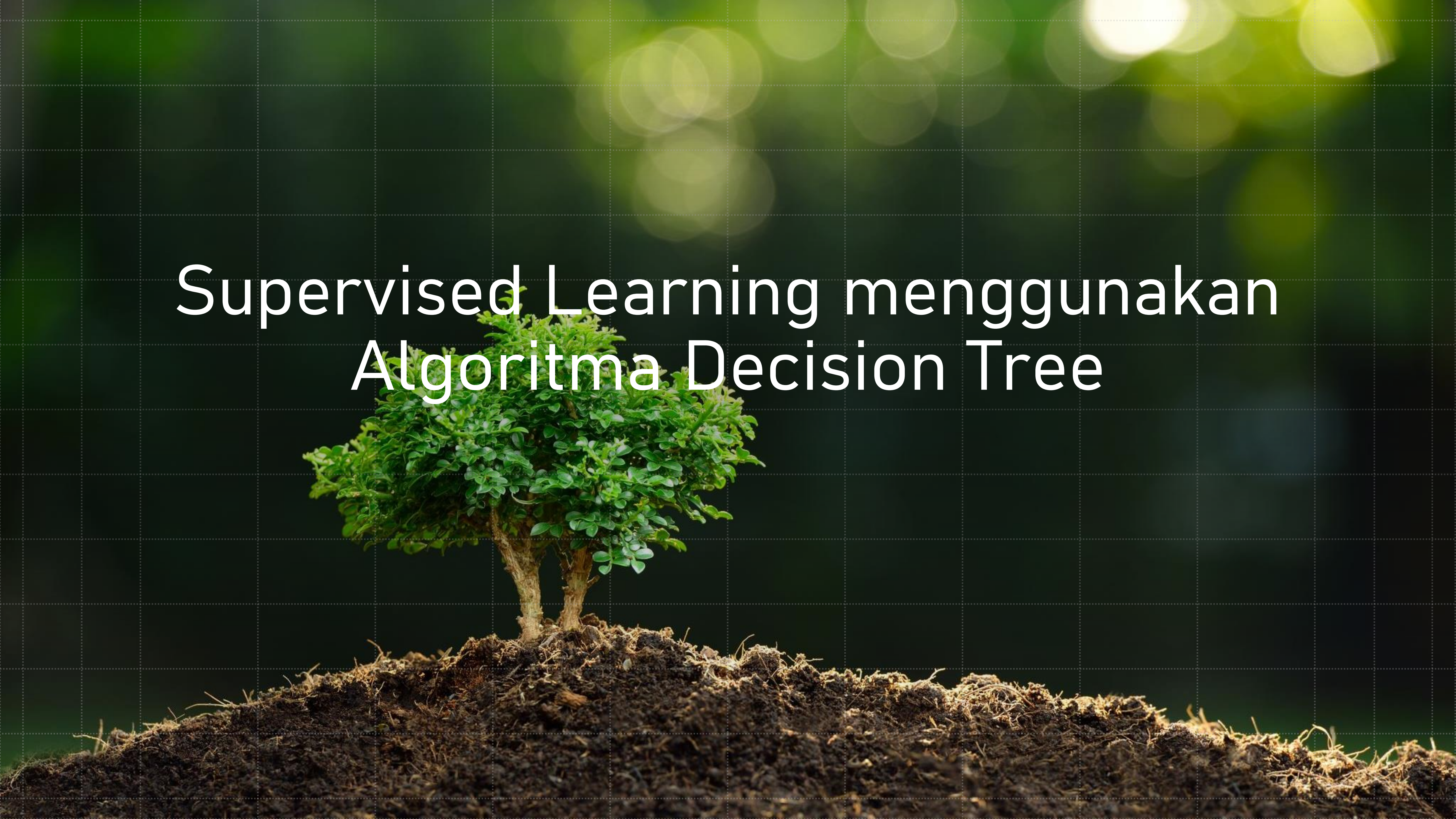
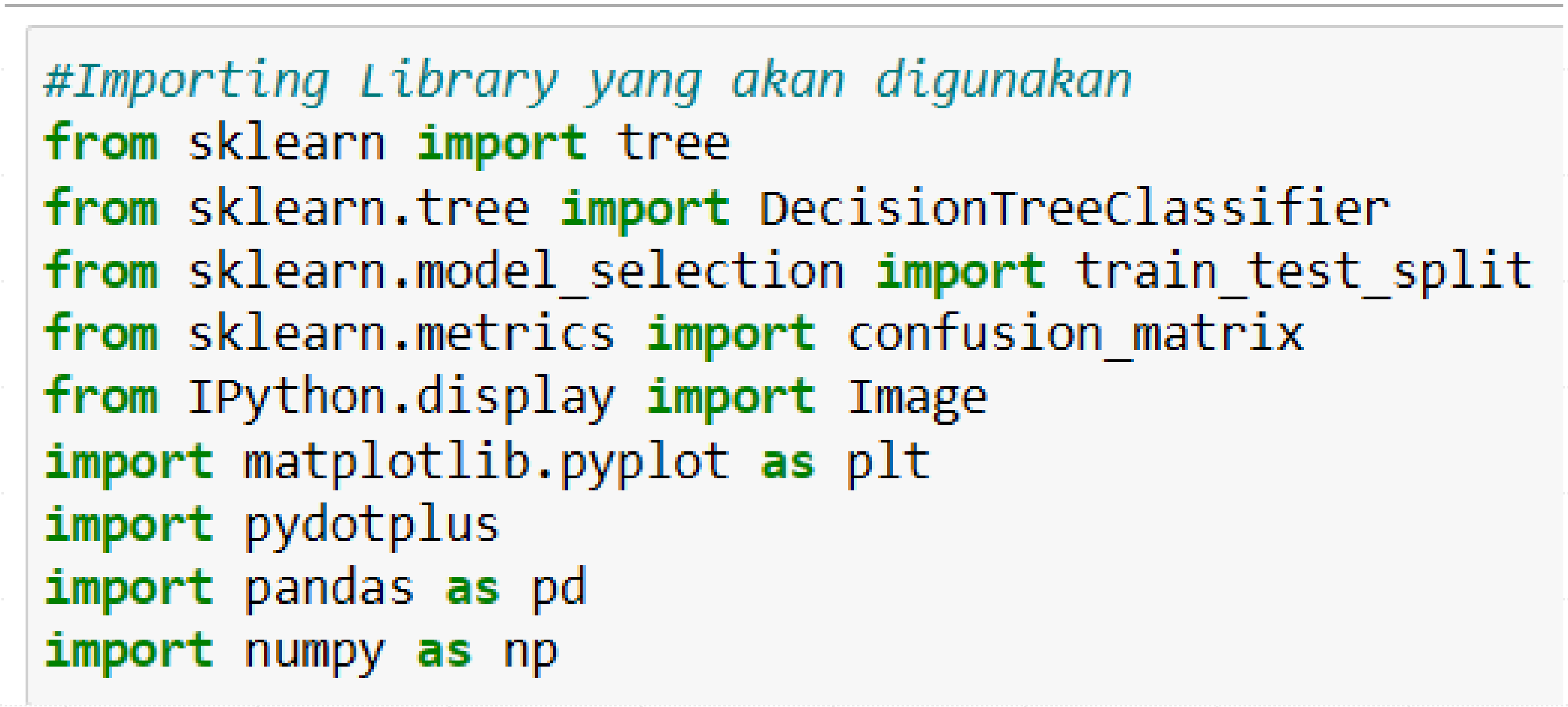
Supervised Learning Menggunakan Algoritma Descision Tree dan Unsupervised Learning Menggunakan K-Means

CODING PYTHON



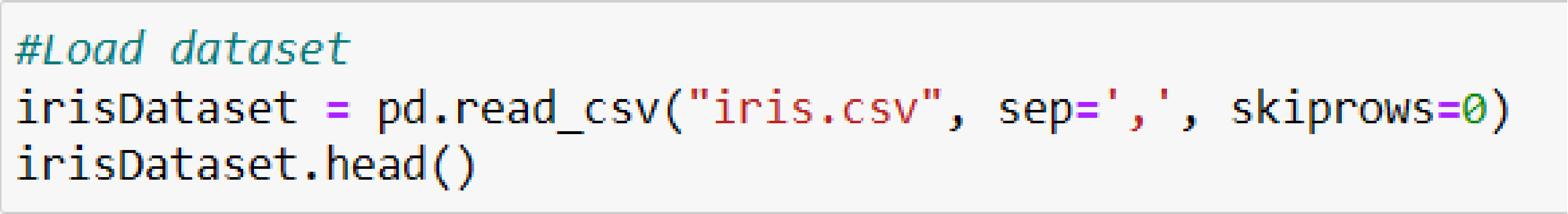
Supervised Learning menggunakan Algoritma Decision Tree

Import Library



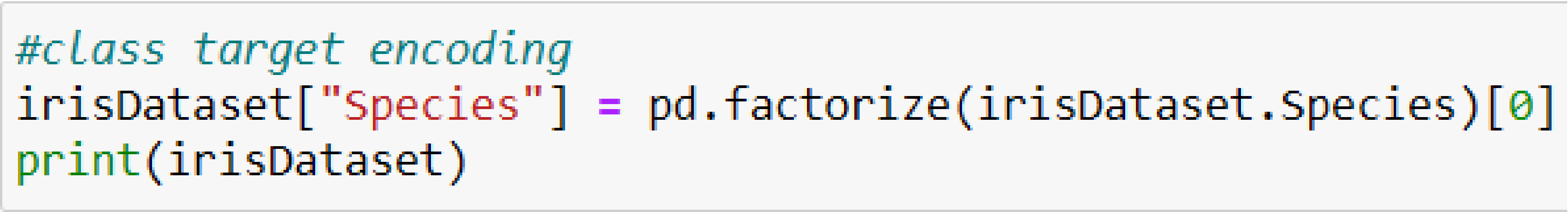
```python
#Importing Library yang akan digunakan
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from IPython.display import Image
import matplotlib.pyplot as plt
import pydotplus
import pandas as pd
import numpy as np
```

Load dataset



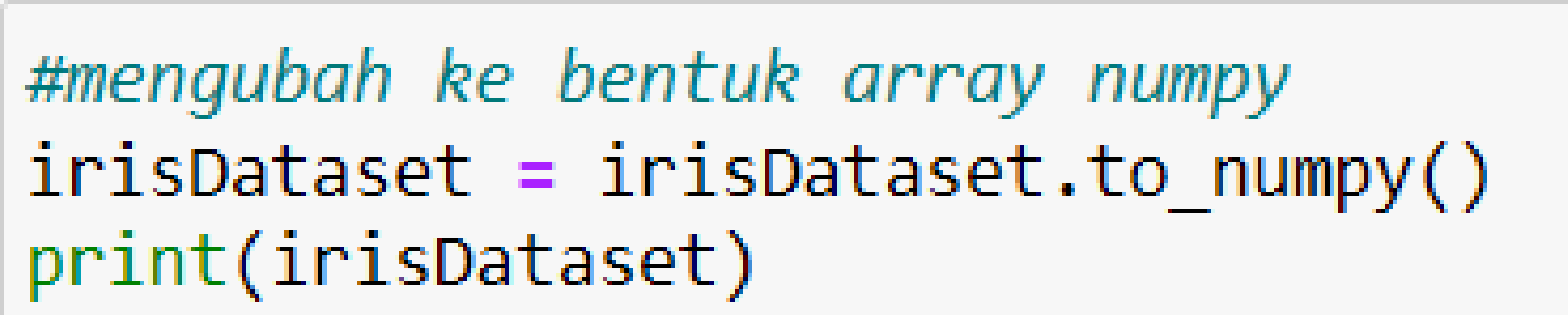
```
#Load dataset
irisDataset = pd.read_csv("iris.csv", sep=',', skiprows=0)
irisDataset.head()
```

Penentuan target sebagai atribut Klasifikasi



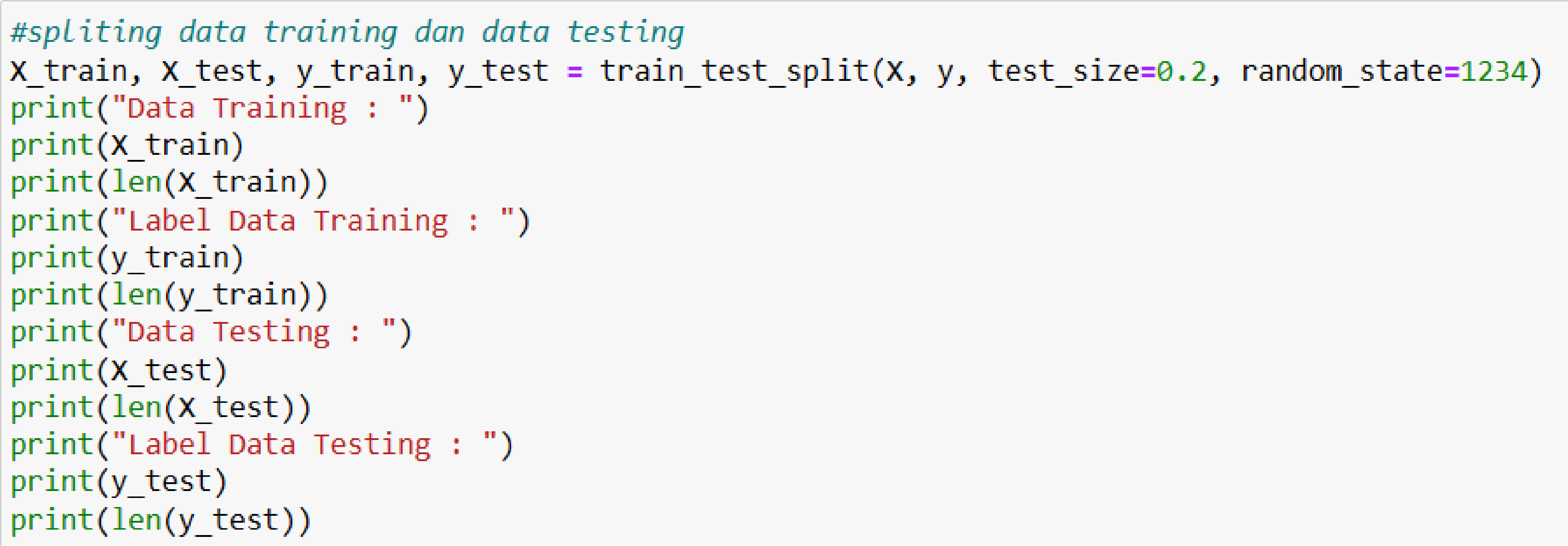
```
#class target encoding
irisDataset["Species"] = pd.factorize(irisDataset.Species)[0]
print(irisDataset)
```

Mengubah ke dalam bentuk array



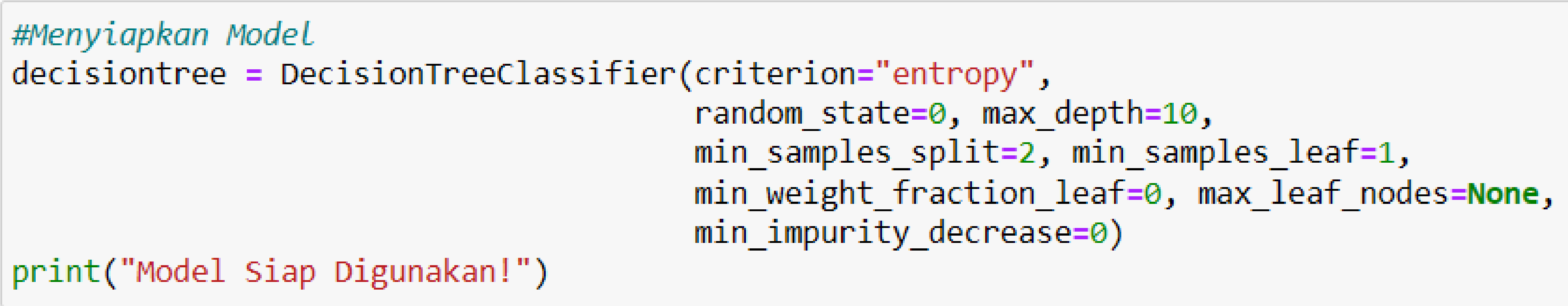
```
#mengubah ke bentuk array numpy
irisDataset = irisDataset.to_numpy()
print(irisDataset)
```

Spliting data training dan data testing



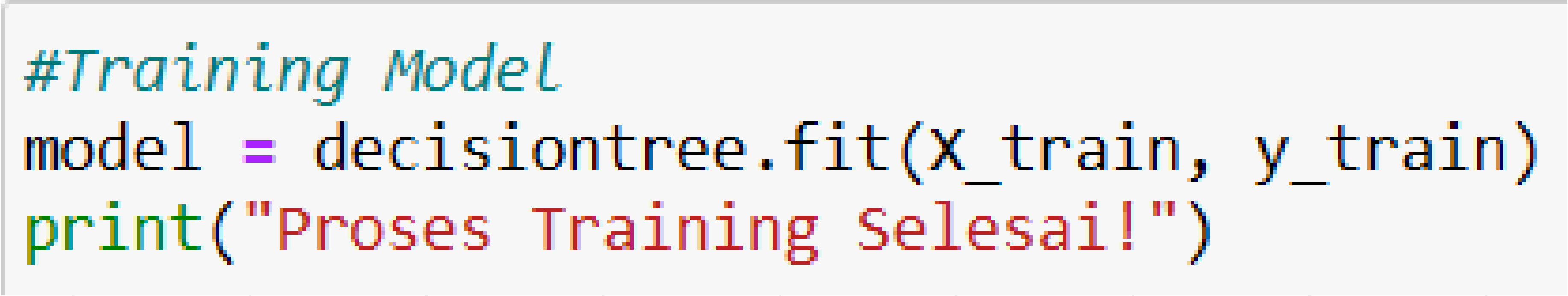
```
#spliting data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
print("Data Training : ")
print(X_train)
print(len(X_train))
print("Label Data Training : ")
print(y_train)
print(len(y_train))
print("Data Testing : ")
print(X_test)
print(len(X_test))
print("Label Data Testing : ")
print(y_test)
print(len(y_test))
```

Pemodelan menggunakan decision tree



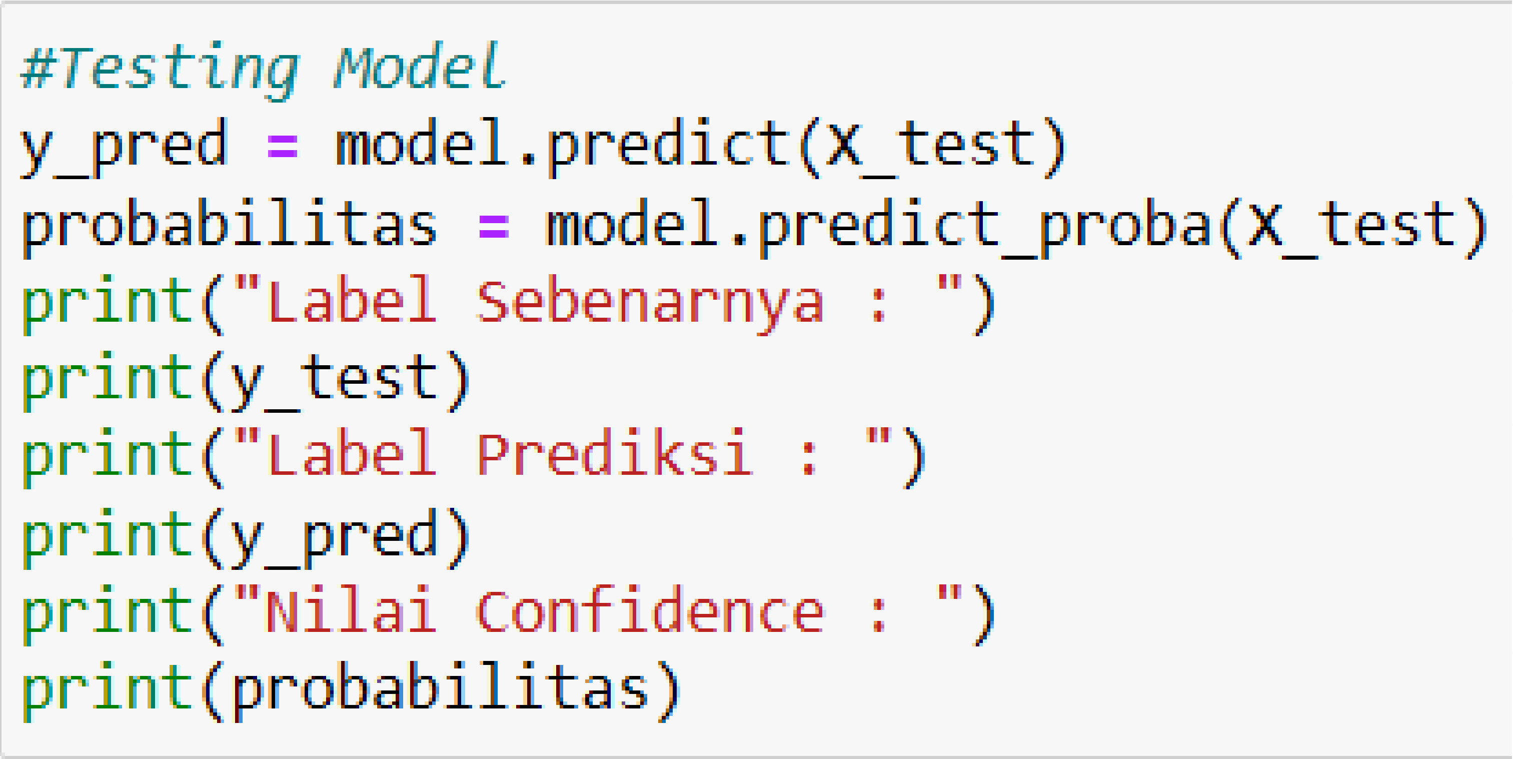
```
#Menyiapkan Model
decisiontree = DecisionTreeClassifier(criterion="entropy",
                                      random_state=0, max_depth=10,
                                      min_samples_split=2, min_samples_leaf=1,
                                      min_weight_fraction_leaf=0, max_leaf_nodes=None,
                                      min_impurity_decrease=0)
print("Model Siap Digunakan!")
```

Proses training pemodelan decision tree



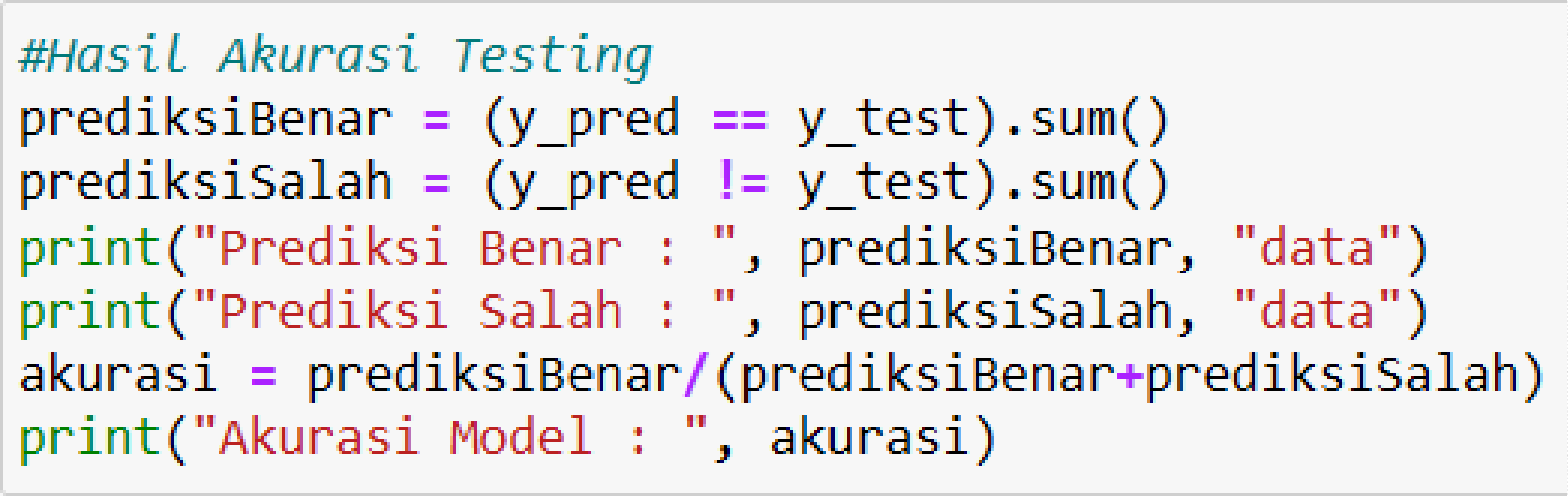
```
#Training Model
model = decisiontree.fit(X_train, y_train)
print("Proses Training Selesai!")
```

Testing pemodelan decision tree



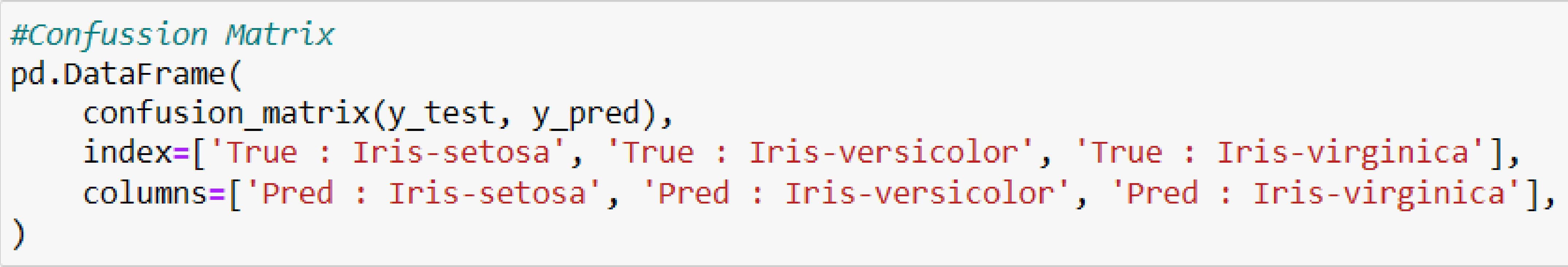
```
#Testing Model
y_pred = model.predict(X_test)
probabilitas = model.predict_proba(X_test)
print("Label Sebenarnya : ")
print(y_test)
print("Label Prediksi : ")
print(y_pred)
print("Nilai Confidence : ")
print(probabilitas)
```

Perhitungan nilai akurasi



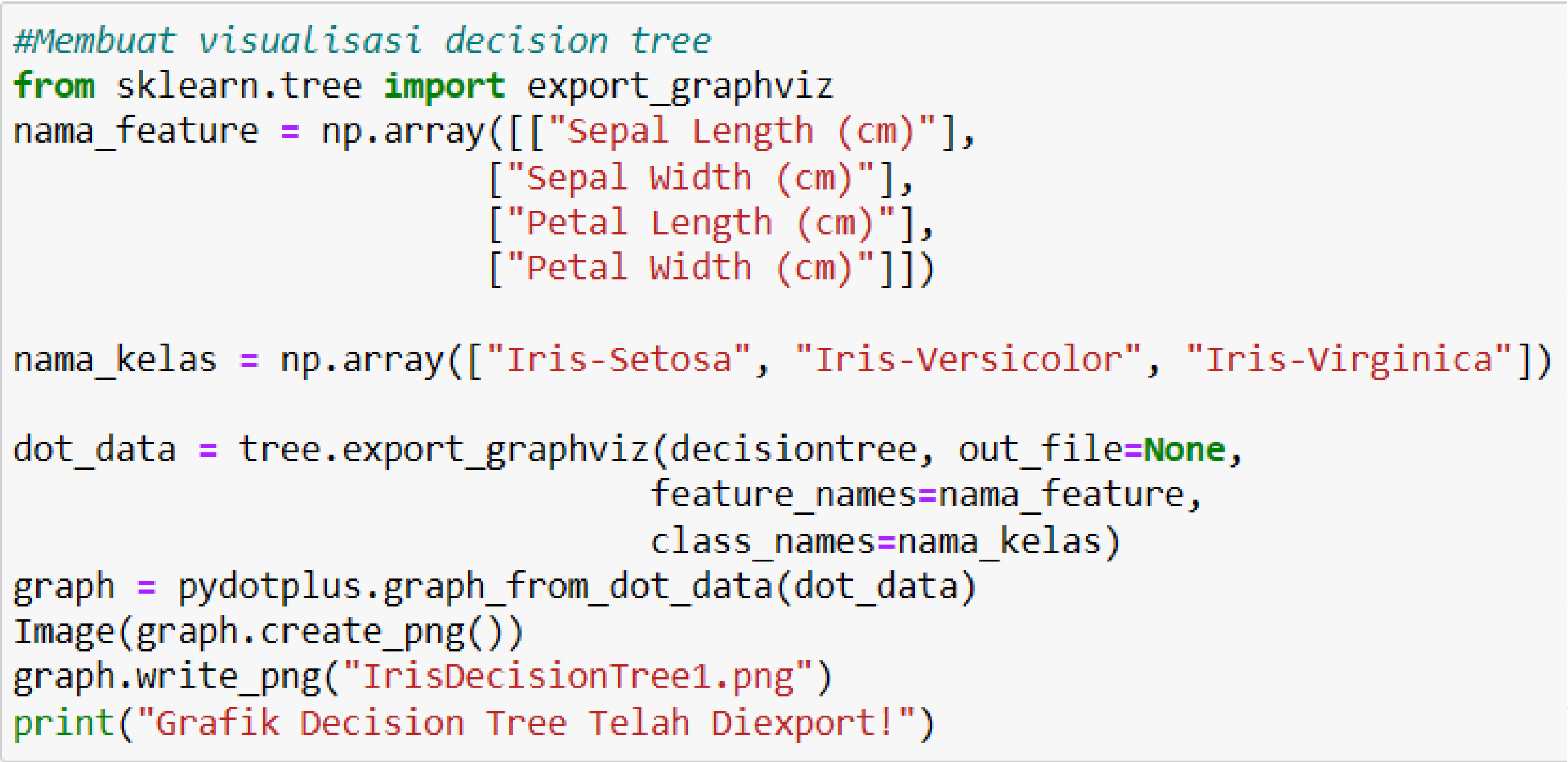
```
#Hasil Akurasi Testing
prediksiBenar = (y_pred == y_test).sum()
prediksiSalah = (y_pred != y_test).sum()
print("Prediksi Benar : ", prediksiBenar, "data")
print("Prediksi Salah : ", prediksiSalah, "data")
akurasi = prediksiBenar/(prediksiBenar+prediksiSalah)
print("Akurasi Model : ", akurasi)
```

Pembuatan tabel confusion matrix

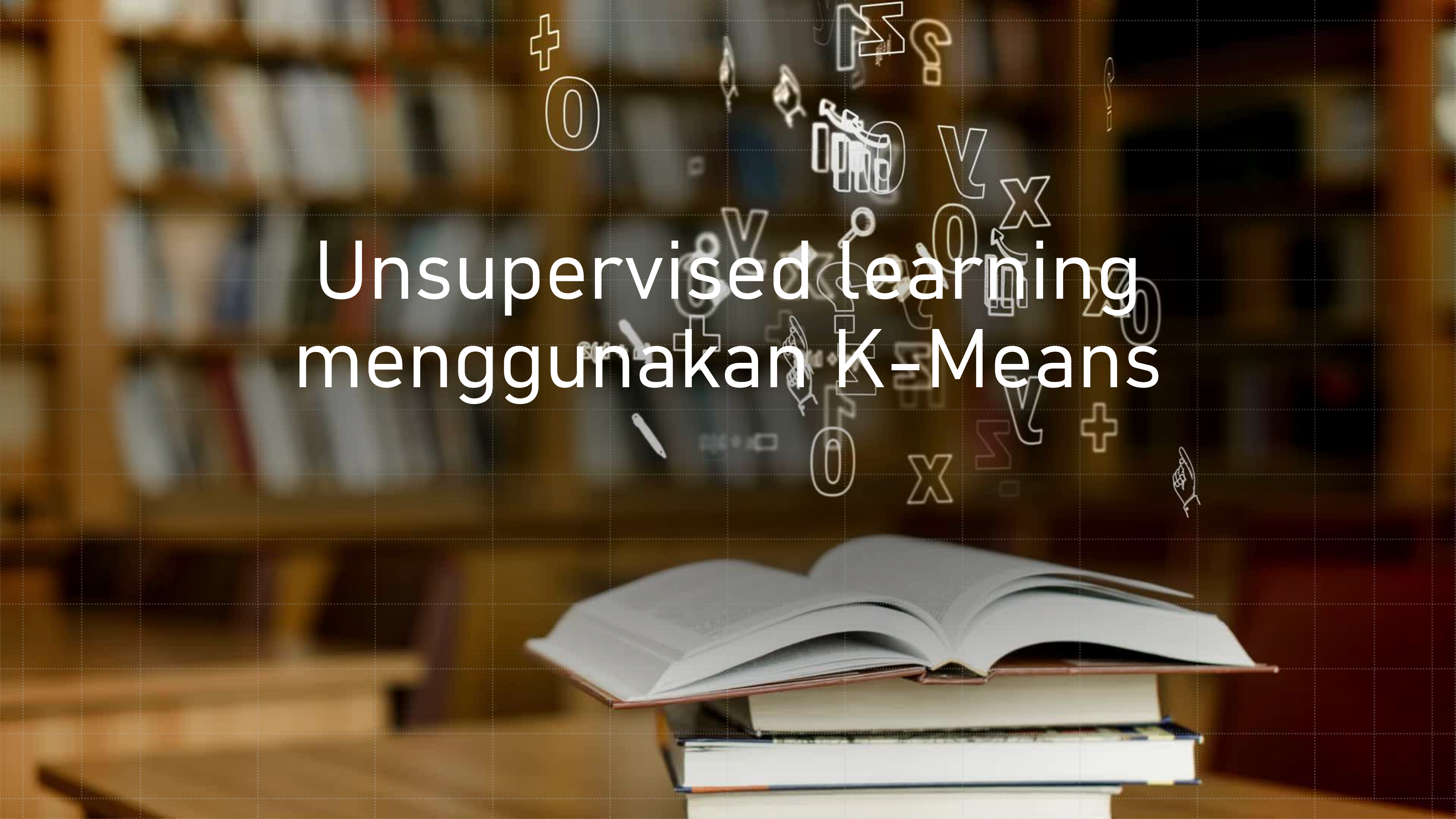


```
#Confussion Matrix
pd.DataFrame(
    confusion_matrix(y_test, y_pred),
    index=['True : Iris-setosa', 'True : Iris-versicolor', 'True : Iris-virginica'],
    columns=['Pred : Iris-setosa', 'Pred : Iris-versicolor', 'Pred : Iris-virginica'],
)
```

Generate pohon keputusan

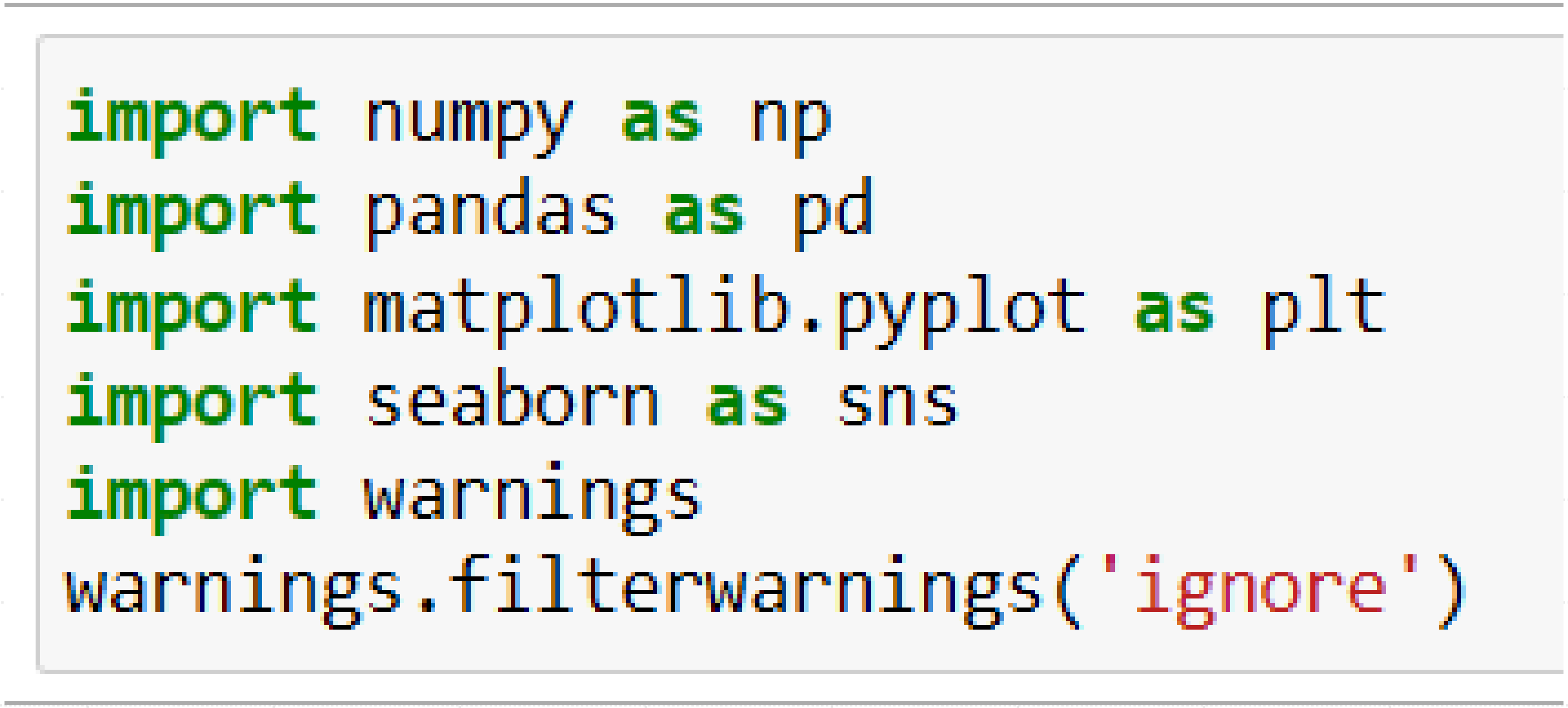


```
#Membuat visualisasi decision tree
from sklearn.tree import export_graphviz
nama_feature = np.array([["Sepal Length (cm)"],
                       ["Sepal Width (cm)"],
                       ["Petal Length (cm)"],
                       ["Petal Width (cm)"]])

nama_kelas = np.array(["Iris-Setosa", "Iris-Versicolor", "Iris-Virginica"])

dot_data = tree.export_graphviz(decisiontree, out_file=None,
                               feature_names=nama_feature,
                               class_names=nama_kelas)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
graph.write_png("IrisDecisionTree1.png")
print("Grafik Decision Tree Telah Diexport!")
```



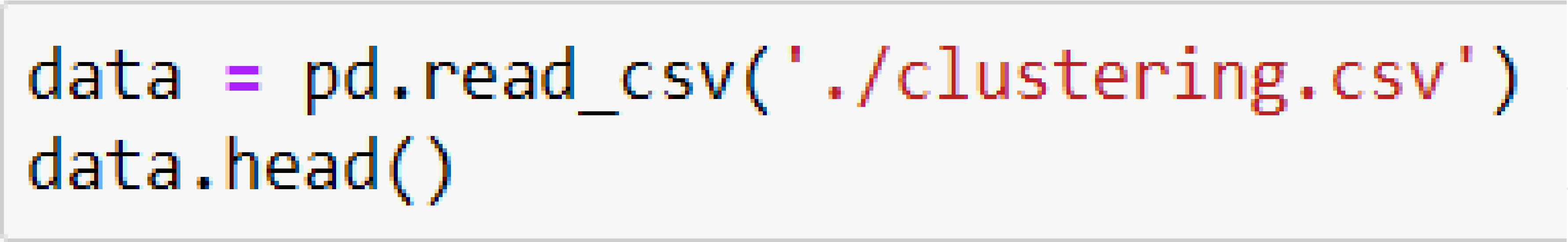
Unsupervised learning menggunakan K-Means

Import library



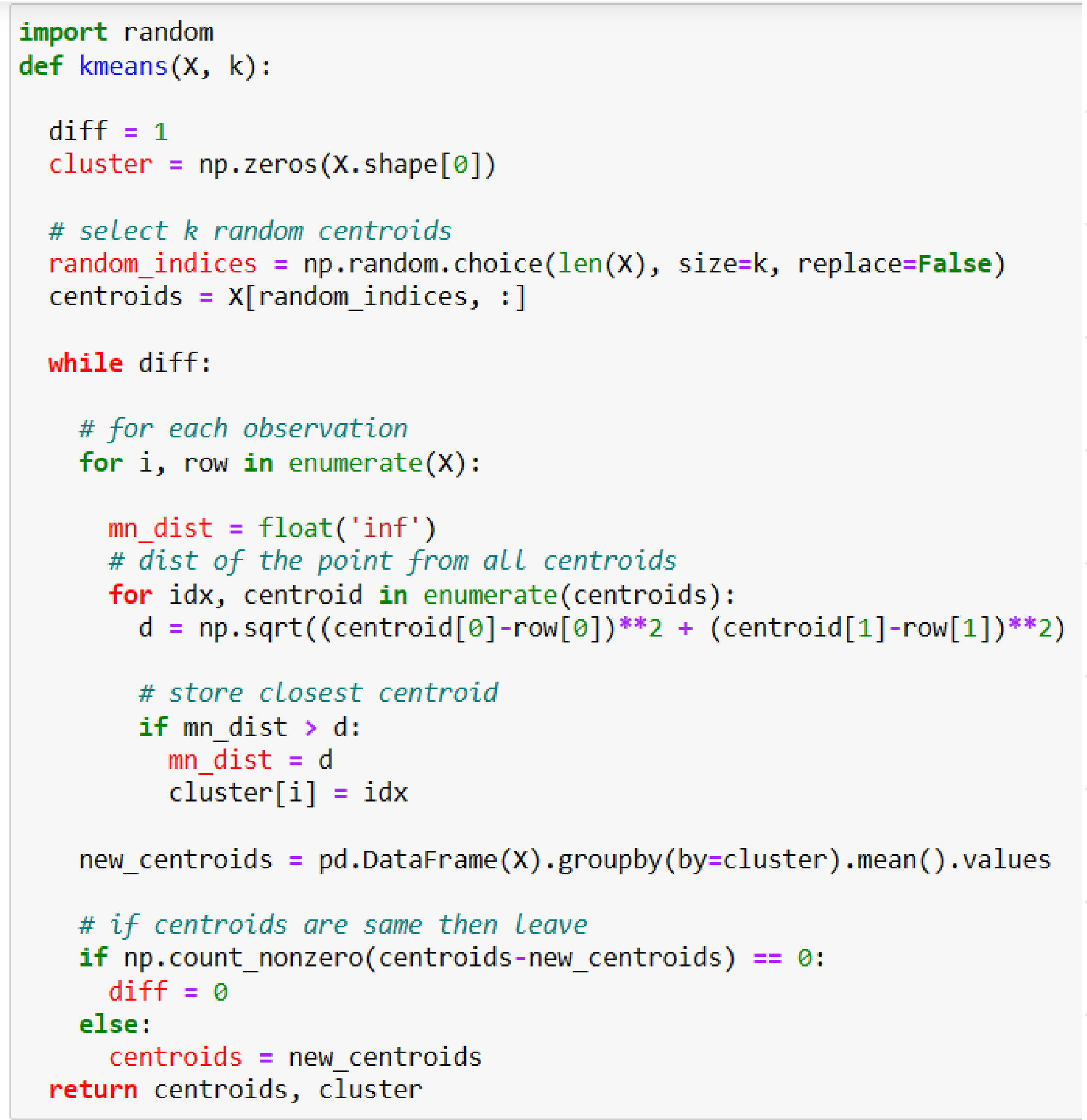
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Import data



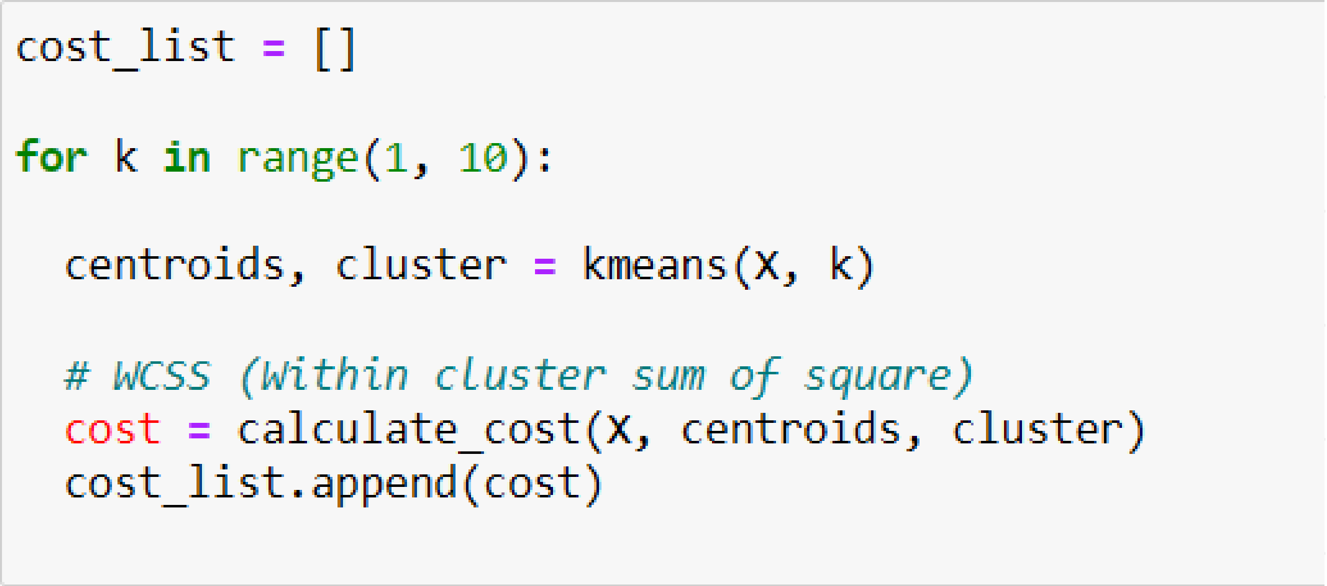
```
data = pd.read_csv('./clustering.csv')
data.head()
```

Perhitungan nilai K terbaik

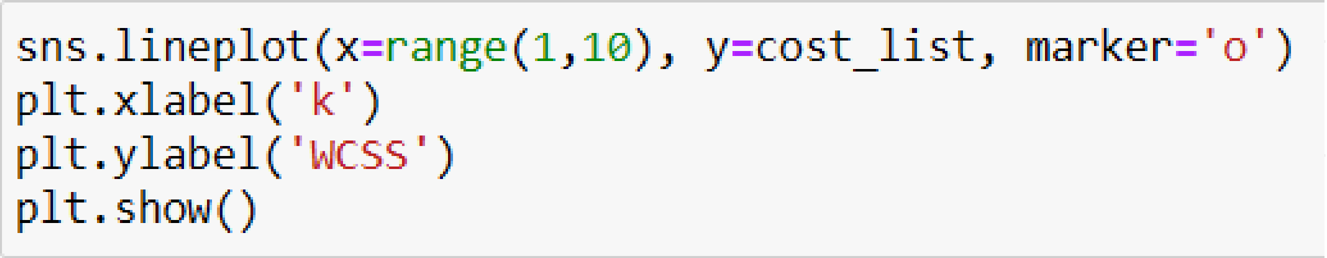


```
import random
def kmeans(X, k):

  diff = 1
  cluster = np.zeros(X.shape[0])

  # select k random centroids
  random_indices = np.random.choice(len(X), size=k, replace=False)
  centroids = X[random_indices, :]

  while diff:

    # for each observation
    for i, row in enumerate(X):

      mn_dist = float('inf')
      # dist of the point from all centroids
      for idx, centroid in enumerate(centroids):
        d = np.sqrt((centroid[0]-row[0])**2 + (centroid[1]-row[1])**2)

        # store closest centroid
        if mn_dist > d:
          mn_dist = d
          cluster[i] = idx

    new_centroids = pd.DataFrame(X).groupby(by=cluster).mean().values

    # if centroids are same then leave
    if np.count_nonzero(centroids-new_centroids) == 0:
      diff = 0
    else:
      centroids = new_centroids
  return centroids, cluster
```

Menampilkan grafik nilai K terbaik menggunakan elbow method

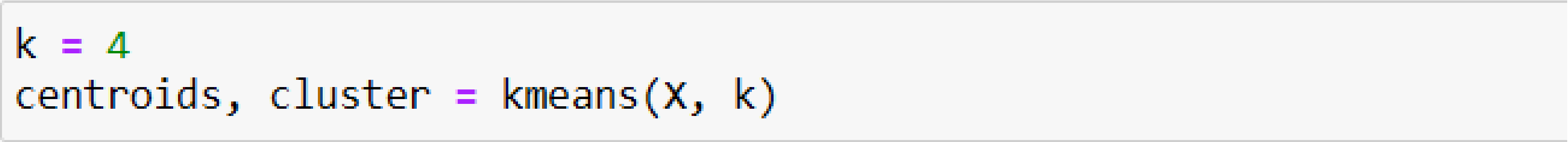


```
cost_list = []

for k in range(1, 10):

  centroids, cluster = kmeans(X, k)

  # WCSS (Within cluster sum of square)
  cost = calculate_cost(X, centroids, cluster)
  cost_list.append(cost)
```

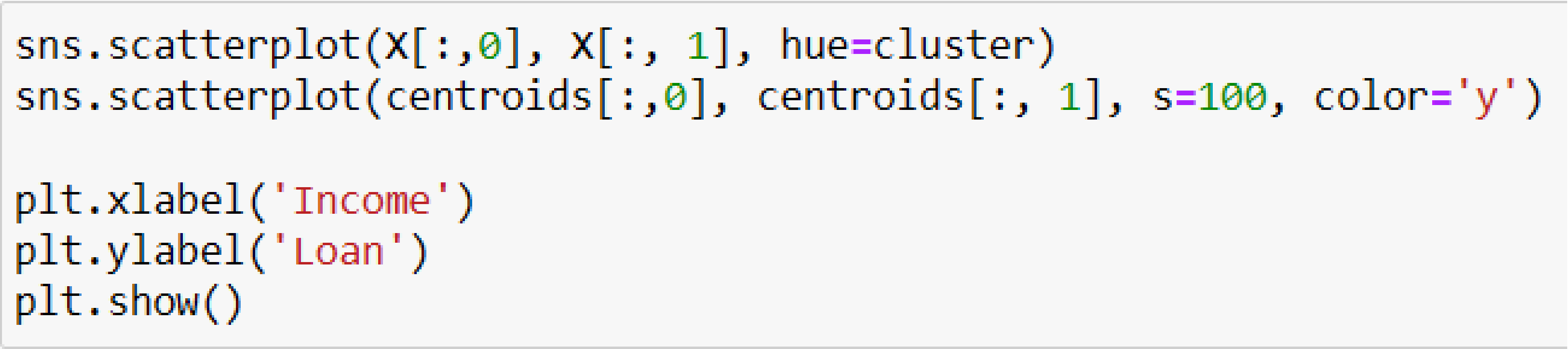


```
sns.lineplot(x=range(1,10), y=cost_list, marker='o')
plt.xlabel('k')
plt.ylabel('WCSS')
plt.show()
```

Pengujian nilai K dan pembuatan grafik klastering



```
k = 4
centroids, cluster = kmeans(X, k)
```



```
sns.scatterplot(X[:,0], X[:, 1], hue=cluster)
sns.scatterplot(centroids[:,0], centroids[:, 1], s=100, color='y')

plt.xlabel('Income')
plt.ylabel('Loan')
plt.show()
```