In [28]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn import preprocessing
from sklearn import neighbors
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn import tree
from sklearn import naive_bayes
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold
from sklearn.ensemble import AdaBoostClassifier
%matplotlib inline
```

Question 1: Group Info

Group Name: Plum

Group Member: Eric Grant

Question 2: Movie Hits

In [34]:
```
movies = pd.read_csv('./hit-movies.csv')

xList = ["in_collection","Action","Adventure","Animation","Comedy","Crime","Docu

finalAcc = []
finalF1  = []
finalAuc = []

i = 1
xData = movies[xList].to_numpy()
yData = movies["Hit"].to_numpy()
#used stratifiedShuffleSplit to improve balance of class labels
#based on the description of the function, should be very similar
#to KFold
skf = StratifiedShuffleSplit(n_splits=10, random_state=3)
for trainI, testI in skf.split(xData, yData):
    #clear arrays
    acc = []
    f1  = []
    auc = []

    #set up data
    x_train_pre, x_test_pre = xData[trainI], xData[testI]
```

```
y_train, y_test = yData[trainI], yData[testI]
scaler = preprocessing.MinMaxScaler().fit(x_train_pre)
x_train = scaler.transform(x_train_pre)
x_test = scaler.transform(x_test_pre)

# Q2c - Knn
for n in [3,9,15]:
    #knn
    knn = neighbors.KNeighborsClassifier(n_neighbors=n)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    #accuracy
    acc.append(accuracy_score(y_test, y_pred))
    #F1-measure
    f1.append(f1_score(y_test, y_pred))
    #AUC
    auc.append(roc_auc_score(y_test, y_pred))

# Q2d - Decision Trees
#decision tree full
dt = tree.DecisionTreeClassifier(class_weight={0:1, 1:6})
dtf = dt.fit(x_train, y_train)
y_pred = dtf.predict(x_test)
#accuracy
acc.append(accuracy_score(y_test, y_pred))
#F1-measure
f1.append(f1_score(y_test, y_pred))
#AUC
auc.append(roc_auc_score(y_test, y_pred))

#decision tree prunned
dt = tree.DecisionTreeClassifier(class_weight={0:1, 1:6}, max_leaf_nodes = 1
dtf = dt.fit(x_train, y_train)
y_pred = dtf.predict(x_test)
#accuracy
acc.append(accuracy_score(y_test, y_pred))
#F1-measure
f1.append(f1_score(y_test, y_pred))
#AUC
auc.append(roc_auc_score(y_test, y_pred))

# Q2e - Naive Bayes
gnb = naive_bayes.GaussianNB()
y_pred = gnb.fit(x_train, y_train).predict(x_test)
#accuracy
acc.append(accuracy_score(y_test, y_pred))
#F1-measure
f1.append(f1_score(y_test, y_pred))
#AUC
auc.append(roc_auc_score(y_test, y_pred))

# Q2f - Nested Cross Validation
#pre
cv = StratifiedShuffleSplit(n_splits=5, random_state=3)
print("Fold:", i)
i += 1

# Q2f.i - SVM
svmM = SVC(random_state=3)
params = [{"kernel":["rbf"],"C":[0.01,0.1,1]},{"kernel":["poly"],"C":[0.01,0
svmS = GridSearchCV(svmM, params, scoring="roc_auc", cv=cv, refit=True)
```

```
svmR = svmS.fit(x_train, y_train)
best = svmR.best_estimator_
y_pred = best.predict(x_test)
#accuracy
svmAcc = accuracy_score(y_test, y_pred)
acc.append(svmAcc)
#F1-measure
svmF1 = f1_score(y_test, y_pred)
f1.append(svmF1)
#AUC
svmAuc = roc_auc_score(y_test, y_pred)
auc.append(svmAuc)
#print
print("SVM -", svmR.best_params_)

# Q2f.ii - Forests
rfM = RandomForestClassifier(random_state=3)
rfSp = dict()
rfSp["n_estimators"] = [25,50,100]
rfSp["max_features"] = [6,10,14]
rfSe = GridSearchCV(rfM, rfSp, scoring="roc_auc", cv=cv, refit=True)
rfR = rfSe.fit(x_train, y_train)
best = rfR.best_estimator_
y_pred = best.predict(x_test)
#accuracy
rfAcc = accuracy_score(y_test, y_pred)
acc.append(rfAcc)
#F1-measure
rfF1 = f1_score(y_test, y_pred)
f1.append(rfF1)
#AUC
rfAuc = roc_auc_score(y_test, y_pred)
auc.append(rfAuc)
#print
print("Forest -", rfR.best_params_)

# Q2f.iii - AdaBoost
adaM = AdaBoostClassifier(random_state=3)
adaSp = dict()
adaSp["n_estimators"] = [25,50]
adaSe = GridSearchCV(adaM, adaSp, scoring="roc_auc", cv=cv, refit=True)
adaR = adaSe.fit(x_train, y_train)
best = adaR.best_estimator_
y_pred = best.predict(x_test)
#accuracy
adaAcc = accuracy_score(y_test, y_pred)
acc.append(adaAcc)
#F1-measure
adaF1 = f1_score(y_test, y_pred)
f1.append(adaF1)
#AUC
adaAuc = roc_auc_score(y_test, y_pred)
auc.append(adaAuc)
#print
print("AdaBoost -", adaR.best_params_)

#add to final data
finalAcc.append(acc)
finalF1.append(f1)
finalAuc.append(auc)
```

```
#final work and printing
dfAcc = pd.DataFrame(data=finalAcc, index=["F_1", "F_2", "F_3", "F_4", "F_5", "F
dfF1 = pd.DataFrame(data=finalF1, index=["F_1", "F_2", "F_3", "F_4", "F_5", "F_6
dfAuc = pd.DataFrame(data=finalAuc, index=["F_1", "F_2", "F_3", "F_4", "F_5", "F
```

```
Fold: 1
SVM - {'C': 0.01, 'kernel': 'rbf'}
Forest - {'max_features': 10, 'n_estimators': 100}
AdaBoost - {'n_estimators': 25}
Fold: 2
SVM - {'C': 0.1, 'degree': 2, 'kernel': 'poly'}
Forest - {'max_features': 14, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 3
SVM - {'C': 0.1, 'kernel': 'rbf'}
Forest - {'max_features': 10, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 4
SVM - {'C': 0.1, 'kernel': 'rbf'}
Forest - {'max_features': 6, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 5
SVM - {'C': 0.1, 'degree': 2, 'kernel': 'poly'}
Forest - {'max_features': 6, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 6
SVM - {'C': 1, 'degree': 2, 'kernel': 'poly'}
Forest - {'max_features': 10, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 7
SVM - {'C': 0.1, 'kernel': 'rbf'}
Forest - {'max_features': 10, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 8
SVM - {'C': 1, 'kernel': 'rbf'}
Forest - {'max_features': 10, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 9
SVM - {'C': 0.1, 'degree': 2, 'kernel': 'poly'}
Forest - {'max_features': 6, 'n_estimators': 100}
AdaBoost - {'n_estimators': 50}
Fold: 10
SVM - {'C': 0.01, 'degree': 2, 'kernel': 'poly'}
Forest - {'max_features': 6, 'n_estimators': 100}
AdaBoost - {'n_estimators': 25}
```

In [55]:

```
workAcc = dfAcc.copy()
workF1 = dfF1.copy()
workAuc = dfAuc.copy()

workAcc = workAcc.rename(columns={"KNN_3":"KNN3", "KNN_9":"KNN9", "KNN_15":"KNN1
workF1 = workF1.rename(columns={"kNN_3":"KNN3", "kNN_9":"KNN9", "kNN_15":"KNN15"
workAuc = workAuc.rename(columns={"kNN_3":"KNN3", "kNN_9":"KNN9", "kNN_15":"KNN1

output = pd.DataFrame(columns=["accuracy","F1-measure","AUC"])
names = ["KNN3","KNN9","KNN15","DT1","DT2","NB","best SVM","best RF","best AdaBo
for n in range(0,9):
    output.loc[names[n]] = [workAcc[names[n]].mean(), workF1[names[n]].mean(), w

display(output)
```

	accuracy	F1-measure	AUC

	accuracy	F1-measure	AUC
KNN3	0.781622	0.233850	0.549281
KNN9	0.825405	0.173809	0.540247
KNN15	0.830135	0.097203	0.521102
DT1	0.754324	0.279541	0.566322
DT2	0.550405	0.336864	0.601395
NB	0.594730	0.320336	0.583125
best SVM	0.831216	0.001587	0.500400
best RF	0.830946	0.076265	0.516491
best AdaBoost	0.830135	0.020172	0.503574