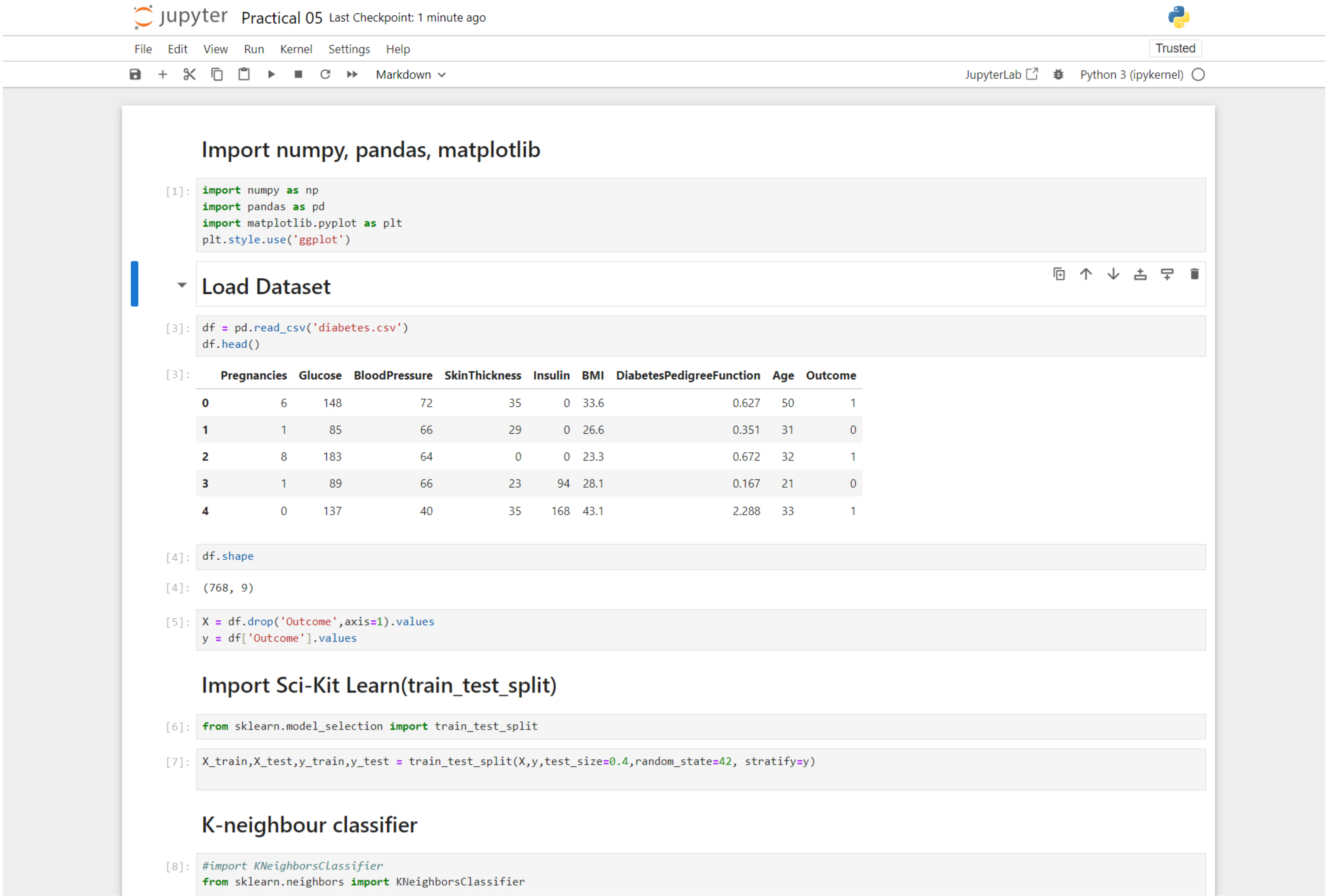


Import numpy, pandas, matplotlib

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

Load Dataset

[3]:
```
df = pd.read_csv('diabetes.csv')
df.head()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[4]:
```
df.shape
```

[4]:
```
(768, 9)
```

[5]:
```
X = df.drop('Outcome',axis=1).values
y = df['Outcome'].values
```

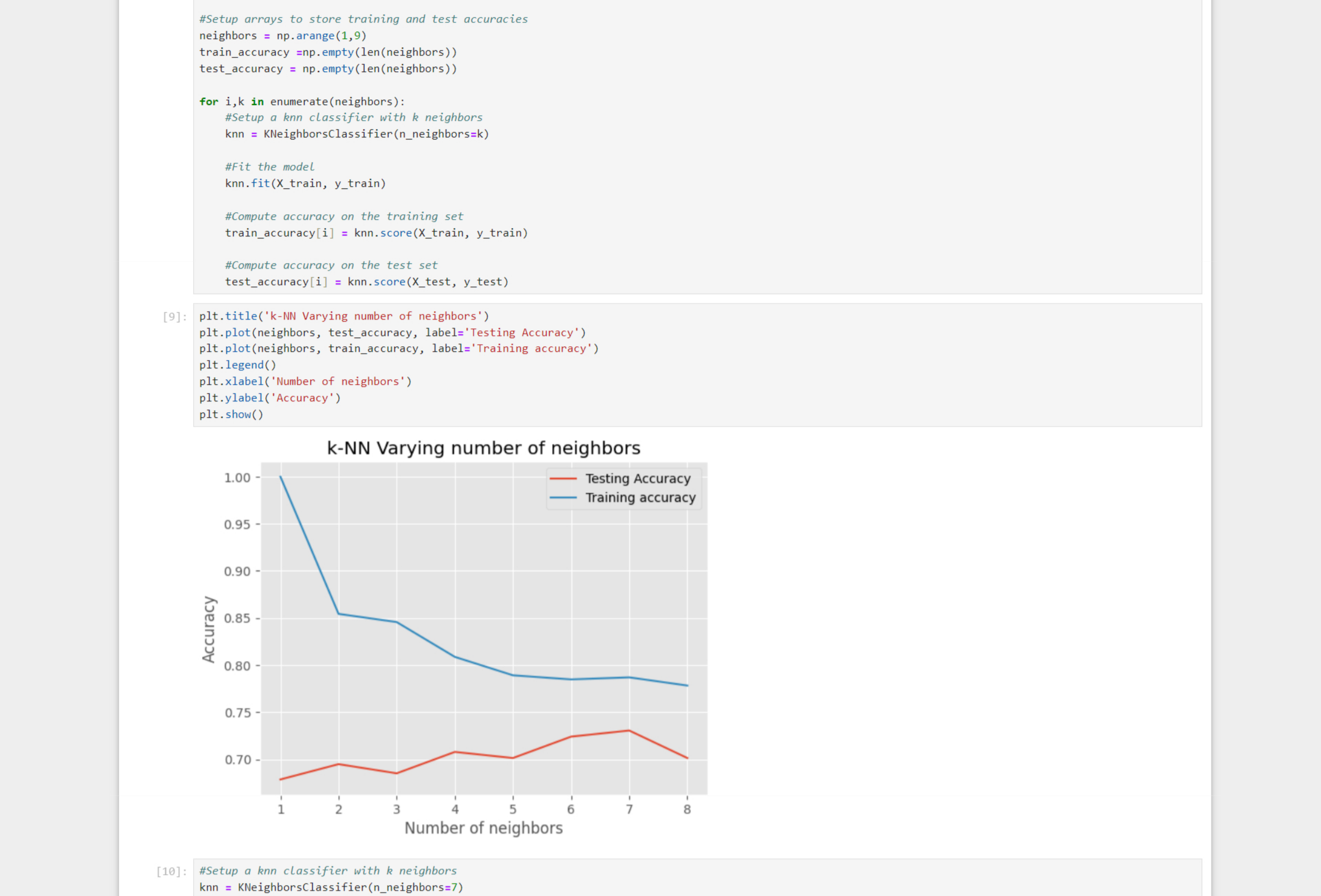
Import Sci-Kit Learn(train_test_split)

[6]:
```
from sklearn.model_selection import train_test_split
```

[7]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)
```

K-neighbour classifier

[8]:
```
#import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
```



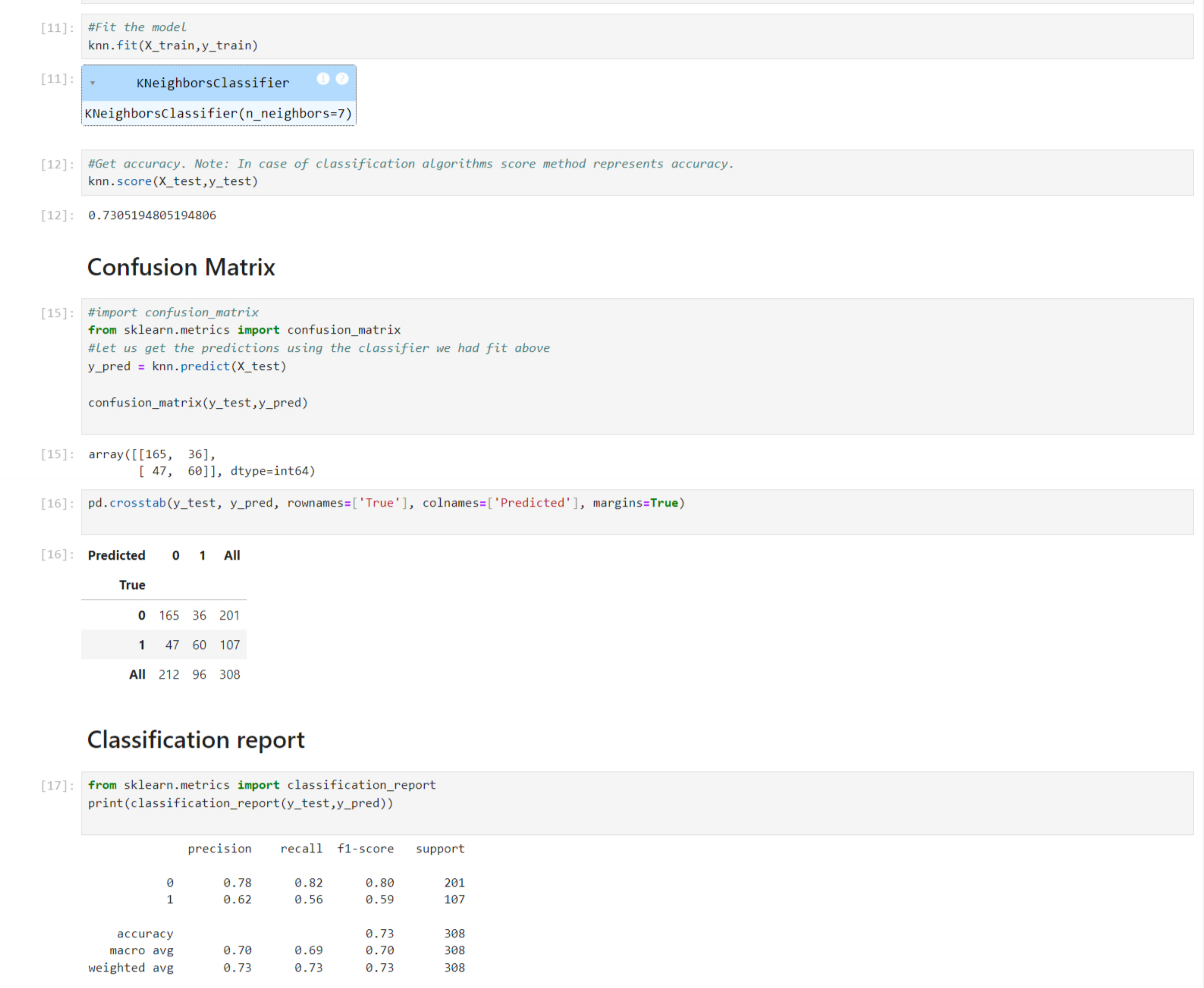
```
#Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

[9]:

```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

[10]:

```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=7)
```



```
[11]: #Fit the model
      knn.fit(X_train,y_train)
```

```
[11]: ▾ KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=7)
```

```
[12]: #Get accuracy. Note: In case of classification algorithms score method represents accuracy.
      knn.score(X_test,y_test)
```

```
[12]: 0.7305194805194806
```

Confusion Matrix

```
[15]: #import confusion_matrix
      from sklearn.metrics import confusion_matrix
      #let us get the predictions using the classifier we had fit above
      y_pred = knn.predict(X_test)

      confusion_matrix(y_test,y_pred)
```

```
[15]: array([[165,  36],
             [ 47,  60]], dtype=int64)
```

```
[16]: pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

[16]:

Predicted	**0**	**1**	**All**
True			
0	165	36	201
1	47	60	107
All	212	96	308

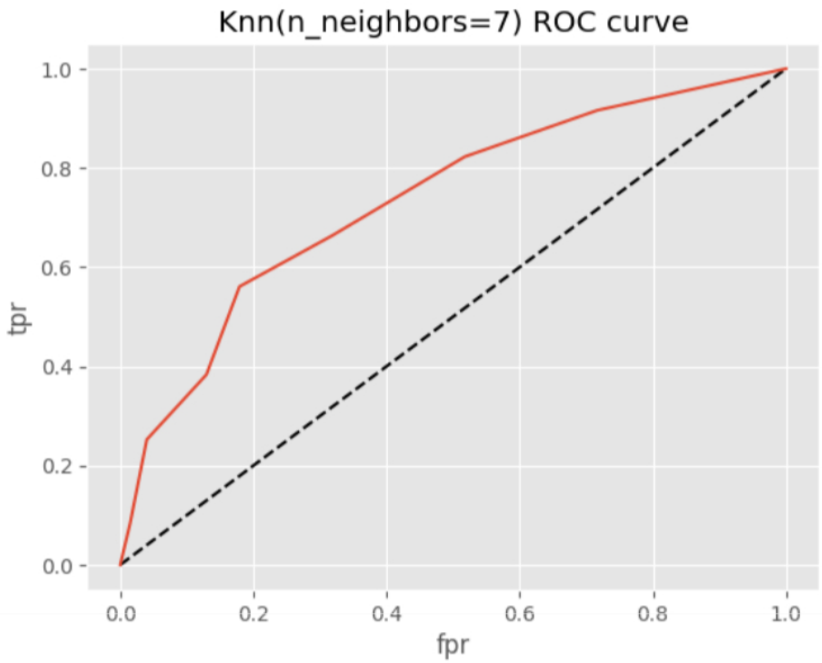
Classification report

```
[17]: from sklearn.metrics import classification_report
      print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       201
           1       0.62      0.56      0.59       107

    accuracy                           0.73       308
   macro avg       0.70      0.69      0.70       308
weighted avg       0.73      0.73      0.73       308
```

ROC curve

[18]:
```
y_pred_proba = knn.predict_proba(X_test)[:,1]
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=7) ROC curve')
plt.show()
```

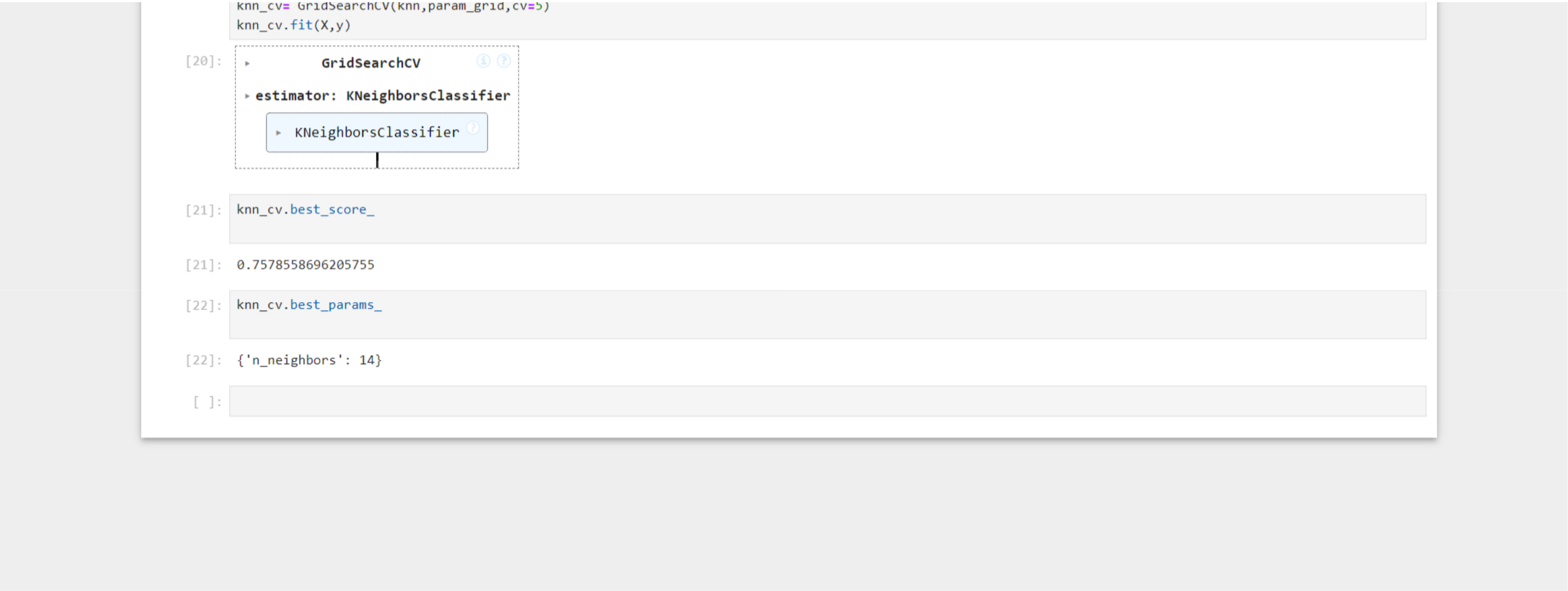


[19]:
```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

[19]:
```
0.7345050448691124
```

Grid Search CV

[20]:
```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,50)}
knn = KNeighborsClassifier()
```



```
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)
```

[20]:

[21]:

```
knn_cv.best_score_
```

[21]:

```
0.7578558696205755
```

[22]:

```
knn_cv.best_params_
```

[22]:

```
{'n_neighbors': 14}
```

[ ]: