kdeplot(df[col][df['class'] ==1], label = 'Original note')
#     plt.title('Class vs ' + col)
#     plt.tight_layout()
#     i+=1
# plt.show()
```

In [76]:
```
import warnings

warnings.filterwarnings('ignore')
```

```
In [77]: sns.pairplot(df, hue="class")
```

Out[77]: <seaborn.axisgrid.PairGrid at 0x125e077b8>



Preparing Our Data To Build Our Model

```
In [78]: df.head()
```

Out[78]:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

```
In [79]: #defining features and target variable
X = df.drop(['class'],axis=1)
y = df['class']
```

In [80]:
```
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, rando
m_state=1)
```

Scaling Our Data

In [81]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [82]:
```
X_train
```

Out[82]:
```
array([[-1.58438248,  0.1072115 , -0.14276339,  0.03334576],
       [-1.08829139, -2.53123321,  2.67783284, -0.35092979],
       [ 1.13672843, -0.15348755, -0.16820608,  0.86368769],
       ...,
       [-1.6900361 ,  0.72314447, -0.19588896, -2.05114485],
       [ 0.57766241,  0.02698182,  0.1851622 ,  0.52080477],
       [-0.9644631 ,  0.30908695, -0.49734797, -0.03521515]])
```

In [83]:
```
y_train.head()
```

Out[83]:
```
1226    1
1085    1
148     0
1178    1
478     0
Name: class, dtype: int64
```

Logistic Regression

In [84]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

classifier=LogisticRegression(solver='liblinear',random_state=1)
classifier.fit(X_train,y_train)

accuracies=cross_val_score(estimator=classifier,X=X_train,y=y_train,cv=10) # C
V: Determines the cross-validation splitting strategy (How many folds, default
is 5-folds) Evaluate a score by cross-validation. estimator: object to use to f
it the data.
print("Accuracies:\n",accuracies)

y_test_pred=classifier.predict(X_test)


print("Mean Accuracy: ",accuracies.mean())
```

```
Accuracies:
 [0.98181818 0.99090909 0.98181818 0.99090909 0.99090909 0.99090909
 0.96363636 0.99082569 0.97247706 0.98165138]
Mean Accuracy:  0.9835863219349459
```

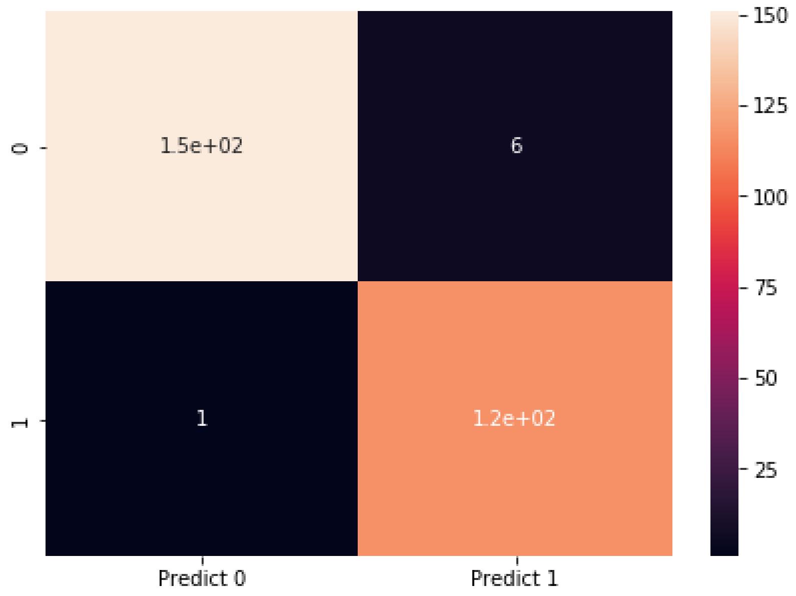
In [85]:
```
accuracy_score(y_test,y_test_pred)
```

Out[85]:
```
0.9745454545454545
```

In [86]:
```python
from sklearn import metrics

print("Confusion Matrix For Logistic Regression")
cm=metrics.confusion_matrix(y_test, y_test_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                     columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For Logistic Regression
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x125f5f320>



Support Vector Machine

In [87]:
```python
from sklearn.svm import SVC

svm_classifier=SVC(kernel='linear')
svm_classifier.fit(X_train,y_train)

svm_accuracies=cross_val_score(estimator=svm_classifier,X=X_train,y=y_train,cv=
10)
print("Accuracies:\n",svm_accuracies)
```

```
Accuracies:
 [0.99090909 0.99090909 0.99090909 0.99090909 0.99090909 0.99090909
 0.96363636 0.99082569 0.97247706 0.98165138]
```

In [88]:
```python
svm_pred=svm_classifier.predict(X_test)

print("Mean Accuracy: ",svm_accuracies.mean())
```

```
Mean Accuracy:  0.9854045037531277
```

In [89]:
```python
accuracy_score(y_test,svm_pred)
```

Out[89]: 0.9818181818181818

In [90]:
```
print("Confusion Matrix For svm_pred")
cm=metrics.confusion_matrix(y_test,svm_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                     columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For svm_pred
```

Out[90]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x125c80c18>
```

Support Vector Machine (rbf)

Kernels in SVM classification refer to the function that is responsible for defining the decision boundaries between the classes. Apart from the classic linear kernel which assumes that the different classes are separated by a straight line, a RBF (radial basis function) kernel is used when the boundaries are hypothesized to be curve-shaped.

RBF kernel uses two main parameters, gamma and C that are related to:

1. the decision region (how spread the region is), and
2. the penalty for misclassifying a data point

respectively

In [91]:
```
from sklearn.svm import SVC

svm_rbf_classifier=SVC(kernel='rbf',gamma='auto')
svm_rbf_classifier.fit(X_train,y_train)

svm_rbf_accuracies=cross_val_score(estimator=svm_rbf_classifier,X=X_test,y=y_te
st,cv=10)
print("Accuracies:\n",svm_rbf_accuracies)
print("Mean Accuracy: ",svm_rbf_accuracies.mean())
```

```
Accuracies:
 [1.         0.96428571 1.         1.         1.         1.
 0.96296296 1.         1.         1.        ]
Mean Accuracy:  0.9927248677248677
```

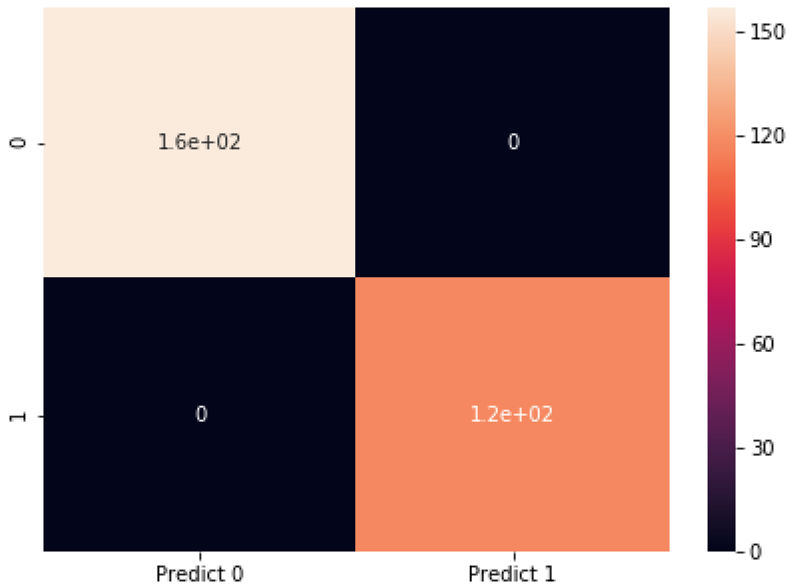
In [92]:
```
svm_rbf_pred=svm_rbf_classifier.predict(X_test)

accuracy_score(y_test,svm_rbf_pred)
```

Out[92]:
```
1.0
```

In [93]:
```
print("Confusion Matrix For svm_rbf")
cm=metrics.confusion_matrix(y_test,svm_rbf_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                     columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For svm_rbf
```

Out[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x126c31cc0>
```



In [ ]:

RandomForestClassifier

In [94]:
```
from sklearn.ensemble import RandomForestClassifier

rdf_classifier=RandomForestClassifier(n_estimators=50,criterion='entropy',rando
m_state=0)
rdf_classifier.fit(X_train,y_train)
rdf_accuracies=cross_val_score(estimator=rdf_classifier,X=X_test,y=y_test,cv=1
0)

print("Accuracies:\n",rdf_accuracies)
print("Mean Accuracy: ",rdf_accuracies.mean())
```

```
Accuracies:
 [0.96428571 1.         0.89285714 0.96428571 1.         0.96296296
 1.         0.96296296 1.         1.        ]
Mean Accuracy:  0.9747354497354497
```

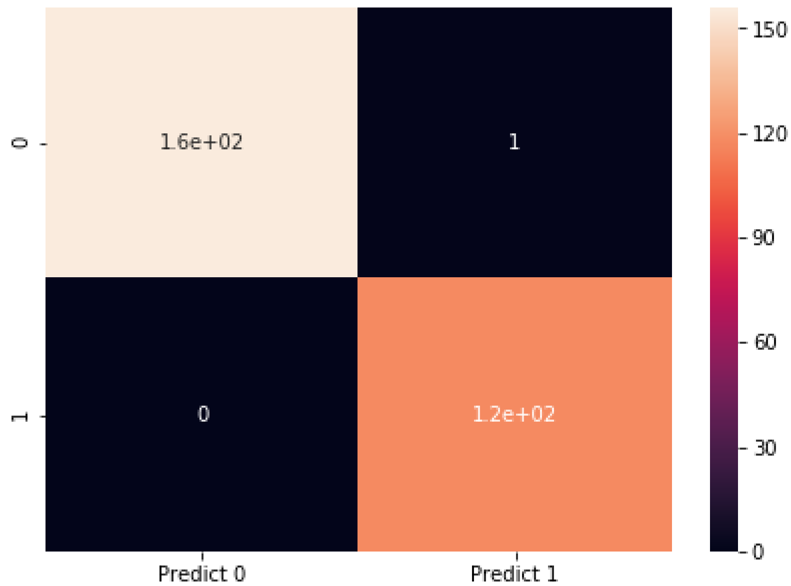
In [95]:
```
rdf_pred=rdf_classifier.predict(X_test)

accuracy_score(y_test,rdf_pred)
```

Out[95]: 0.9963636363636363

In [96]:
```
print("Confusion Matrix For Random Forest")
cm=metrics.confusion_matrix(y_test,rdf_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                  columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For Random Forest
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x126f4d160>



KNeighborsClassifier

In [97]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, GridSearchCV

param_grid = {
    'leaf_size' : [2,5,7,9,11],
    'n_neighbors' : [2,5,7,9,11],
    'p' : [1,2]
}

grid = GridSearchCV(KNeighborsClassifier(), param_grid = param_grid)
grid.fit(X_train, y_train)
```

Out[97]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'leaf_size': [2, 5, 7, 9, 11],
                         'n_neighbors': [2, 5, 7, 9, 11], 'p': [1, 2]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [98]:
```
grid.best_params_
```

Out[98]:
```
{'leaf_size': 2, 'n_neighbors': 2, 'p': 1}
```

In [99]:
```
final_KNN_Model = grid.best_estimator_
```

In [100]:
```
KNN = KNeighborsClassifier(n_neighbors=2,p=1,leaf_size=2 )
```

In [101]:
```
# Call Nearest Neighbour algorithm

KNN.fit(X_train, y_train)
```

Out[101]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=2, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=2, p=1,
                     weights='uniform')
```

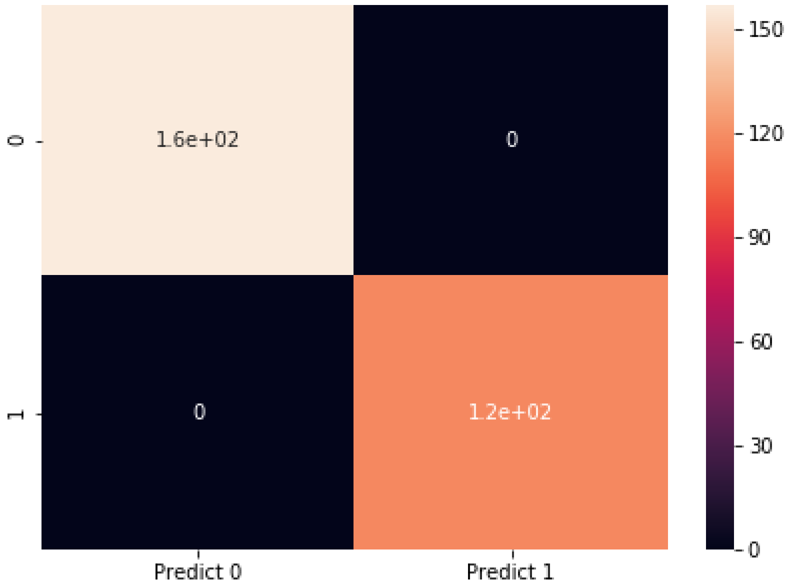
In [102]:
```
KNN_predicted = KNN.predict(X_test)

accuracy_score(y_test,KNN_predicted)
```

Out[102]:
```
1.0
```

In [103]:
```
print("Confusion Matrix For KNN")
cm=metrics.confusion_matrix(y_test,KNN_predicted, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                    columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For KNN
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12706aac8>
```



Multilayer Perceptron

[https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

In [104]:

```
from sklearn.neural_network import MLPClassifier
multi_classifier=MLPClassifier(hidden_layer_sizes=(8,4), max_iter=8000, alpha=
0.0001, solver='sgd', verbose=10,  random_state=21,tol=0.000000001)
multi_classifier.fit(X_train,y_train)
multi_accuracies=cross_val_score(estimator=multi_classifier,X=X_test,y=y_test,c
v=10)
```

```
Iteration 1, loss = 1.02065261
Iteration 2, loss = 1.00232763
Iteration 3, loss = 0.97715486
Iteration 4, loss = 0.94973488
Iteration 5, loss = 0.92231418
Iteration 6, loss = 0.89651964
Iteration 7, loss = 0.87283524
Iteration 8, loss = 0.85051410
Iteration 9, loss = 0.83056223
Iteration 10, loss = 0.81132826
Iteration 11, loss = 0.79467501
Iteration 12, loss = 0.77862224
Iteration 13, loss = 0.76455037
Iteration 14, loss = 0.75105250
Iteration 15, loss = 0.73882781
Iteration 16, loss = 0.72727541
Iteration 17, loss = 0.71657090
Iteration 18, loss = 0.70615519
Iteration 19, loss = 0.69650376
Iteration 20, loss = 0.68705241
Iteration 21, loss = 0.67795411
Iteration 22, loss = 0.66915932
Iteration 23, loss = 0.66056959
Iteration 24, loss = 0.65225336
Iteration 25, loss = 0.64391579
Iteration 26, loss = 0.63595619
Iteration 27, loss = 0.62803230
Iteration 28, loss = 0.62041858
Iteration 29, loss = 0.61270498
Iteration 30, loss = 0.60516212
Iteration 31, loss = 0.59779594
Iteration 32, loss = 0.59035692
Iteration 33, loss = 0.58320321
Iteration 34, loss = 0.57595144
Iteration 35, loss = 0.56900180
Iteration 36, loss = 0.56205103
Iteration 37, loss = 0.55529365
Iteration 38, loss = 0.54845094
Iteration 39, loss = 0.54178612
Iteration 7939, loss = 0.00267872
Iteration 7940, loss = 0.00267823
Iteration 7941, loss = 0.00267822
Iteration 7942, loss = 0.00267728
Iteration 7943, loss = 0.00267686
Iteration 7944, loss = 0.00267644
Iteration 7945, loss = 0.00267593
Iteration 7946, loss = 0.00267553
Iteration 7947, loss = 0.00267498
Iteration 7948, loss = 0.00267469
Iteration 7949, loss = 0.00267420
Iteration 7950, loss = 0.00267370
Iteration 7951, loss = 0.00267321
Iteration 7952, loss = 0.00267275
Iteration 7953, loss = 0.00267233
Iteration 7954, loss = 0.00267206
Iteration 7955, loss = 0.00267141
Iteration 7956, loss = 0.00267096
Iteration 7957, loss = 0.00267046
Iteration 7958, loss = 0.00267001
Iteration 7959, loss = 0.00266954
Iteration 7960, loss = 0.00266905
Iteration 7961, loss = 0.00266854
Iteration 7962, loss = 0.00266827
Iteration 7963, loss = 0.00266759
Iteration 7964, loss = 0.00266713
```

```
Iteration 7965, loss = 0.00266703
Iteration 7966, loss = 0.00266623
Iteration 7967, loss = 0.00266566
Iteration 7968, loss = 0.00266522
Iteration 7969, loss = 0.00266476
Iteration 7970, loss = 0.00266438
Iteration 7971, loss = 0.00266380
Iteration 7972, loss = 0.00266325
Iteration 7973, loss = 0.00266272
Iteration 7974, loss = 0.00266232
Iteration 7975, loss = 0.00266184
Iteration 7976, loss = 0.00266140
Iteration 7977, loss = 0.00266092
Iteration 7978, loss = 0.00266053
Iteration 7979, loss = 0.00266003
Iteration 7980, loss = 0.00265940
Iteration 7981, loss = 0.00265899
Iteration 7982, loss = 0.00265848
Iteration 7983, loss = 0.00265816
Iteration 7984, loss = 0.00265794
Iteration 7985, loss = 0.00265720
Iteration 7986, loss = 0.00265674
Iteration 7987, loss = 0.00265626
Iteration 7988, loss = 0.00265585
Iteration 7989, loss = 0.00265543
Iteration 7990, loss = 0.00265495
Iteration 7991, loss = 0.00265449
Iteration 7992, loss = 0.00265408
Iteration 7993, loss = 0.00265365
Iteration 7994, loss = 0.00265305
Iteration 7995, loss = 0.00265277
Iteration 7996, loss = 0.00265212
Iteration 7997, loss = 0.00265191
Iteration 7998, loss = 0.00265126
Iteration 7999, loss = 0.00265093
Iteration 8000, loss = 0.00265054
```

In [105]:

```
print("Accuracies:\n",multi_accuracies)
print("Mean Accuracy: ",multi_accuracies.mean())
```

```
Accuracies:
 [1.         1.         1.         1.         1.         1.
 0.92592593 1.         1.         1.        ]
Mean Accuracy:  0.9925925925925926
```

In [106]:

```
multi_predicted = multi_classifier.predict(X_test)
```

In [107]:

```
accuracy_score(y_test,multi_predicted)
```

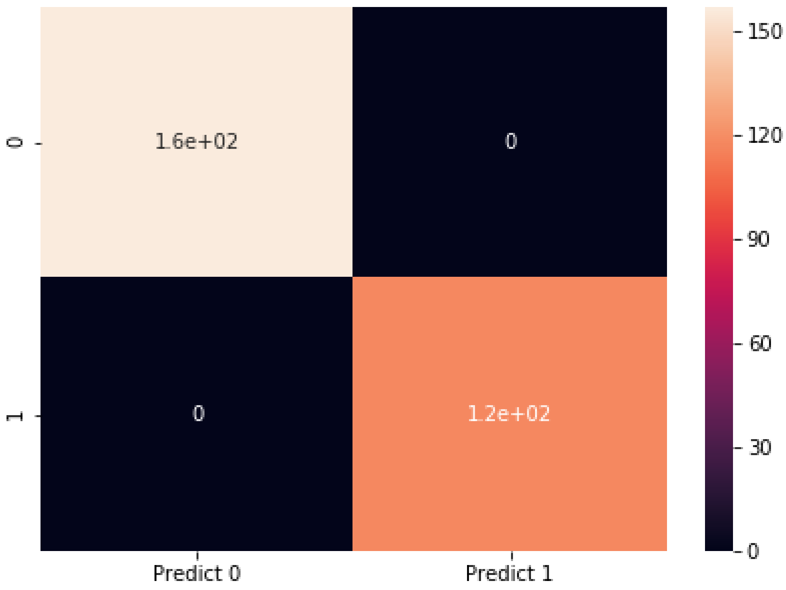
Out[107]:

```
1.0
```

In [108]:
```
print("Confusion Matrix For MLPClassifier")
cm=metrics.confusion_matrix(y_test,multi_predicted, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in [0,1]],
                     columns = [i for i in ["Predict 0","Predict 1"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For MLPClassifier
```

Out[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12718aa20>
```



In [ ]:

Printing each Algorithm and the accuracy score

In [109]:
```
print("LogisticRegression:", accuracy_score(y_test,y_test_pred))
print("Support Vector Machine (using kernel=linear):", accuracy_score(y_test,sv
m_pred))
print("Support Vector Machine (using kernel=rbf):", accuracy_score(y_test,svm_r
bf_pred))
print("RandomForestClassifier:", accuracy_score(y_test,rdf_pred))
print("KNeighborsClassifier:", accuracy_score(y_test,KNN_predicted))
print("MLPClassifier:", accuracy_score(y_test,multi_predicted))
```

```
LogisticRegression: 0.9745454545454545
Support Vector Machine (using kernel=linear): 0.9818181818181818
Support Vector Machine (using kernel=rbf): 1.0
RandomForestClassifier: 0.9963636363636363
KNeighborsClassifier: 1.0
MLPClassifier: 1.0
```

It can be seen that Support Vector Machine (using kernel=rbf), KNeighborsClassifier: 1.0 and MLPClassifier: 1.0 are having hightest accuracy score of 100% and RandomForestClassifier is also doing very great with accuracy score of 99%

These can also be verified from the confusion matrix

Accuracy of 100% is quite weired and I suggest you try different approach to verify this such instead of 80:20, try 70:30 splitting and also try tuning the parameters of the different algorithms to verfify you results

logistic regression documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

SVM documentation: https://scikit-learn.org/stable/modules/svm.html (https://scikit-learn.org/stable/modules/svm.html)

GridSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

In [ ]:

In [ ]: