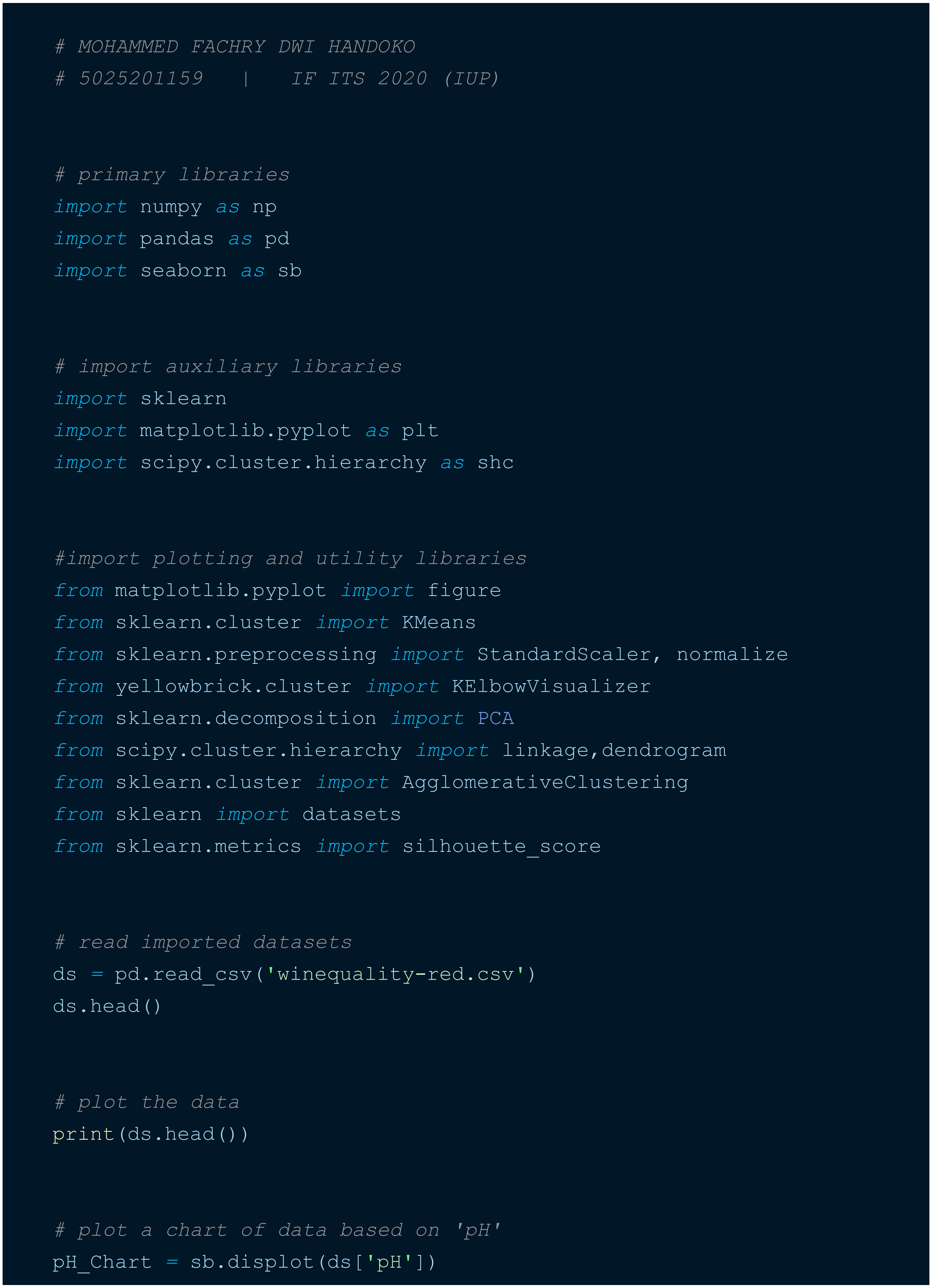
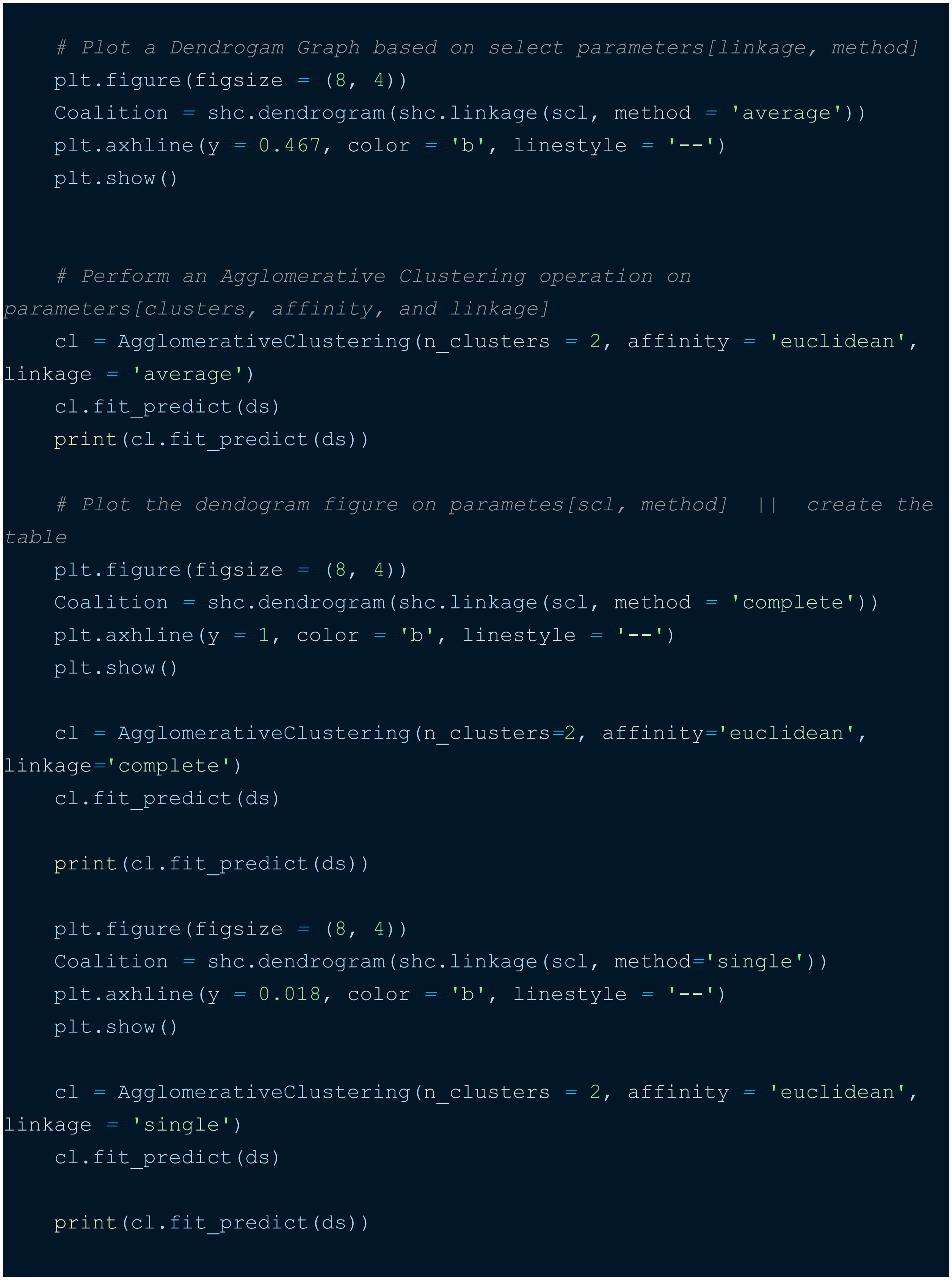


```
# MOHAMMED FACHRY DWI HANDOKO
# 5025201159   |   IF ITS 2020 (IUP)


# primary libraries
import numpy as np
import pandas as pd
import seaborn as sb


# import auxiliary libraries
import sklearn
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as shc


#import plotting and utility libraries
from matplotlib.pyplot import figure
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, normalize
from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import linkage,dendrogram
from sklearn.cluster import AgglomerativeClustering
from sklearn import datasets
from sklearn.metrics import silhouette_score


# read imported datasets
ds = pd.read_csv('winequality-red.csv')
ds.head()


# plot the data
print(ds.head())


# plot a chart of data based on 'pH'
pH_Chart = sb.displot(ds['pH'])
```

```
    # build data for the pair plot
    visualization = sb.pairplot(ds)
    plt.show()
    print(visualization)
    plt.figure(figsize = (13, 9))

    ds = ds.drop('pH', axis = 1)
    ds.columns


    # Loop over the datasets to get the K-Mean value
    wcss = []

    for i in range (1, 7):
        kmeans = KMeans(n_clusters = i, init = 'k-means++')
        kmeans.fit(ds)
        wcss.append(kmeans.inertia_)

    plt.plot(range(1, 11), wcss)
    plt.title('Elbow Methods Graphics')
    plt.xlabel('Cluster')
    plt.ylabel('WCSS')
    plt.show()


    # Render the graphic to represent the K-Mean
    model = KMeans()
    visible = KElbowVisualizer(model, k=(1,11), timings = False)
    visible.fit(ds)
    visible.show()


    # Loop over the K-Means to get the silhouette score
    for i in range(2,11):
        kmeans = KMeans(n_clusters=i,max_iter=100)
        kmeans.fit(ds)
        SilhoutteScore = silhouette_score(ds, kmeans.labels_,
metric='euclidean')
```

```
        print("{} silhouette score : {}".format(i,SilhoutteScore))


    # Render the graphic for silhouette score from i=2 => 11
    coefOfSilhouette = []
    for i in range(2,11):
        kmeans = KMeans(n_clusters=i, max_iter=100)
        kmeans.fit(ds)
        SilhoutteScore = silhouette_score(ds, kmeans.labels_)
        coefOfSilhouette.append(SilhoutteScore)


    # Plot the coefficient of the silhouette score
    plt.plot(range(2,11), coefOfSilhouette)
    plt.xticks(range(2,11))
    plt.xlabel("number of clusters")
    plt.ylabel("Coefficient of Silhouette")
    plt.show()


    # Loop over the values and build the legend / markers for the
displayed data
    bca = PCA()
    X = bca.fit_transform(ds)
    kmeans = KMeans(n_clusters = 3)
    label = kmeans.fit_predict(X)
    graphics = np.unique(label)

    for i in graphics:
        plt.scatter(X[label==i,0], X[label==i,1], label=i, s=20)

    plt.legend()
    plt.title('Post-Drop pH')
    plt.show()

    scl = normalize(ds)
    scl = pd.DataFrame(scl, columns = ds.columns)
    scl.head()
    print(scl.head())
```



```
    # Plot a Dendrogam Graph based on select parameters[linkage, method]
    plt.figure(figsize = (8, 4))
    Coalition = shc.dendrogram(shc.linkage(scl, method = 'average'))
    plt.axhline(y = 0.467, color = 'b', linestyle = '--')
    plt.show()


    # Perform an Agglomerative Clustering operation on
parameters[clusters, affinity, and linkage]
    cl = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean',
linkage = 'average')
    cl.fit_predict(ds)
    print(cl.fit_predict(ds))

    # Plot the dendogram figure on parametes[scl, method]  ||  create the
table
    plt.figure(figsize = (8, 4))
    Coalition = shc.dendrogram(shc.linkage(scl, method = 'complete'))
    plt.axhline(y = 1, color = 'b', linestyle = '--')
    plt.show()

    cl = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
linkage='complete')
    cl.fit_predict(ds)

    print(cl.fit_predict(ds))

    plt.figure(figsize = (8, 4))
    Coalition = shc.dendrogram(shc.linkage(scl, method='single'))
    plt.axhline(y = 0.018, color = 'b', linestyle = '--')
    plt.show()

    cl = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean',
linkage = 'single')
    cl.fit_predict(ds)

    print(cl.fit_predict(ds))
```