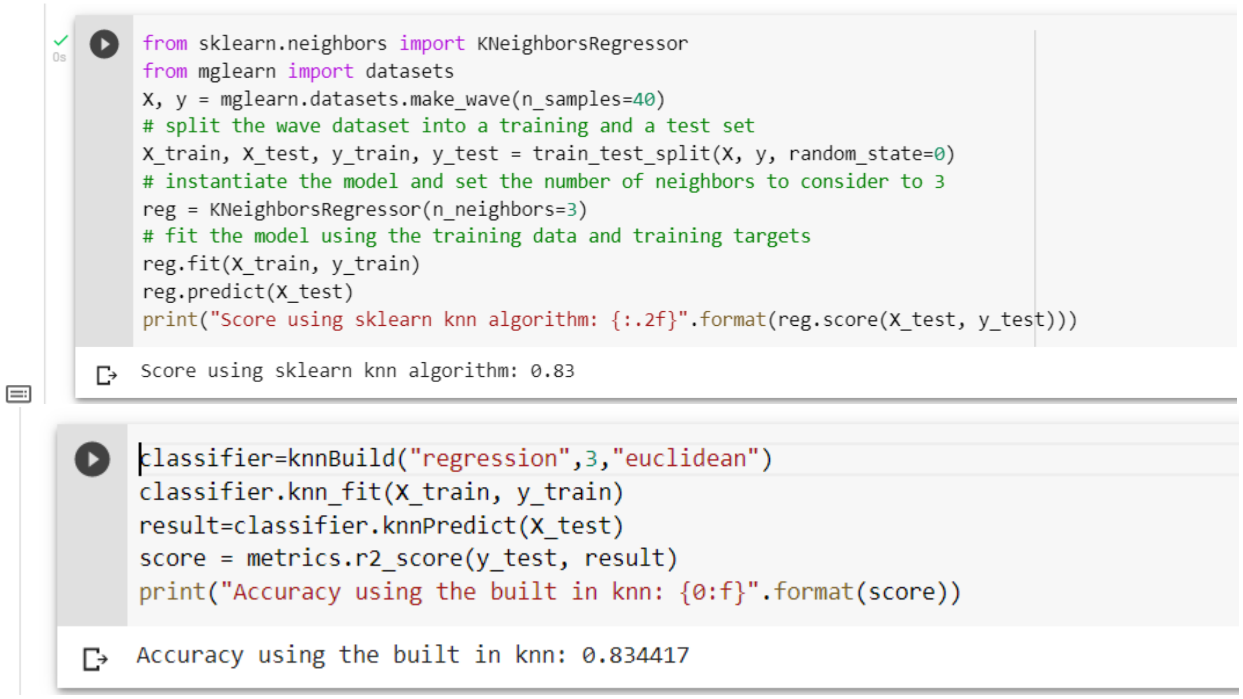
Name: Manavi Uttam Ghorpade

1. brief paragraph describing the performance of your implementation to that of scikit learn knn, as supported by your screen captured image (Task 2.4 above), embedded here.

Here is a screen shot of the accuracy of our custom knn and sklearn's Knn.(regression)

Our custom Knn for regression when tested with k=3 performs same as that of the sklearns regression knn. For large dataset our custom Knn may take some time but it still works very well. When we tested with other values of k like 1,5,9 our Knn works good and accuracy is above 0.70 always.



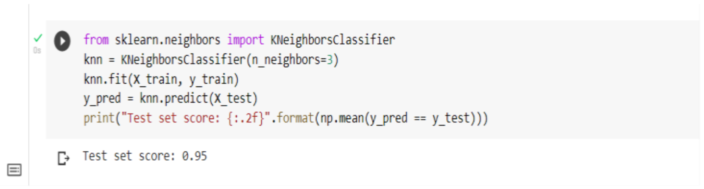
```
from sklearn.neighbors import KNeighborsRegressor
from mglearn import datasets
X, y = mglearn.datasets.make_wave(n_samples=40)
# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)
# fit the model using the training data and training targets
reg.fit(X_train, y_train)
reg.predict(X_test)
print("Score using sklearn knn algorithm: {:.2f}".format(reg.score(X_test, y_test)))
```

```
Score using sklearn knn algorithm: 0.83
```

```
classifier=knnBuild("regression",3,"euclidean")
classifier.knn_fit(X_train, y_train)
result=classifier.knnPredict(X_test)
score = metrics.r2_score(y_test, result)
print("Accuracy using the built in knn: {0:f}".format(score))
```

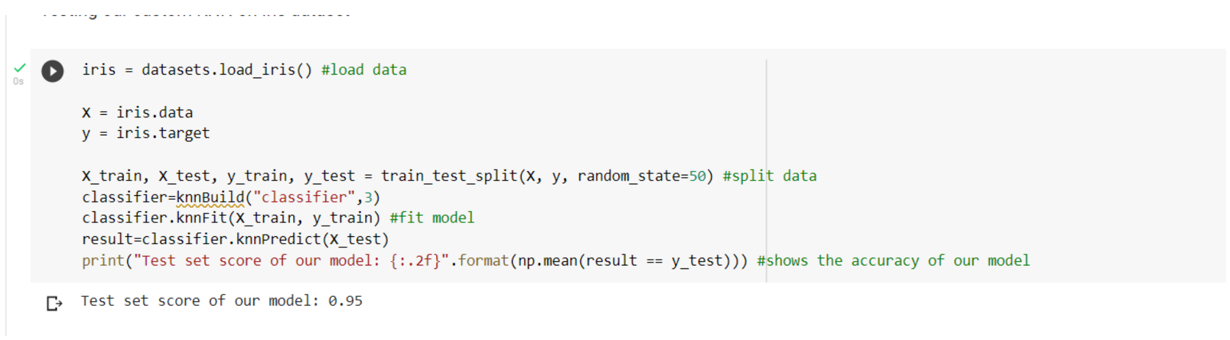
```
Accuracy using the built in knn: 0.834417
```

Here is another scrrenshot with Knn for classifier where our custom KNN works sames as the sklearns for value of k=3 and works same for all others.



```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.95
```



```
iris = datasets.load_iris() #load data

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=50) #split data
classifier=knnBuild("classifier",3)
classifier.knnFit(X_train, y_train) #fit model
result=classifier.knnPredict(X_test)
print("Test set score of our model: {:.2f}".format(np.mean(result == y_test))) #shows the accuracy of our model
```

```
Test set score of our model: 0.95
```

2.brief paragraph reflecting on how you would improve your implementation for future upgrades.

for better results, normalizing data on the same scale is highly recommended. Generally, the normalization range considered between 0 and 1. KNN is not suitable for the large dimensional data. In such cases, dimension needs to reduce to improve the performance. Also, handling missing values will help us in improving results. Moreover , we can remove the complications by providing the simple data .In future we can add our own test_split function and score function.