Importing pandas and reading dataset

In [ ]:
```
import pandas as pd
CLASS = "class"

df = pd.read_csv("./pd_speech_features.csv") # i deleted the first row in the file. it contained nothing useful!
df
```

Out[ ]:

	Unnamed: 0	Unnamed: 1	Baseline Features	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	
0	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_ku
1	0	1	0.85247	0.71826	0.57227	240	239	0.00806353	8.68E-05	0.00218	...	
2	0	1	0.76686	0.69481	0.53966	234	233	0.008258256	7.31E-05	0.00195	...	
3	0	1	0.85083	0.67604	0.58982	232	231	0.00833959	6.04E-05	0.00176	...	
4	1	0	0.41121	0.79672	0.59257	178	177	0.010857733	0.000182739	0.00419	...	
...	...	...	...	...	...	...	...	...	...	...	...	
752	250	0	0.80903	0.56355	0.28385	417	416	0.004626942	5.22E-05	0.00064	...	
753	250	0	0.16084	0.56499	0.59194	415	413	0.004549703	0.000219994	0.00143	...	
754	251	0	0.88389	0.72335	0.46815	381	380	0.005069271	0.000102654	0.00076	...	
755	251	0	0.83782	0.7489	0.49823	340	339	0.005679019	5.51E-05	0.00092	...	
756	251	0	0.81304	0.76471	0.46374	340	339	0.005675776	3.71E-05	0.00078	...	

757 rows × 755 columns

Correcting dataframe header

In [ ]:
```
headers = df.iloc[0]
df  = pd.DataFrame(df.values[1:], columns=headers)
```

Analyzing Dataset

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: object(755)
memory usage: 4.4+ MB
```

In [ ]:
```
df.describe()
```

Out[ ]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisV
count	756	756	756	756	756	756	756	756	756	756	...	
unique	252	2	740	745	748	315	319	755	646	358	...	
top	0	1	0.82273	0.72248	0.62128	237	236	0.006004477	7.17E-05	0.00076	...	
freq	3	390	3	2	2	9	8	2	3	9	...	

4 rows × 755 columns

In [ ]:
```
df[CLASS].value_counts()
```

Out[ ]:
```
1    564
0    192
Name: class, dtype: int64
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
0
id                          0
gender                      0
PPE                         0
DFA                         0
RPDE                        0
                           ..
tqwt_kurtosisValue_dec_33   0
tqwt_kurtosisValue_dec_34   0
tqwt_kurtosisValue_dec_35   0
tqwt_kurtosisValue_dec_36   0
class                       0
Length: 755, dtype: int64
```

In [ ]:
```
df.isnull().sum().sum()
```

Out[ ]:
```
0
```

Preprocessing

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler, minmax_scale

df.iloc[:, :-1] = MinMaxScaler().fit_transform(df.iloc[:, :-1])
df
```

Out[ ]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kur
0	0.0	1.0	0.936278	0.56531	0.583	0.262983	0.263274	0.548552	0.021947	0.071532	...	
1	0.0	1.0	0.837434	0.489455	0.537514	0.256354	0.256637	0.566485	0.018001	0.063181	...	
2	0.0	1.0	0.934385	0.428738	0.607479	0.254144	0.254425	0.573975	0.014344	0.056282	...	
3	0.003984	0.0	0.426804	0.819111	0.611315	0.194475	0.19469	0.805881	0.04958	0.144517	...	
4	0.003984	0.0	0.330615	0.822669	0.524431	0.258564	0.25885	0.557581	0.765643	0.186638	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
751	0.996016	0.0	0.886123	0.064857	0.180701	0.458564	0.459071	0.232063	0.011982	0.015614	...	
752	0.996016	0.0	0.13773	0.069515	0.610436	0.456354	0.455752	0.22495	0.06031	0.044299	...	
753	1.0	0.0	0.972555	0.581775	0.437769	0.418785	0.419248	0.272799	0.026514	0.019971	...	
754	1.0	0.0	0.919363	0.664424	0.479726	0.373481	0.373894	0.328953	0.012817	0.025781	...	
755	1.0	0.0	0.890753	0.715566	0.431618	0.373481	0.373894	0.328654	0.007633	0.020697	...	

756 rows × 755 columns

In [ ]:
```
Y = df[CLASS]
X = df.drop(CLASS, axis=1)
```

In [ ]:
```
X
```

Out[ ]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kur
0	0.0	1.0	0.936278	0.56531	0.583	0.262983	0.263274	0.548552	0.021947	0.071532	...	
1	0.0	1.0	0.837434	0.489455	0.537514	0.256354	0.256637	0.566485	0.018001	0.063181	...	
2	0.0	1.0	0.934385	0.428738	0.607479	0.254144	0.254425	0.573975	0.014344	0.056282	...	
3	0.003984	0.0	0.426804	0.819111	0.611315	0.194475	0.19469	0.805881	0.04958	0.144517	...	
4	0.003984	0.0	0.330615	0.822669	0.524431	0.258564	0.25885	0.557581	0.765643	0.186638	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
751	0.996016	0.0	0.886123	0.064857	0.180701	0.458564	0.459071	0.232063	0.011982	0.015614	...	
752	0.996016	0.0	0.13773	0.069515	0.610436	0.456354	0.455752	0.22495	0.06031	0.044299	...	
753	1.0	0.0	0.972555	0.581775	0.437769	0.418785	0.419248	0.272799	0.026514	0.019971	...	
754	1.0	0.0	0.919363	0.664424	0.479726	0.373481	0.373894	0.328953	0.012817	0.025781	...	
755	1.0	0.0	0.890753	0.715566	0.431618	0.373481	0.373894	0.328654	0.007633	0.020697	...	

756 rows × 754 columns

In [ ]:
```
Y
```

Out[ ]:
```
0      1
1      1
2      1
3      1
4      1
      ..
751    0
752    0
753    0
754    0
755    0
Name: class, Length: 756, dtype: object
```

In [ ]:
```
Y.value_counts()
```

Out[ ]:
```
1    564
0    192
Name: class, dtype: int64
```

As you see the size of category with value of 1 is mush bigger than 0 and this will force the model to

predict 1 more ...

In [ ]:
```
564/(192 + 562)
```

Out[ ]:
```
0.7480106100795756
```

If our model predict 1 for all inputs then we'll get 74% !

Spliting Dataset into train and test

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=None)
x_train
```

Out[ ]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kur
308	0.406375	1.0	0.490884	0.899172	0.692773	0.276243	0.276549	0.510003	0.036605	0.23602	...	
24	0.031873	1.0	0.908926	0.419325	0.435049	0.287293	0.287611	0.484784	0.021919	0.069354	...	
706	0.936255	0.0	0.940412	0.284661	0.326713	0.439779	0.440265	0.249921	0.026437	0.025781	...	
285	0.378486	1.0	0.940469	0.563369	0.724799	0.370166	0.370575	0.333397	0.027492	0.109659	...	
500	0.661355	1.0	0.867188	0.626253	0.34995	0.334807	0.335177	0.390488	0.014056	0.038489	...	
...	...	...	...	...	...	...	...	...	...	...	...	
485	0.641434	1.0	0.905508	0.316588	0.236076	0.319337	0.31969	0.417225	0.011751	0.045389	...	
188	0.247012	0.0	0.78475	0.388238	0.550709	0.323757	0.324115	0.407214	0.376811	0.104575	...	
143	0.187251	1.0	0.929997	0.257909	0.415215	0.365746	0.36615	0.339437	0.01348	0.031227	...	
537	0.713147	1.0	0.231021	0.776089	0.668433	0.261878	0.262168	0.551446	0.042456	0.07008	...	
561	0.74502	0.0	0.887982	0.377014	0.29763	0.454144	0.454646	0.236288	0.011665	0.014524	...	

529 rows × 754 columns

In [ ]:
```
x_test
```

Out[ ]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kur
322	0.426295	1.0	0.929212	0.690076	0.216772	0.280663	0.280973	0.49931	0.012817	0.045025	...	
588	0.780876	0.0	0.904689	0.121369	0.230385	0.509392	0.509956	0.190304	0.003168	0.012709	...	
468	0.621514	1.0	0.071503	0.278353	0.599863	1.0	1.0	0.0	0.341725	0.02215	...	
527	0.697211	0.0	0.931394	0.039982	0.327284	0.553591	0.554204	0.159493	0.00697	0.015614	...	
212	0.278884	0.0	0.904758	0.629132	0.219394	0.670718	0.67146	0.097677	0.008986	0.016703	...	
...	...	...	...	...	...	...	...	...	...	...	...	
675	0.896414	0.0	0.878087	0.684156	0.360328	0.327072	0.327434	0.402856	0.012788	0.028322	...	
303	0.40239	1.0	0.850839	0.936631	0.337341	0.358011	0.358407	0.351524	0.023388	0.12963	...	
196	0.258964	1.0	0.932526	0.237401	0.256287	0.330387	0.330752	0.397404	0.004119	0.015614	...	
543	0.721116	0.0	0.932295	0.730834	0.294896	0.460773	0.461283	0.23023	0.014228	0.029049	...	
704	0.932271	0.0	0.943587	0.063628	0.389522	0.426519	0.426991	0.263701	0.021861	0.042847	...	

227 rows × 754 columns

In [ ]:
```
from sklearn.model_selection import ShuffleSplit, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns


cv = ShuffleSplit(n_splits=10, random_state=None)
def runCrossVal(model):
    score = cross_val_score(model, x_train, y_train, cv=cv).mean() * 100
    print(f"accuracy on cross_val_score: {score}")
    return model

def runOnTrain(model):
    model.fit(x_train, y_train)
    predicted_y = model.predict(x_train)
    print(f"accuracy on train: {accuracy_score(y_true=y_train, y_pred=predicted_y)*100}")

def runModelOnTest(model):
    runCrossVal(model)
    runOnTrain(model)
    model.fit(x_train, y_train)
    predicted_y = model.predict(x_test)
    print(f"accuracy on test data: {accuracy_score(y_true=y_test, y_pred=predicted_y)*100}")
```

```
    return predicted_y, y_test

def confusionMatrix(predicted_y, y_test):
    cf_matrix = confusion_matrix(y_test, predicted_y)
    sns.heatmap(cf_matrix, annot=True)
```

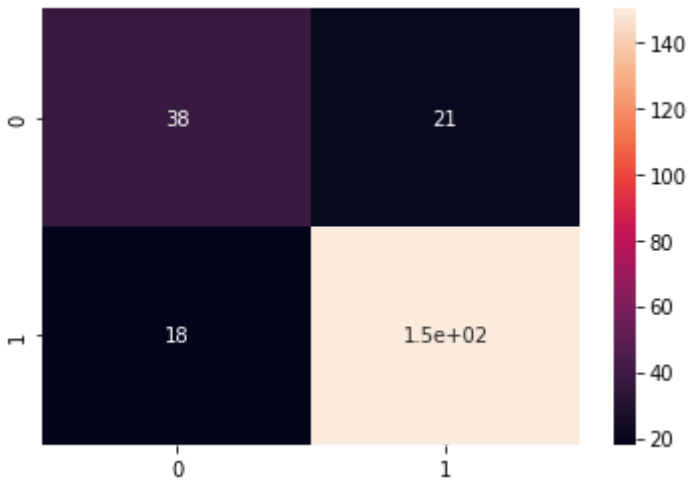
Decision Tree

in this model. in most of the times the model overfit. but with just a little playing with parameter we can improve that. and later when we used PCA to get out of this curse of dimensionality it gets much much better.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier as DecisionTree
```

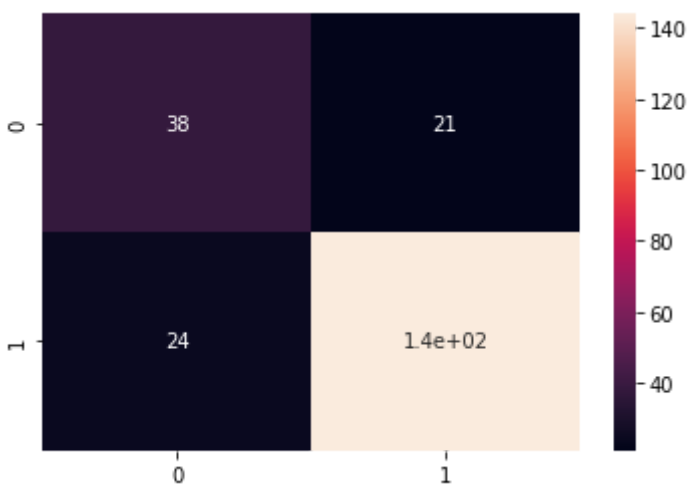
In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree())
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 77.54716981132074
accuracy on train: 100.0
accuracy on test data: 82.81938325991189
```



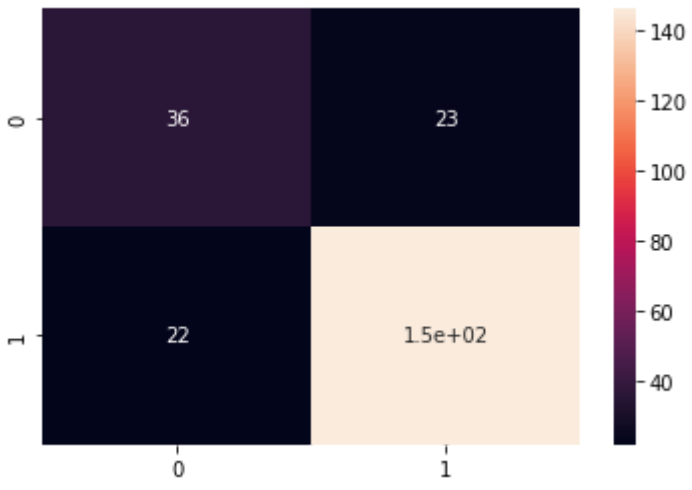
In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree(criterion='entropy', max_depth=10))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 81.1320754716981
accuracy on train: 100.0
accuracy on test data: 80.1762114537445
```



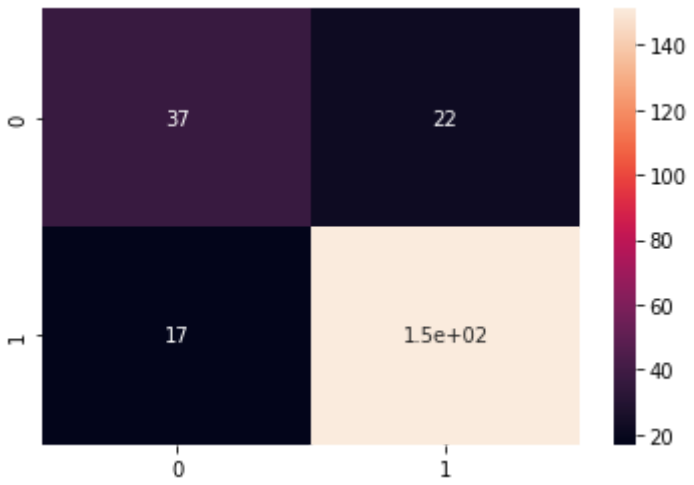
In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree(splitter="random", max_depth=10))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 81.88679245283019
accuracy on train: 100.0
accuracy on test data: 80.1762114537445
```



In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree(min_samples_split=3, max_depth=10))
confusionMatrix(predicted_y, y_test)
```

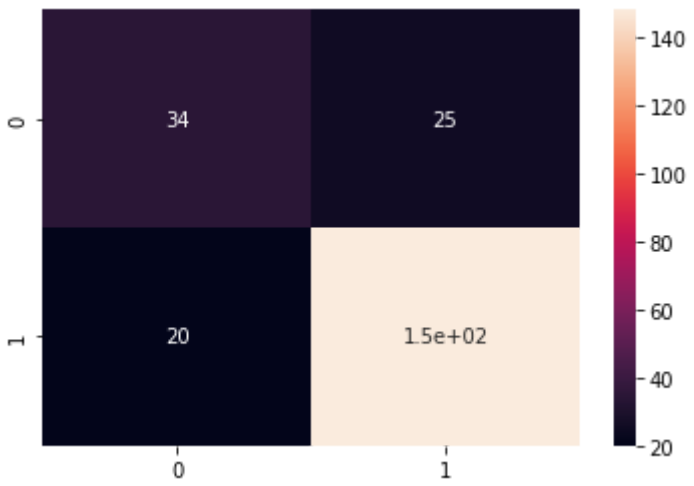
```
accuracy on cross_val_score: 80.9433962264151
accuracy on train: 99.24385633270322
accuracy on test data: 82.81938325991189
```



This one seems better anyway

In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree(min_samples_leaf=2, max_depth=8))
confusionMatrix(predicted_y, y_test)
```

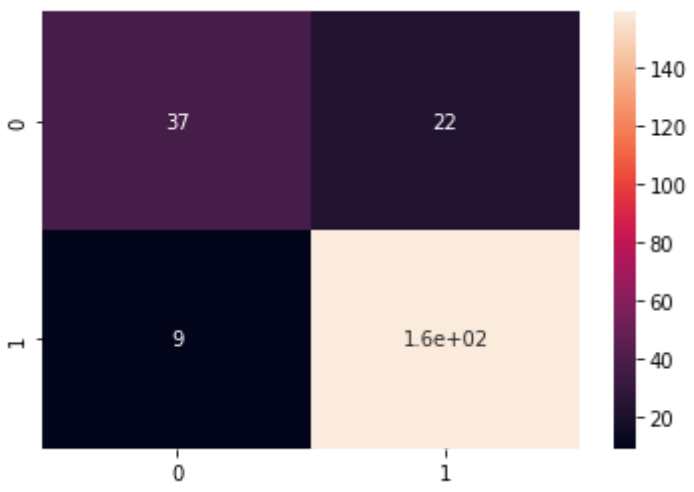
```
accuracy on cross_val_score: 78.11320754716982
accuracy on train: 96.78638941398866
accuracy on test data: 80.1762114537445
```



KNN

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier as KNN
predicted_y, y_test = runModelOnTest(KNN())
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 83.9622641509434
accuracy on train: 92.62759924385632
accuracy on test data: 86.34361233480176
```



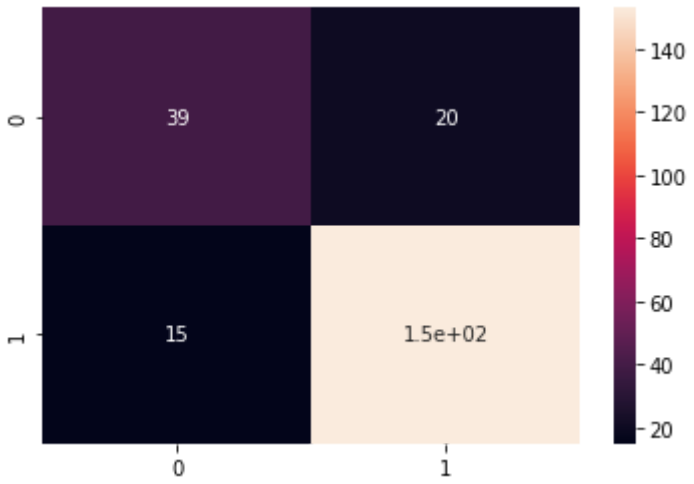
For KNN this one seems better.

first, 84% is not that bad for test.

also the accuracy over the train(91) is closer to accuracy over test than the others(we can say that the model hasn't overfit)

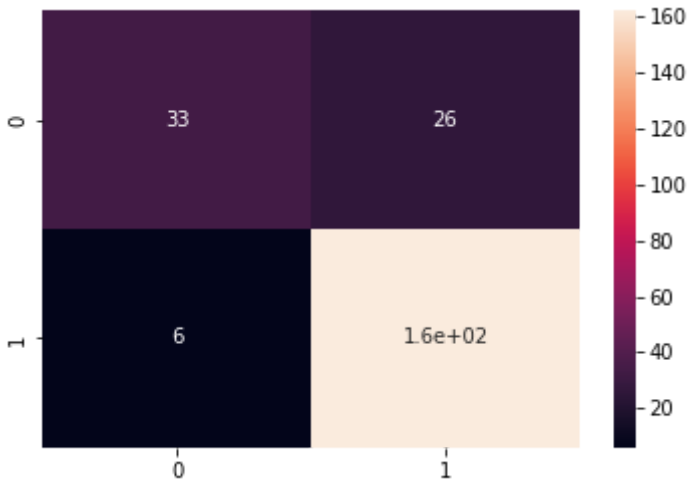
In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(6))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 84.71698113207546
accuracy on train: 91.68241965973534
accuracy on test data: 84.58149779735683
```



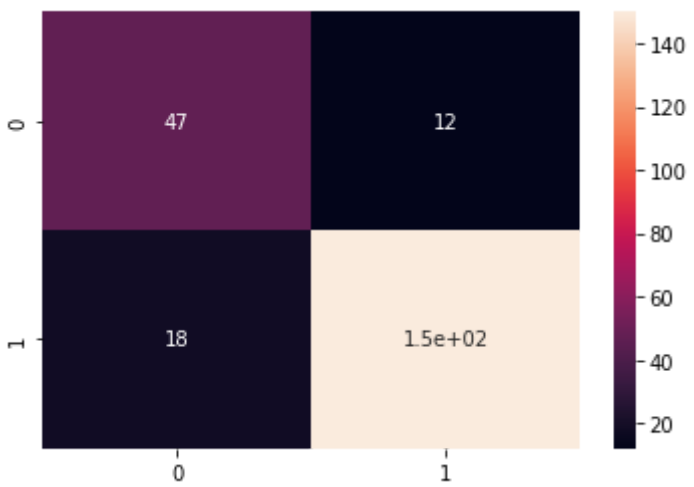
In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(7))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 84.71698113207549
accuracy on train: 89.79206049149339
accuracy on test data: 85.90308370044053
```



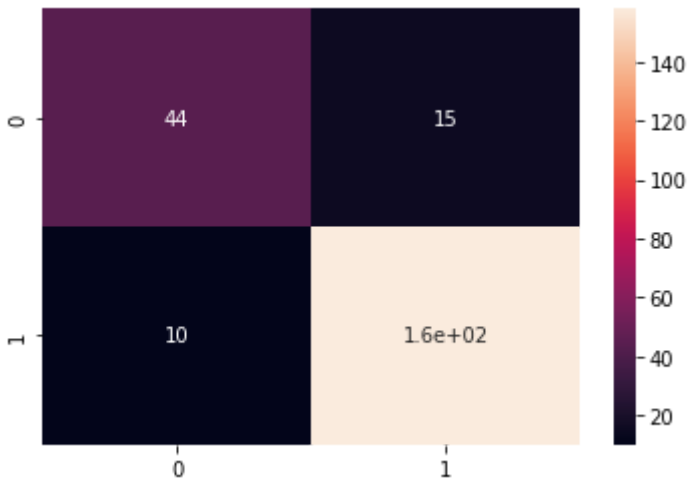
In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(4))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 85.47169811320755
accuracy on train: 94.5179584120983
accuracy on test data: 86.78414096916299
```



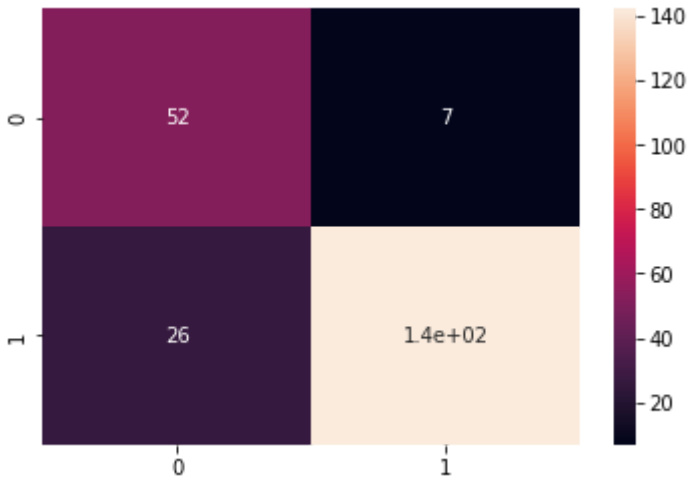
In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(3))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 85.0943396226415
accuracy on train: 97.54253308128544
accuracy on test data: 88.98678414096916
```



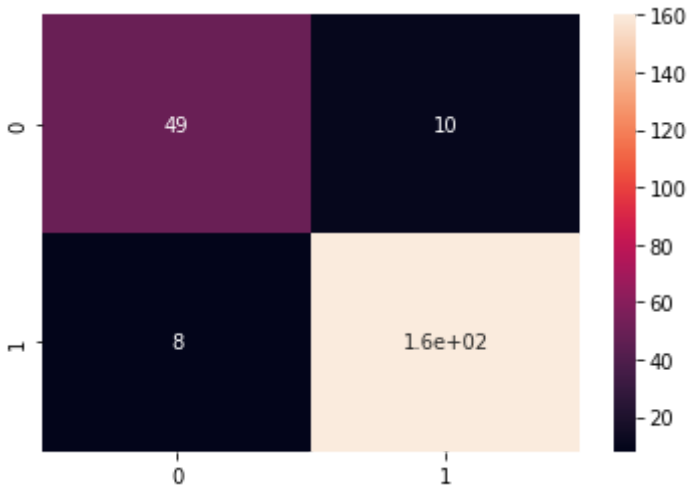
In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(2))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 92.26415094339622
accuracy on train: 96.78638941398866
accuracy on test data: 85.46255506607929
```



In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(1))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 92.64150943396226
accuracy on train: 100.0
accuracy on test data: 92.07048458149781
```

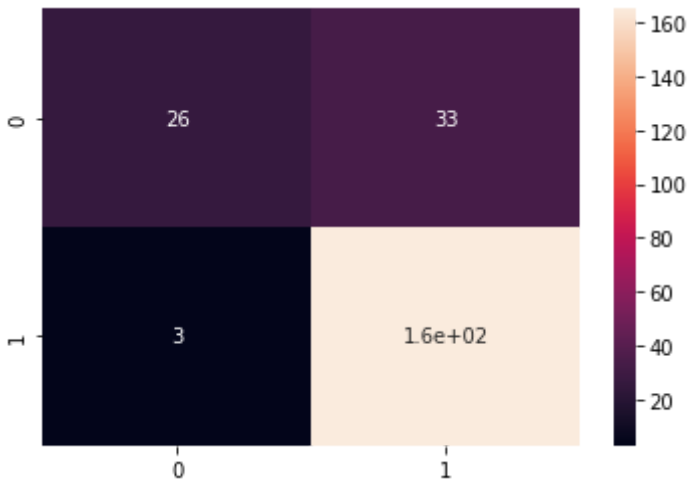


SVM

also for the SVC model the default parameter seems ok and is close to some other set of paramters below. both accuracy over train and test are close and not bad(84% and 87%) so model hasn't overfit

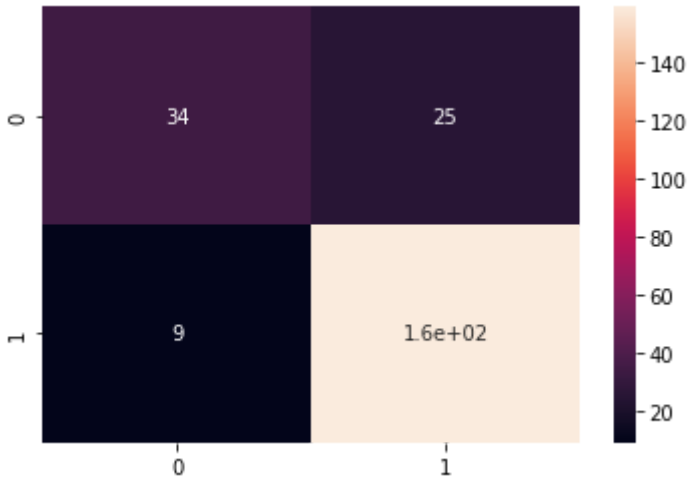
In [ ]:
```
from sklearn.svm import SVC
predicted_y, y_test = runModelOnTest(SVC())
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 83.58490566037737
accuracy on train: 87.14555765595463
accuracy on test data: 84.14096916299559
```



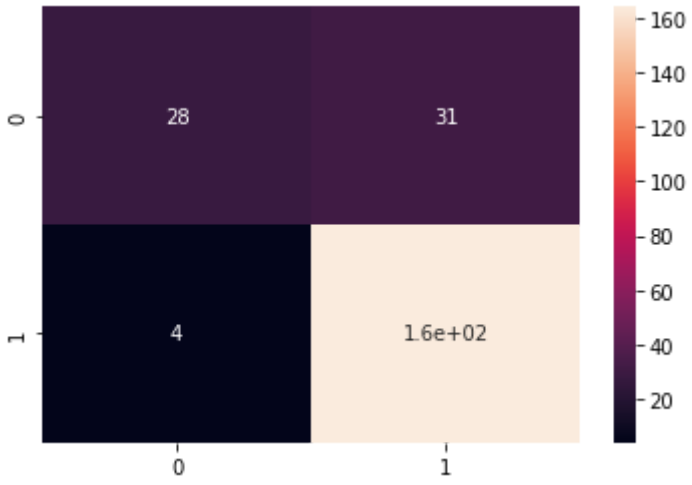
In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=15))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 90.18867924528303
accuracy on train: 95.65217391304348
accuracy on test data: 85.02202643171806
```



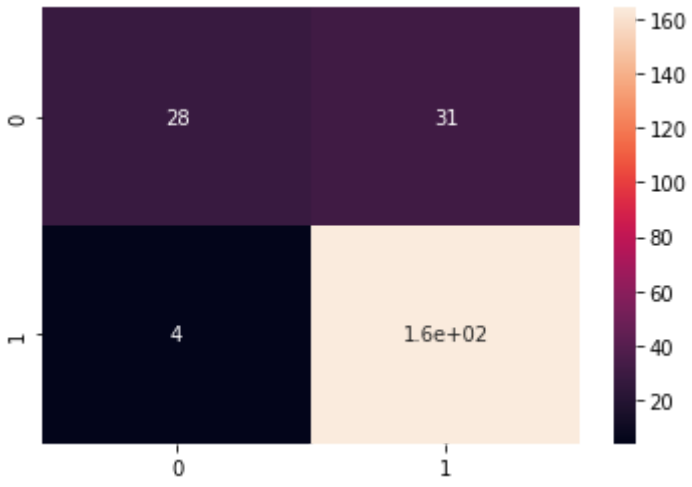
In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=15, gamma="auto"))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 84.52830188679246
accuracy on train: 88.27977315689981
accuracy on test data: 84.58149779735683
```



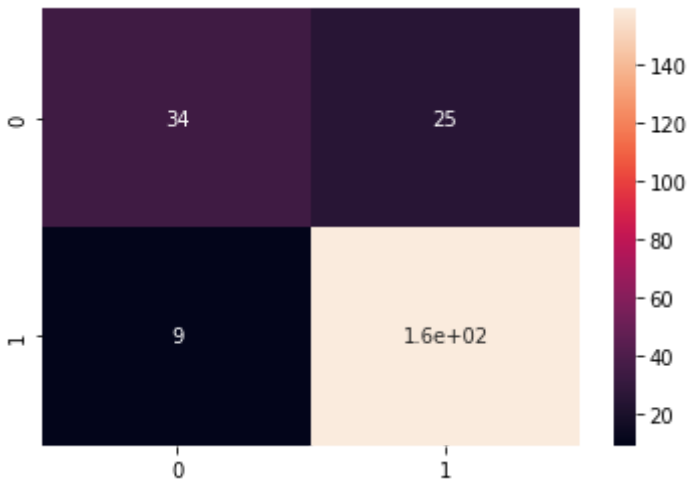
In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=15, gamma="auto"))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 84.71698113207549
accuracy on train: 88.27977315689981
accuracy on test data: 84.58149779735683
```



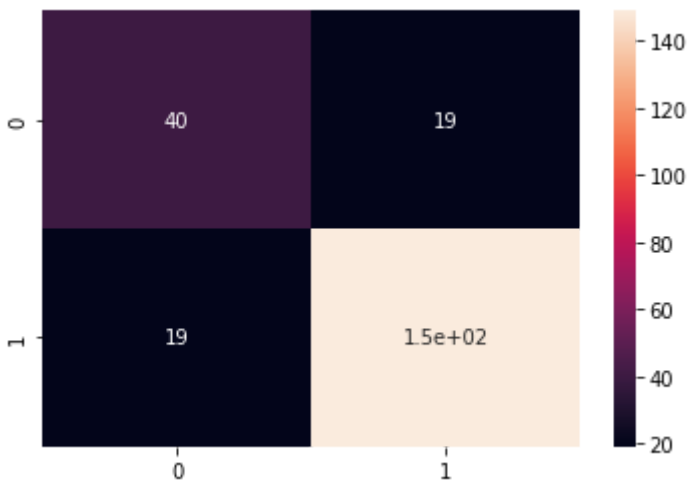
In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=15, coef0=0.1))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 89.05660377358491
accuracy on train: 95.65217391304348
accuracy on test data: 85.02202643171806
```



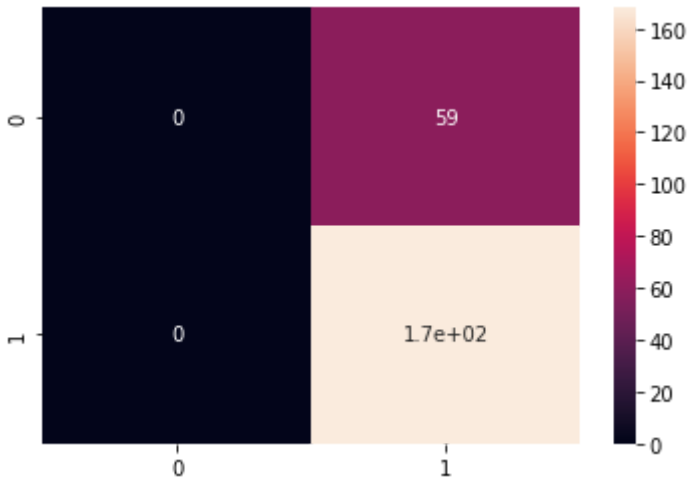
In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=15, coef0=0.1, kernel="linear"))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 83.39622641509435
accuracy on train: 100.0
accuracy on test data: 83.25991189427313
```



In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC(C=2, coef0=0.1, kernel="sigmoid"))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 75.84905660377358
accuracy on train: 74.85822306238185
accuracy on test data: 74.00881057268722
```

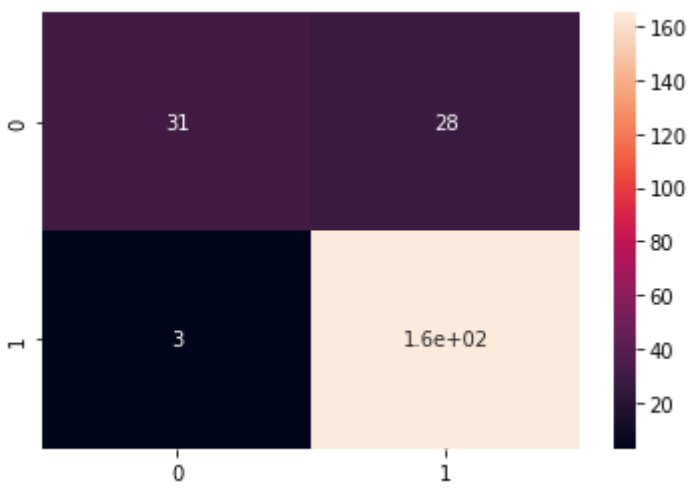


Random Forest

Random forest also overfit a lot. however we finally managed to avoid that using the "max_depth" parameter in the last cell

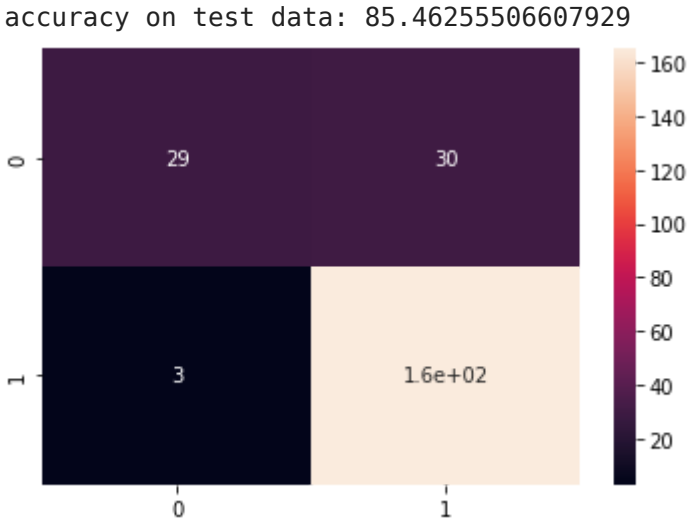
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier as RandomForest
predicted_y, y_test = runModelOnTest(RandomForest())
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 89.62264150943398
accuracy on train: 100.0
accuracy on test data: 86.34361233480176
```



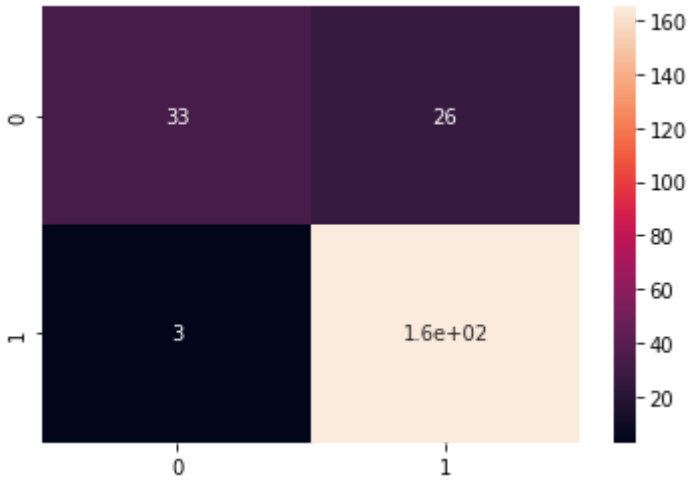
In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(n_estimators=150))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 86.41509433962264
accuracy on train: 100.0
accuracy on test data: 85.46255506607929
```



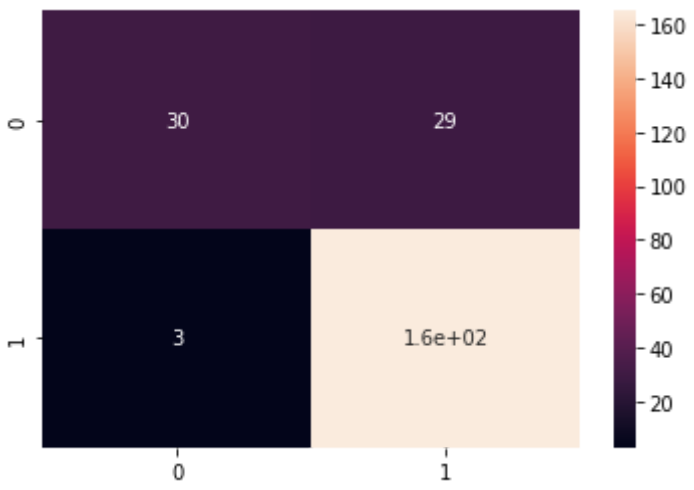
In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(n_estimators=80))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 88.30188679245283
accuracy on train: 100.0
accuracy on test data: 87.22466960352423
```



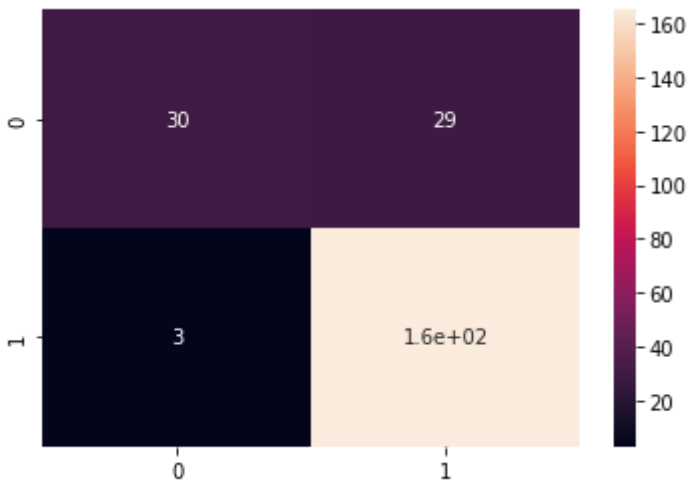
In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(criterion="entropy"))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 89.24528301886794
accuracy on train: 100.0
accuracy on test data: 85.90308370044053
```



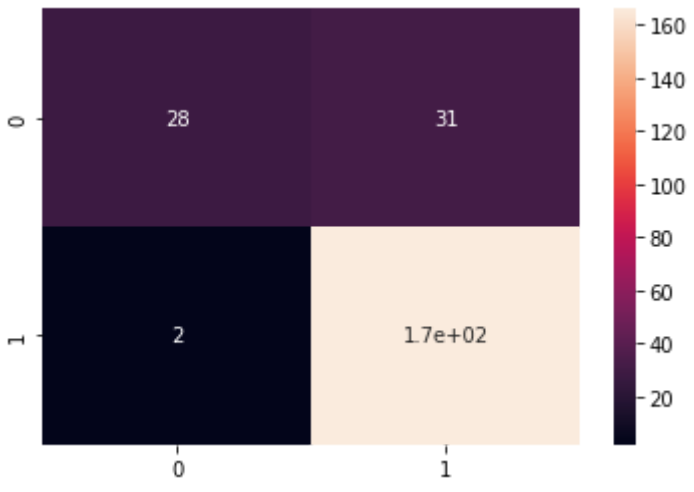
In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(criterion="entropy", n_estimators=50))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 85.47169811320755
accuracy on train: 100.0
accuracy on test data: 85.90308370044053
```



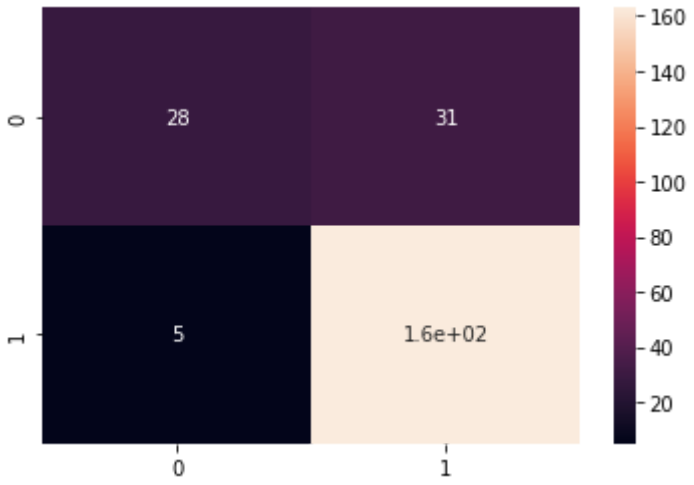
In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(criterion="entropy", min_samples_split=2, max_depth=8))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 88.11320754716981
accuracy on train: 99.8109640831758
accuracy on test data: 85.46255506607929
```



In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(criterion="entropy", min_samples_split=2, max_depth=6))
confusionMatrix(predicted_y, y_test)
```

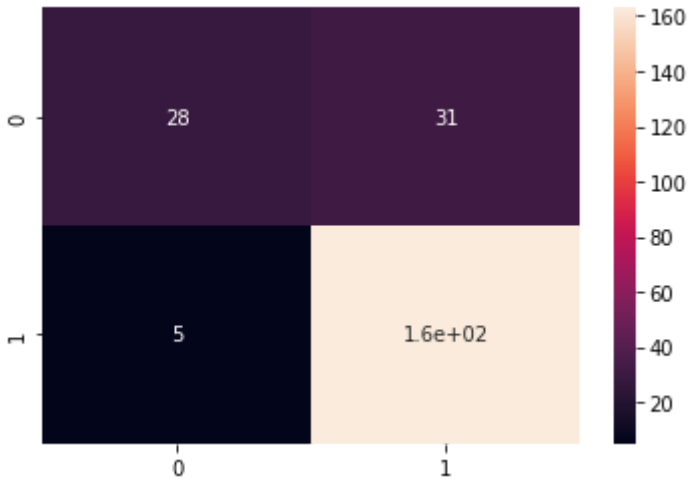
```
accuracy on cross_val_score: 85.09433962264153
accuracy on train: 98.67674858223062
accuracy on test data: 84.14096916299559
```



Here it seems that max_depth of 4 is useful to avoid the model from overfitting

In [ ]:
```
y_predict, y_test = runModelOnTest(RandomForest(criterion="entropy", min_samples_split=2, max_depth=4))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 84.15094339622642
accuracy on train: 90.92627599243856
accuracy on test data: 83.70044052863436
```



PCA

In [ ]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
X = pd.DataFrame(pca.fit_transform(X))
X
```

Out[ ]:

	0	1	2
0	-1.345269	-0.561286	-1.013519
1	-1.457767	-0.497860	-1.117411
2	-1.929643	-0.181124	-0.952128
3	-0.387073	-1.768126	1.814325
4	-0.162711	-1.188957	1.912331
...	...	...	...
751	3.326896	-0.101502	-0.079766
752	2.175160	0.245812	1.182430
753	1.210299	-0.037975	0.001728
754	0.793806	-0.622370	0.119452
755	0.730732	-0.704891	-0.010727

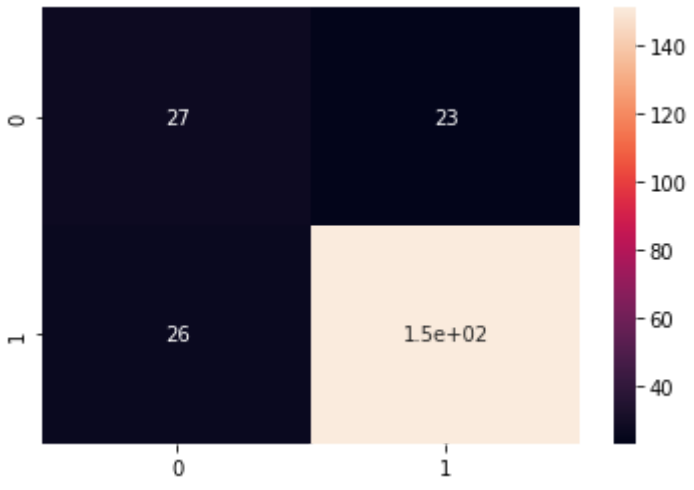
756 rows × 3 columns

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=None)
```

Decision Tree

In [ ]:
```
predicted_y, y_test = runModelOnTest(DecisionTree(min_samples_leaf=2, max_depth=8))
confusionMatrix(predicted_y, y_test)
```

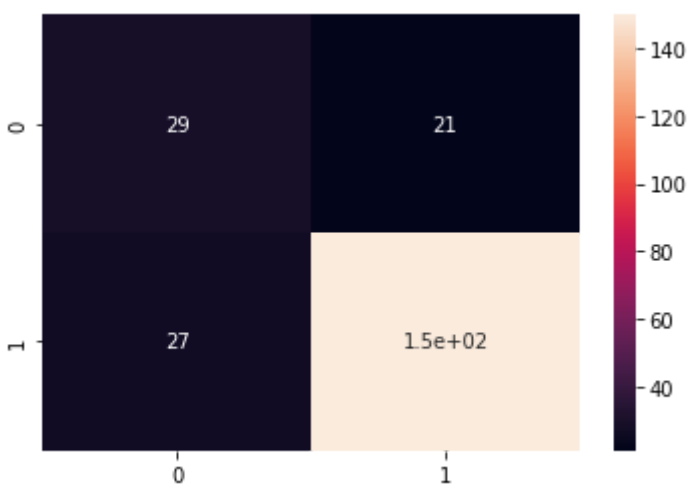
```
accuracy on cross_val_score: 72.26415094339622
accuracy on train: 92.43856332703214
accuracy on test data: 78.41409691629956
```



KNN

In [ ]:
```
predicted_y, y_test = runModelOnTest(KNN(3))
confusionMatrix(predicted_y, y_test)
```

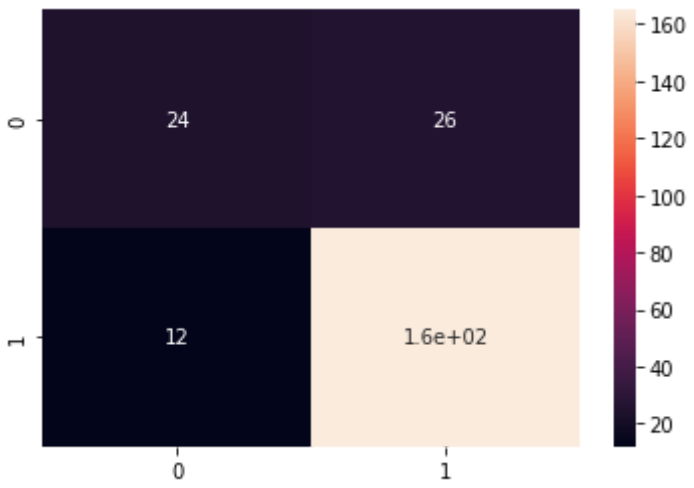
```
accuracy on cross_val_score: 77.16981132075473
accuracy on train: 87.90170132325142
accuracy on test data: 78.8546255506608
```



SVM

In [ ]:
```
predicted_y, y_test = runModelOnTest(SVC())
confusionMatrix(predicted_y, y_test)
```

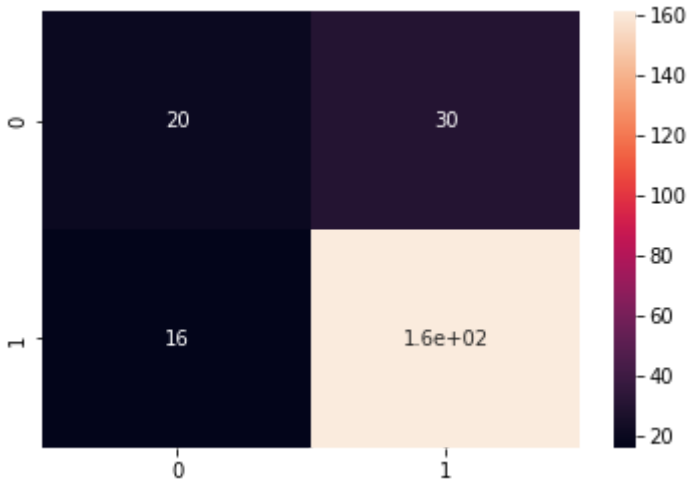
```
accuracy on cross_val_score: 79.0566037735849
accuracy on train: 79.77315689981096
accuracy on test data: 83.25991189427313
```



Random Forest

In [ ]:
```
predicted_y, y_test = runModelOnTest(RandomForest(criterion="entropy", min_samples_split=2, max_depth=4))
confusionMatrix(predicted_y, y_test)
```

```
accuracy on cross_val_score: 75.66037735849058
accuracy on train: 82.6086956521739
accuracy on test data: 79.73568281938326
```



BEFORE PCA:

In [ ]:
```
d = pd.DataFrame({"Model": ["Decision Tree", "KNN", "SVM", "Random Forst"],
                  "train acc": [96.786, 91.682, 87.145, 90.926],
                  "Cross Val": [78.1132, 84.7169, 83.5849, 84.1509],
                  "Test": [80.1762, 84.5814, 84.1409, 83.7004]})
d
```

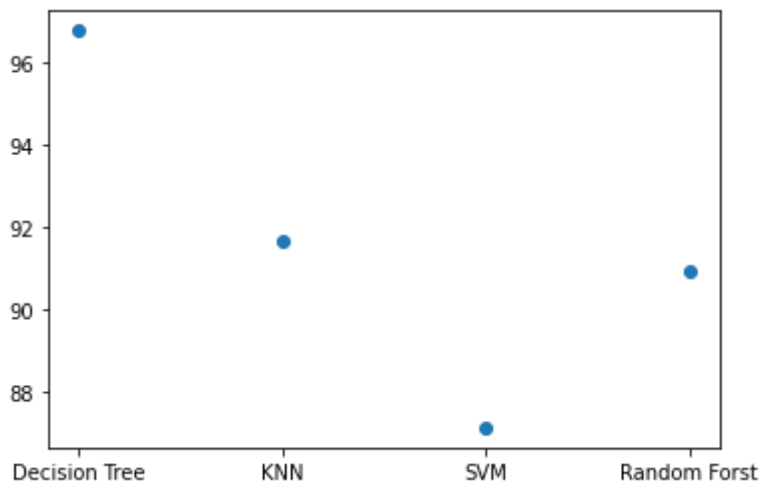
Out[ ]:

	Model	train acc	Cross Val	Test
0	Decision Tree	96.786	78.1132	80.1762
1	KNN	91.682	84.7169	84.5814
2	SVM	87.145	83.5849	84.1409
3	Random Forst	90.926	84.1509	83.7004

In [ ]:
```
import matplotlib.pyplot as plt
```

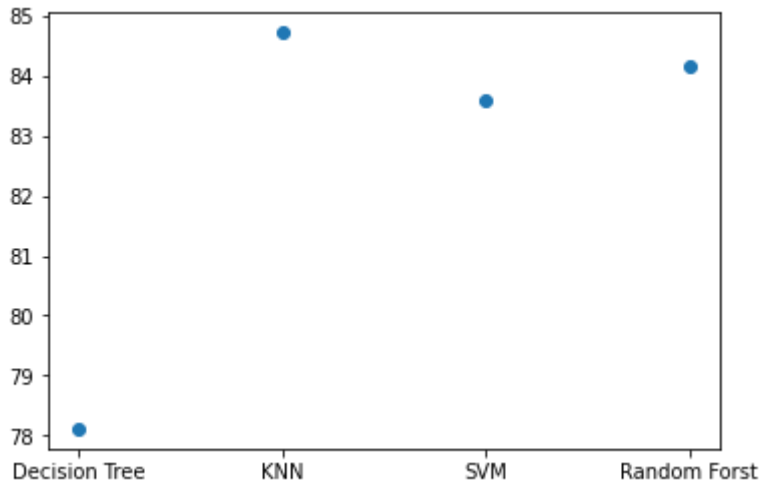
In [ ]:
```
plt.scatter(d["Model"], d["train acc"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f90298f40>
```



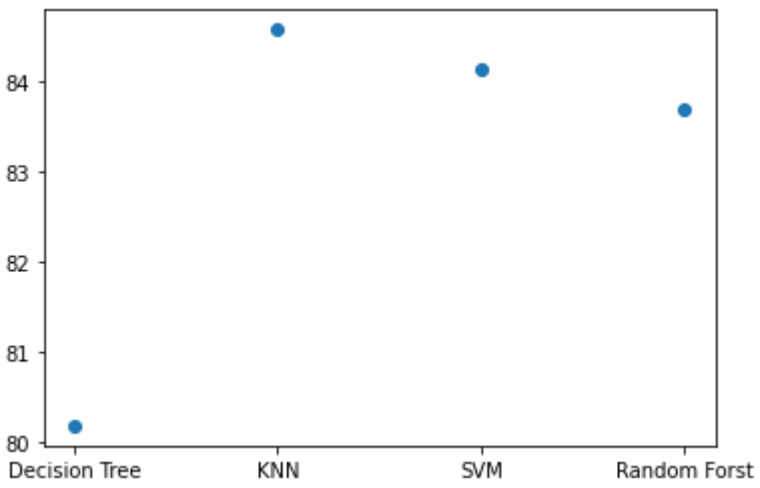
In [ ]:
```
plt.scatter(d["Model"], d["Cross Val"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f9027a8b0>
```



In [ ]:
```
plt.scatter(d["Model"], d["Test"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f901df460>
```



AFTER PCA

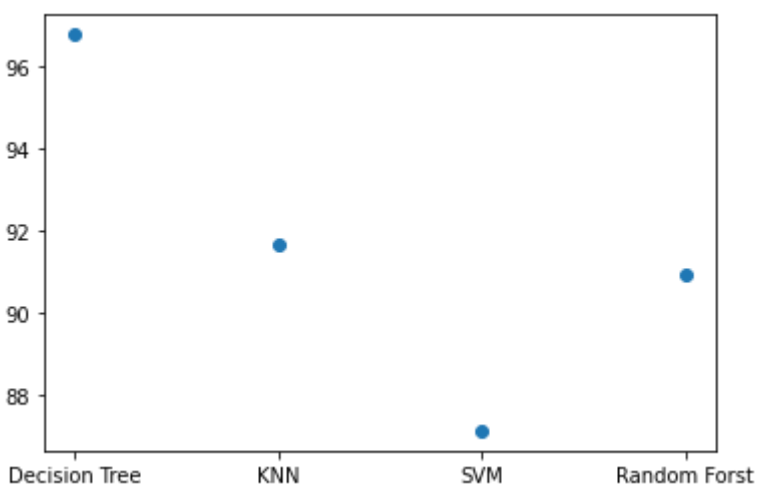
In [ ]:
```
d = pd.DataFrame({"Model": ["Decision Tree", "KNN", "SVM", "Random Forst"],
                  "train acc": [92.4385, 87.9017, 79.7731, 82.6086],
                  "Cross Val": [72.2641, 77.1698, 79.0566, 75.6603],
                  "Test": [78.4140, 78.8546, 83.2599, 79.7356]})
d
```

Out[ ]:

	Model	train acc	Cross Val	Test
0	Decision Tree	92.4385	72.2641	78.4140
1	KNN	87.9017	77.1698	78.8546
2	SVM	79.7731	79.0566	83.2599
3	Random Forst	82.6086	75.6603	79.7356

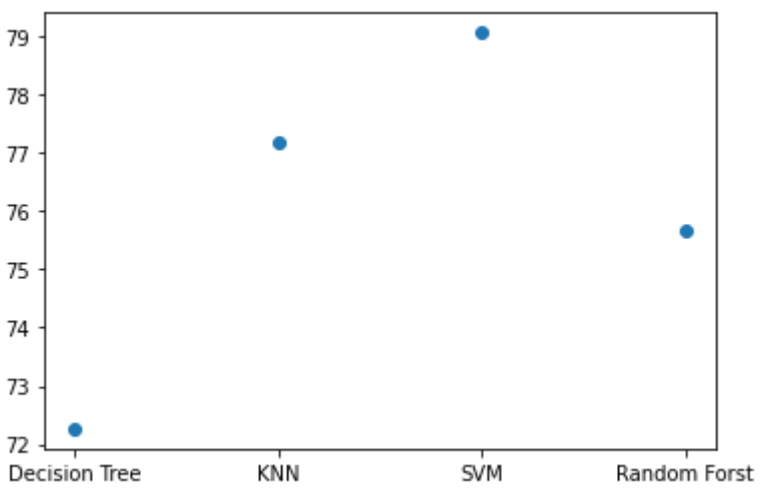
In [ ]:
```
plt.scatter(d["Model"], d["train acc"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f901aee80>
```



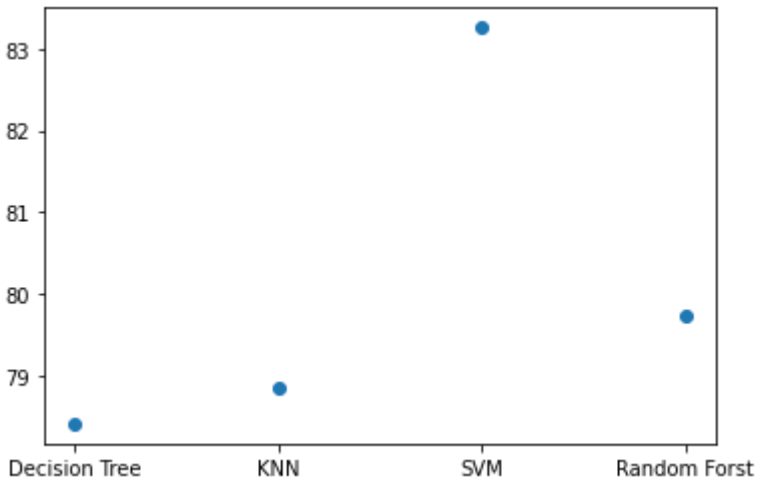
In [ ]:
```
plt.scatter(d["Model"], d["Cross Val"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f9036a940>
```



In [ ]:
```
plt.scatter(d["Model"], d["Test"])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f9f90308850>
```



as you can see we can conclude that PCA can potentially reduce accuracy becase we're going to some lower-dimentional space and lose some information.

so it's rational to get lower accuracy