Ex.2 31-7-2025

Implement a classifier

Aim:-

To implement a classifier using a open-source dataset & evaluate its performance

Objective:

* To load and explore the Iris dataset by implementing the K-Nearest Neighbors (KNN) algorithm ~~using~~ using scikit-learn in Python

* Preprocess the dataset using feature scaling
* Split the data into training and testing sets
* Train a KNN Classifier
* Evaluate the model using metrics such as confusion matrix, classification report and accuracy

Pseudocode:

```
* Import necessary libraries (pandas, numpy, sklearn)
2. Load the Iris dataset using scikit-learn
3. Extract features (x) and labels (y) from the dataset
```

4. Display dataset details (shape, target names, features names.)

5. Split the dataset into training and testing sets (80% train, 20% test).

6. Scale the features using Standard Scaler:
   - Fit scaler on training data and transform it.
   - Transform test data using the same Scaler.

7. Initialize the KNN Classifier with k=3

8. Train the KNN model using the scaled training data

9. Predict labels for the test data using the trained model.

10. evalute the model:
    - Print confusion matrix
    - Print classification report
    - Calculate and print overall accuracy

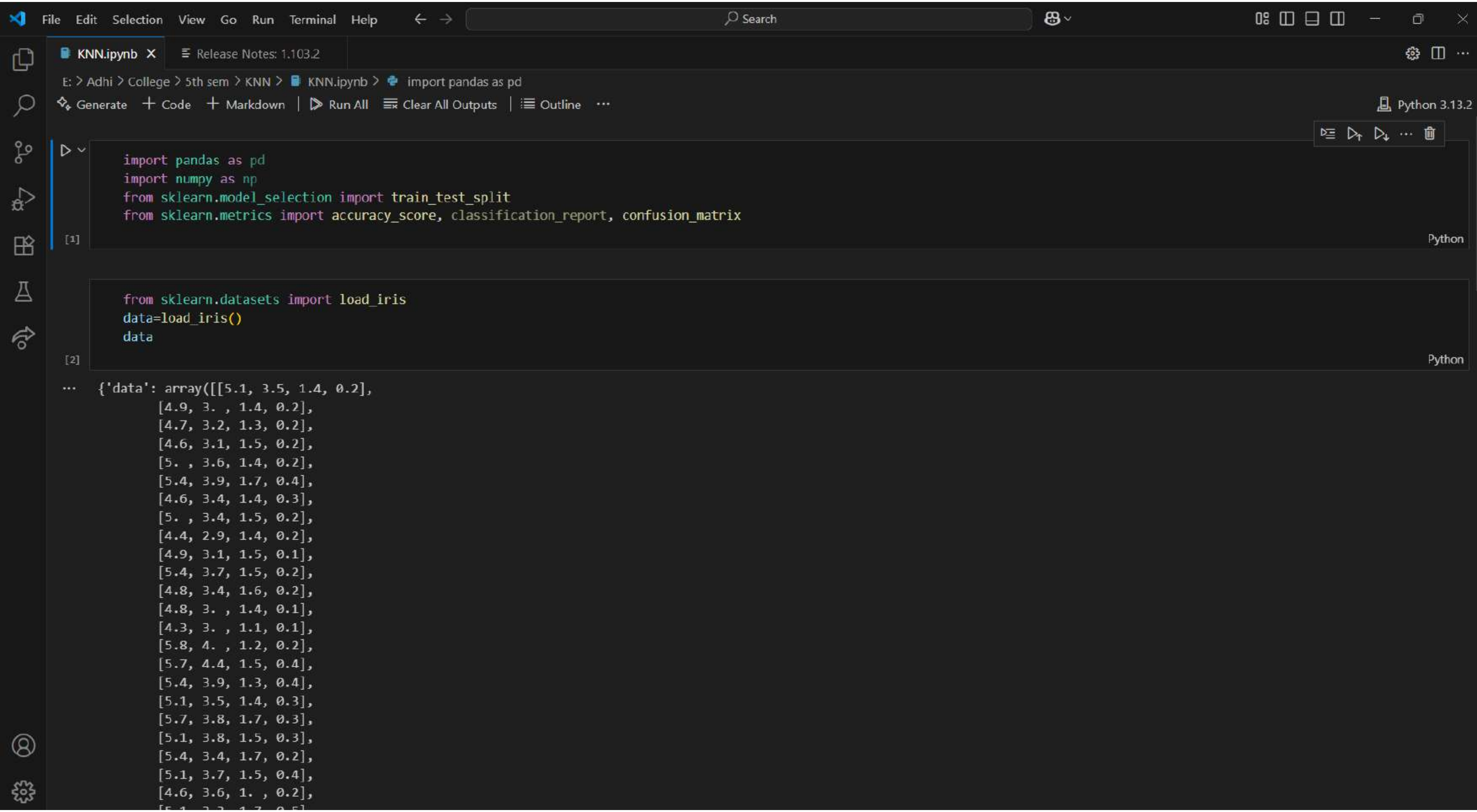
11. End

Observation

1. Dataset Used: The Iris dataset
2. Data splitting: The dataset is split into training and testing sets using an 80-20 split with train_test_split(), ensuring reproducibility with random_state=42.
3. Feature Scaling: StandardScaler is applied to normalize the features
4. Model training: A KNN classifier with 3 neighbors (n_neighbors=3) is trained on the scaled training data
5. Model Evaluation: The model's performance is evaluated using a confusion matrix and acarcy score.

Result:-

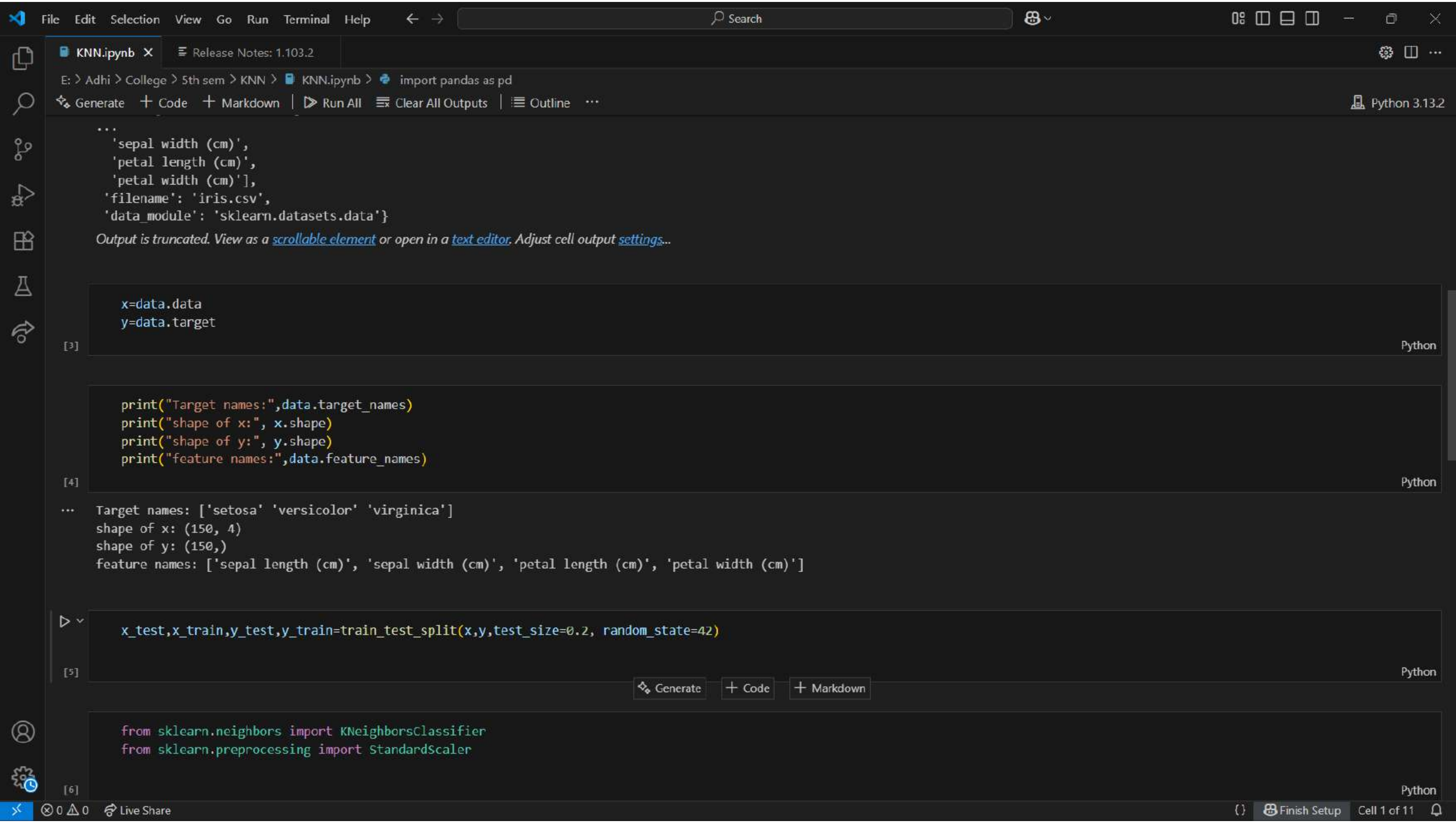
The code was implemented successfully.



```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```python
from sklearn.datasets import load_iris
data=load_iris()
data
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
```



```
...
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
x=data.data
y=data.target
```

[3] Python

```
print("Target names:",data.target_names)
print("shape of x:", x.shape)
print("shape of y:", y.shape)
print("feature names:",data.feature_names)
```

[4] Python

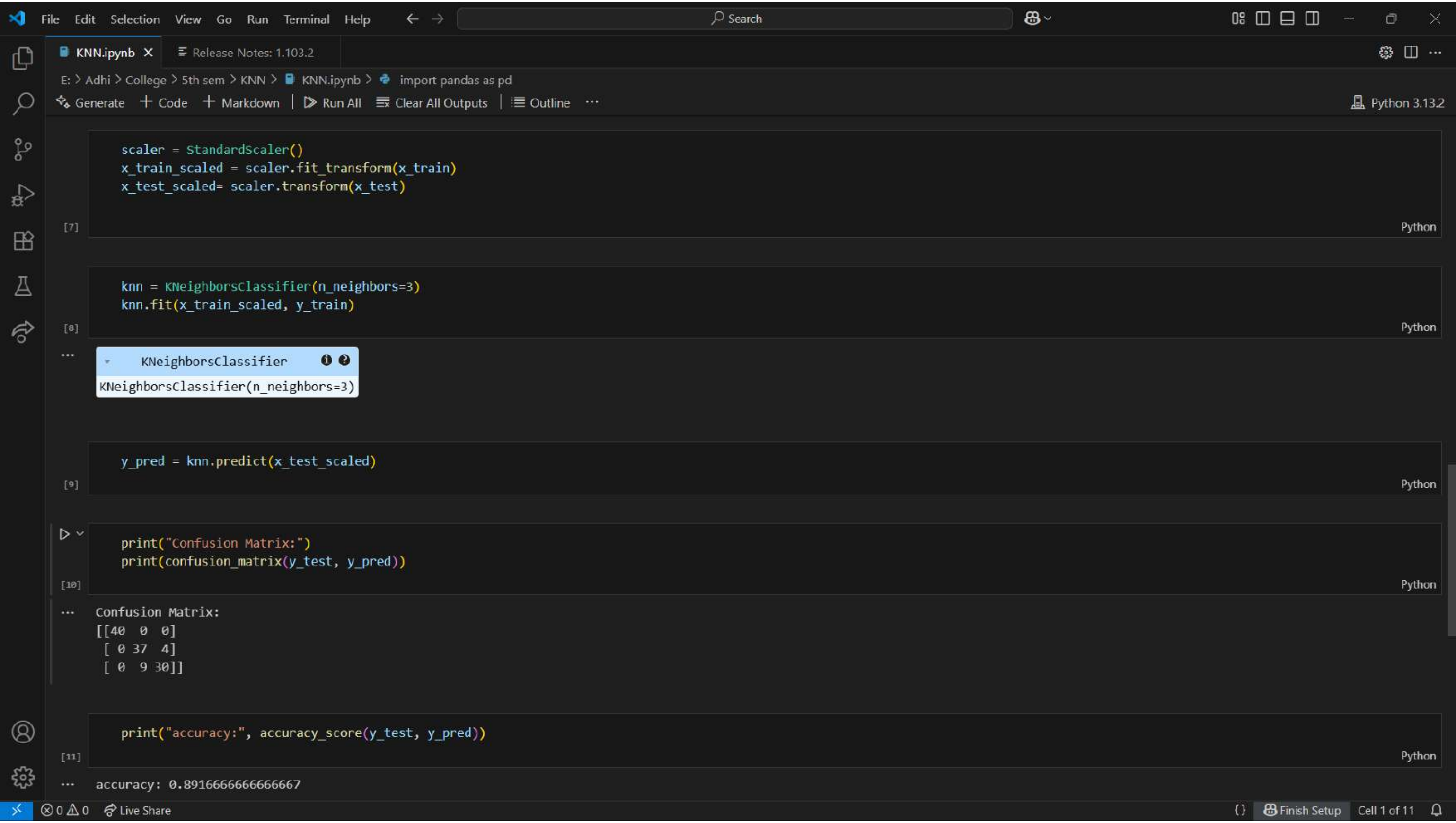
```
Target names: ['setosa' 'versicolor' 'virginica']
shape of x: (150, 4)
shape of y: (150,)
feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size=0.2, random_state=42)
```

[5] Python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

[6] Python



```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled= scaler.transform(x_test)
```

[7] Python

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train_scaled, y_train)
```

[8] Python

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = knn.predict(x_test_scaled)
```

[9] Python

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

[10] Python

```
Confusion Matrix:
[[40  0  0]
 [ 0 37  4]
 [ 0  9 30]]
```

```
print("accuracy:", accuracy_score(y_test, y_pred))
```

[11] Python

```
accuracy: 0.8916666666666667
```