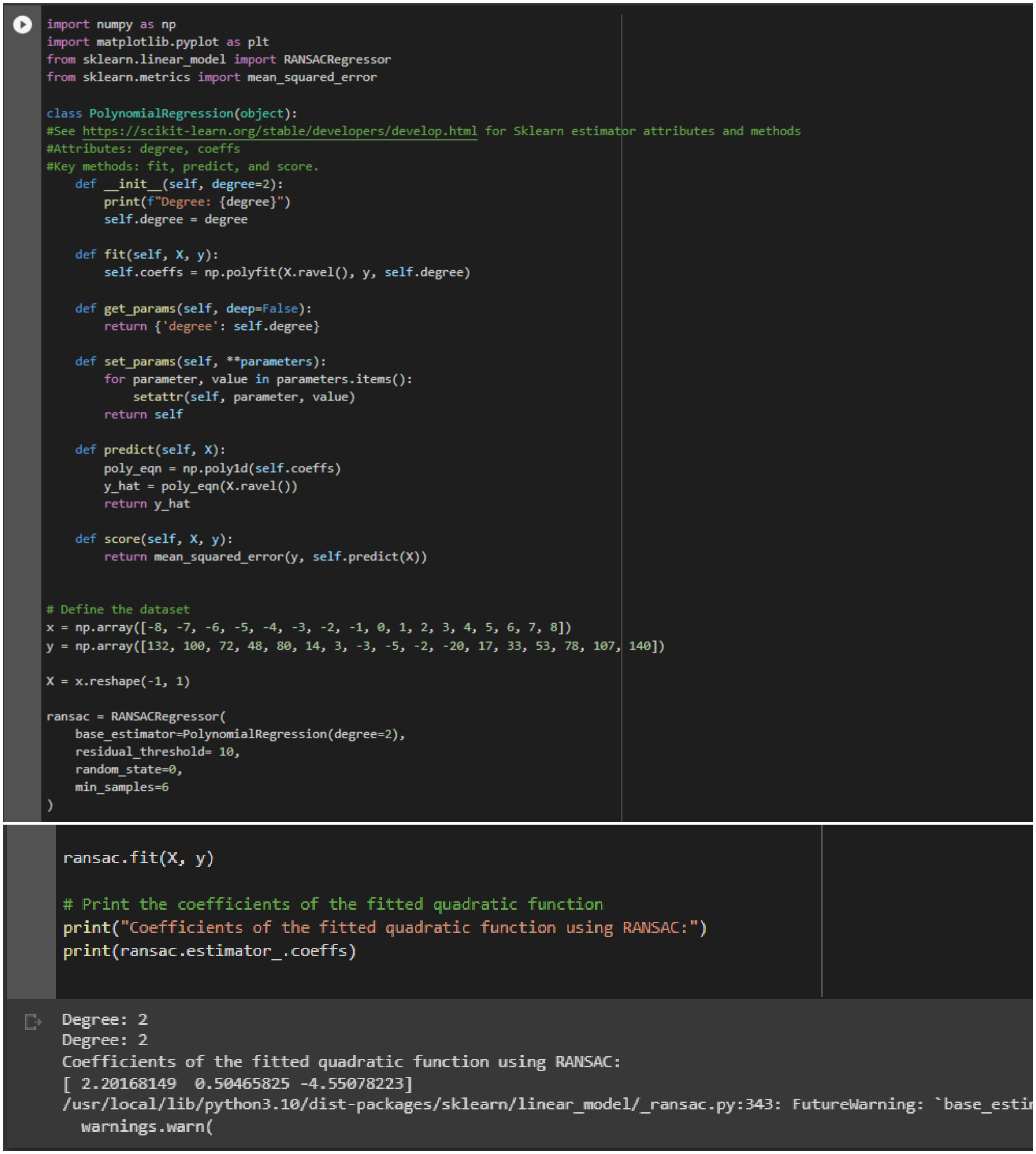
Kunnithi Treecharoensomboon ID 63070503461

CPE383 Machine Learning: Quiz 9

1. 20 points. 1 hour. RANSAC Regression. Use RANSAC to find a, b, c for the following dataset where points (xi, yi) are discrete samples from a function f(x) = ax2 + bx + c with 2 outliers.

   Hint: You should get a, b, and c close to 2.2, 0.5, -4.5, respectively.



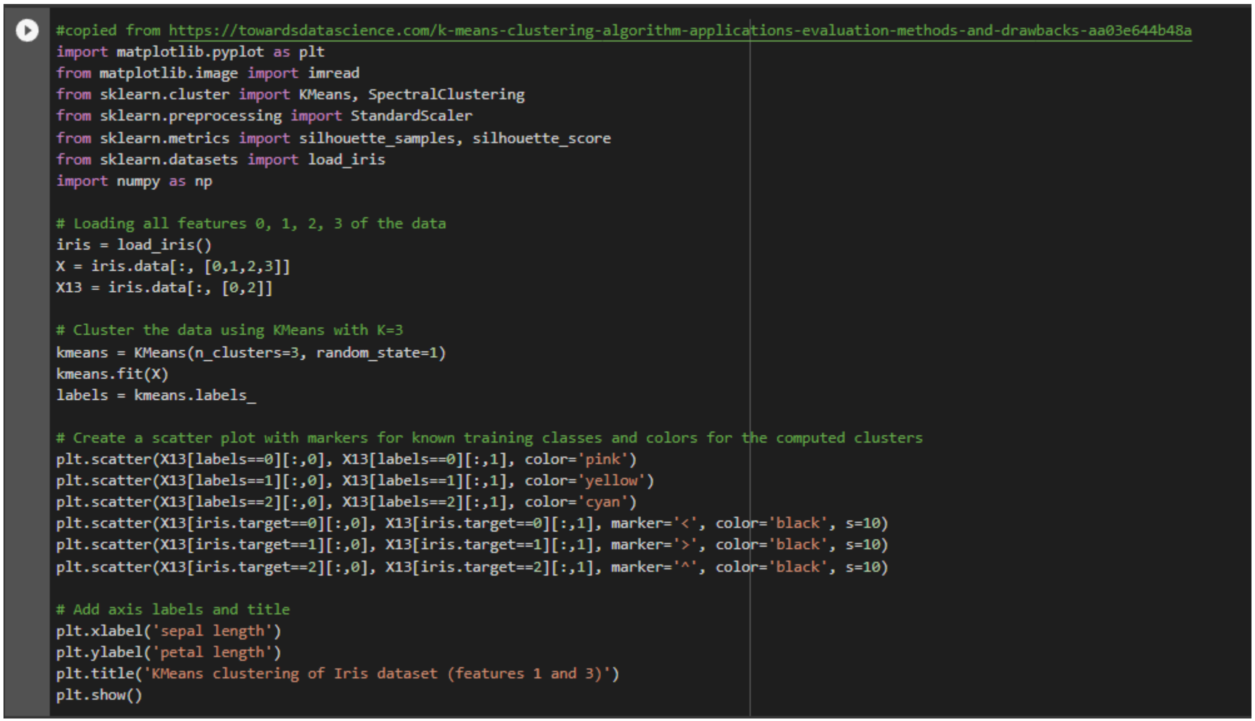
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RANSACRegressor
from sklearn.metrics import mean_squared_error

class PolynomialRegression(object):
#See https://scikit-learn.org/stable/developers/develop.html for Sklearn estimator attributes and methods
#Attributes: degree, coeffs
#Key methods: fit, predict, and score.
    def __init__(self, degree=2):
        print(f"Degree: {degree}")
        self.degree = degree

    def fit(self, X, y):
        self.coeffs = np.polyfit(X.ravel(), y, self.degree)

    def get_params(self, deep=False):
        return {'degree': self.degree}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def predict(self, X):
        poly_eqn = np.poly1d(self.coeffs)
        y_hat = poly_eqn(X.ravel())
        return y_hat

    def score(self, X, y):
        return mean_squared_error(y, self.predict(X))


# Define the dataset
x = np.array([-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([132, 100, 72, 48, 80, 14, 3, -3, -5, -2, -20, 17, 33, 53, 78, 107, 140])

X = x.reshape(-1, 1)

ransac = RANSACRegressor(
    base_estimator=PolynomialRegression(degree=2),
    residual_threshold= 10,
    random_state=0,
    min_samples=6
)
```

```python
ransac.fit(X, y)

# Print the coefficients of the fitted quadratic function
print("Coefficients of the fitted quadratic function using RANSAC:")
print(ransac.estimator_.coeffs)
```

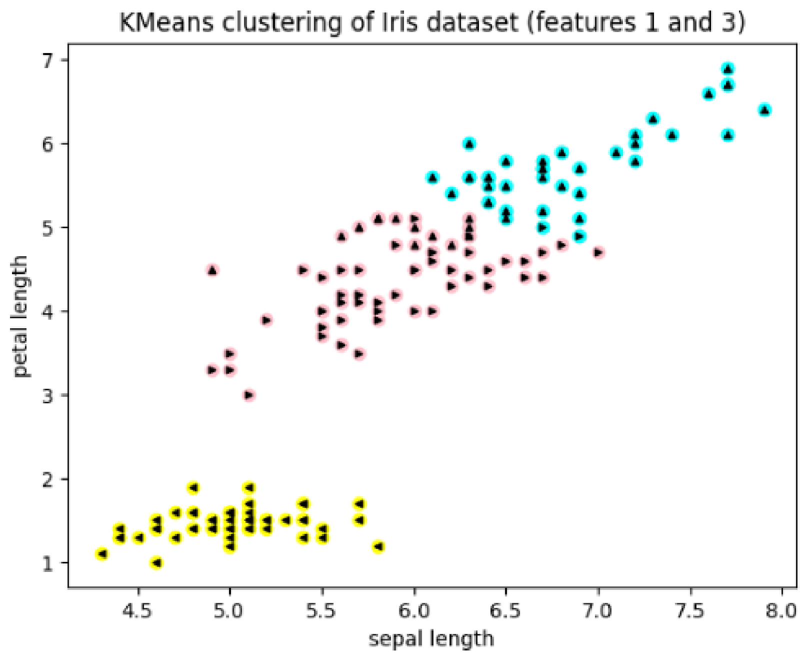
```
Degree: 2
Degree: 2
Coefficients of the fitted quadratic function using RANSAC:
[ 2.20168149  0.50465825 -4.55078223]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ransac.py:343: FutureWarning: `base_esti
  warnings.warn(
```

2. Use K Means clustering on the IRIS dataset.

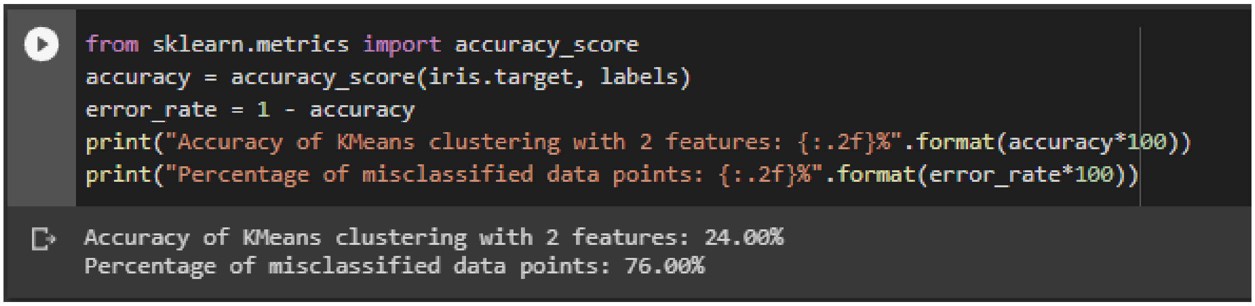
   2.1 10 points. 0.5 hour. Using K = 3, cluster the entire dataset into 3 labels using only features 1 & 3; namely, sepal length and petal length (Note: the example in class used all 4 features for clustering). Show a scatter plot based on these 2 features using known training 3 classes using markers "<" for class 1 (Setosa), ">" for class 2 (Versicolor), and "^" for class 3 (Virginica) while also using colors based on the 3 computed clusters using colors of “pink” for cluster 1, “yellow” for cluster 2, and “cyan” for cluster 3.



```
#copied from https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
import matplotlib.pyplot as plt
from matplotlib.image import imread
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.datasets import load_iris
import numpy as np

# Loading all features 0, 1, 2, 3 of the data
iris = load_iris()
X = iris.data[:, [0,1,2,3]]
X13 = iris.data[:, [0,2]]

# Cluster the data using KMeans with K=3
kmeans = KMeans(n_clusters=3, random_state=1)
kmeans.fit(X)
labels = kmeans.labels_

# Create a scatter plot with markers for known training classes and colors for the computed clusters
plt.scatter(X13[labels==0][:,0], X13[labels==0][:,1], color='pink')
plt.scatter(X13[labels==1][:,0], X13[labels==1][:,1], color='yellow')
plt.scatter(X13[labels==2][:,0], X13[labels==2][:,1], color='cyan')
plt.scatter(X13[iris.target==0][:,0], X13[iris.target==0][:,1], marker='<', color='black', s=10)
plt.scatter(X13[iris.target==1][:,0], X13[iris.target==1][:,1], marker='>', color='black', s=10)
plt.scatter(X13[iris.target==2][:,0], X13[iris.target==2][:,1], marker='^', color='black', s=10)

# Add axis labels and title
plt.xlabel('sepal length')
plt.ylabel('petal length')
plt.title('KMeans clustering of Iris dataset (features 1 and 3)')
plt.show()
```



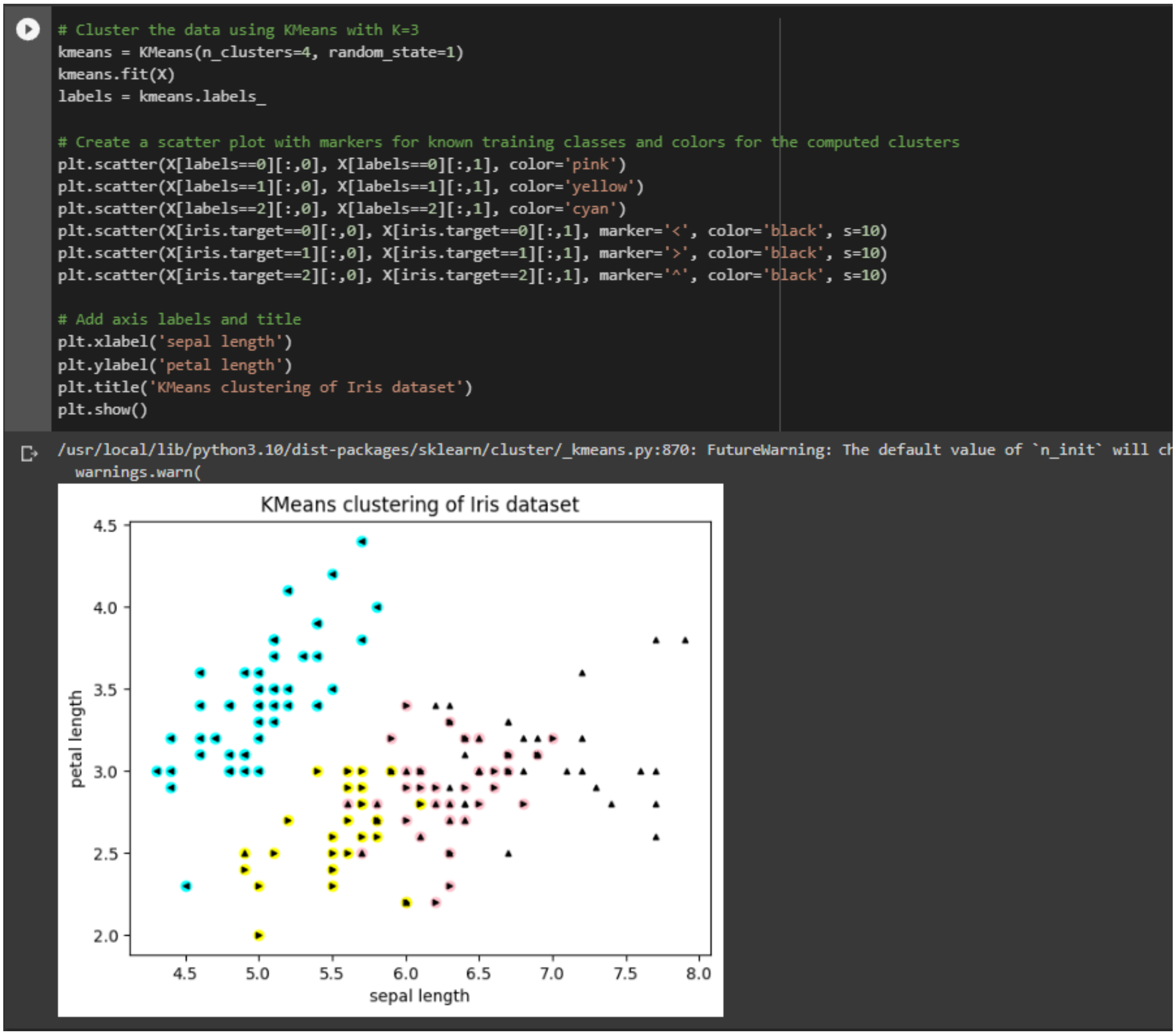
2.2 5 points. Report based on known labels what percent is misclassified when using 2 features.



```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(iris.target, labels)
error_rate = 1 - accuracy
print("Accuracy of KMeans clustering with 2 features: {:.2f}%".format(accuracy*100))
print("Percentage of misclassified data points: {:.2f}%".format(error_rate*100))
```

```
Accuracy of KMeans clustering with 2 features: 24.00%
Percentage of misclassified data points: 76.00%
```

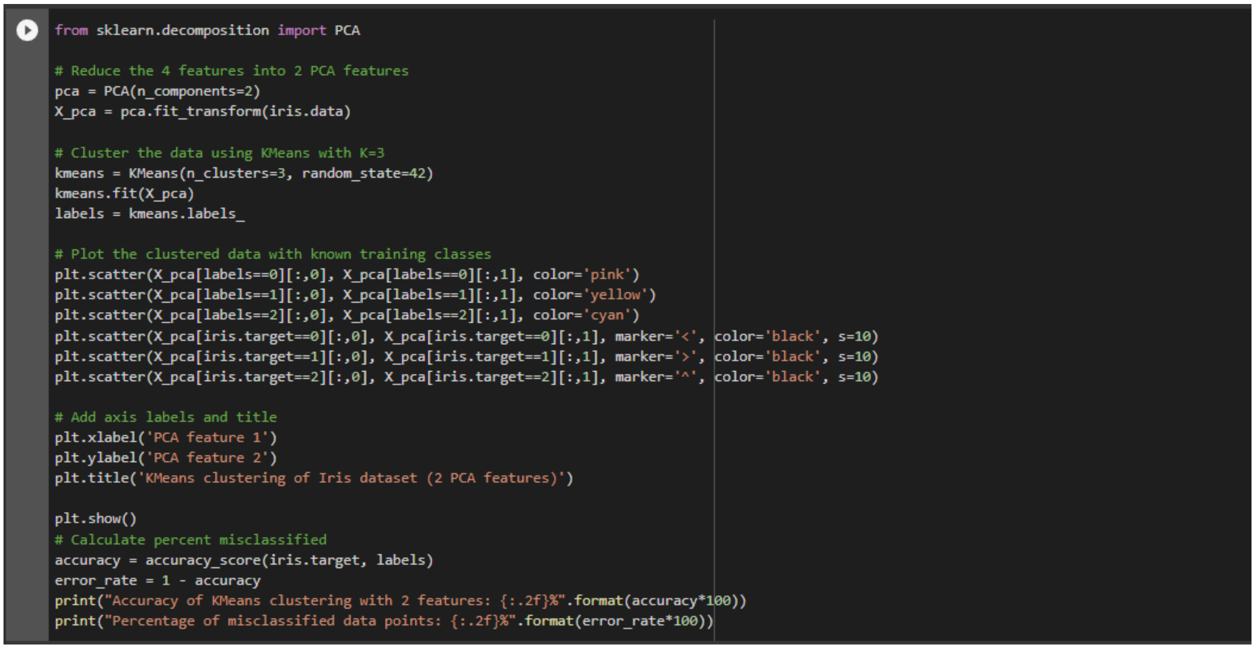
2.3 10 points. 0.5 hour. Plot the result of K Means clustering using all 4 features with K = 4.



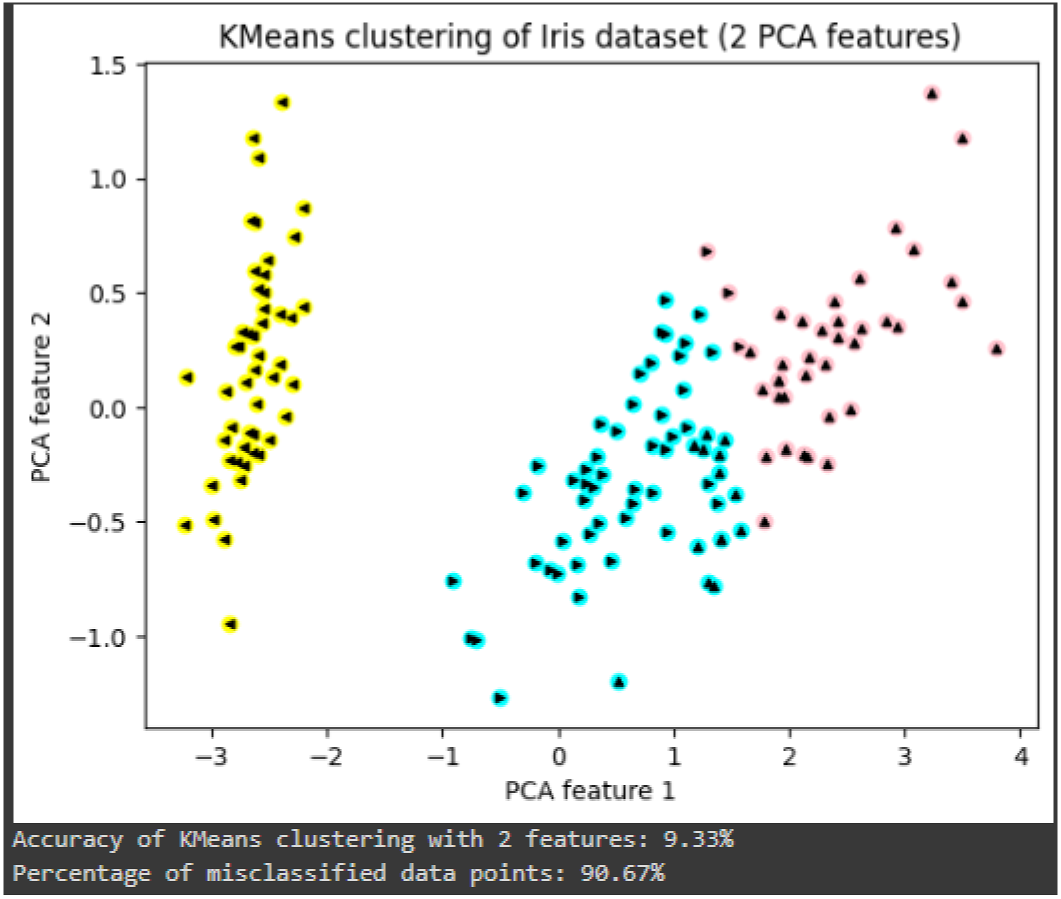
```
# Cluster the data using KMeans with K=3
kmeans = KMeans(n_clusters=4, random_state=1)
kmeans.fit(X)
labels = kmeans.labels_

# Create a scatter plot with markers for known training classes and colors for the computed clusters
plt.scatter(X[labels==0][:,0], X[labels==0][:,1], color='pink')
plt.scatter(X[labels==1][:,0], X[labels==1][:,1], color='yellow')
plt.scatter(X[labels==2][:,0], X[labels==2][:,1], color='cyan')
plt.scatter(X[iris.target==0][:,0], X[iris.target==0][:,1], marker='<', color='black', s=10)
plt.scatter(X[iris.target==1][:,0], X[iris.target==1][:,1], marker='>', color='black', s=10)
plt.scatter(X[iris.target==2][:,0], X[iris.target==2][:,1], marker='^', color='black', s=10)

# Add axis labels and title
plt.xlabel('sepal length')
plt.ylabel('petal length')
plt.title('KMeans clustering of Iris dataset')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
```

2.4 15 points. 1 hour. Reduce the 4 features (sepal length, sepal width, petal length, petal width) into 2 PCA features (an example is also provided in class). Use K = 3 to cluster the entire dataset using these 2 PCA features. Show a scatter plot like in problem 2.1 along with percent misclassified as in problem 2.2.

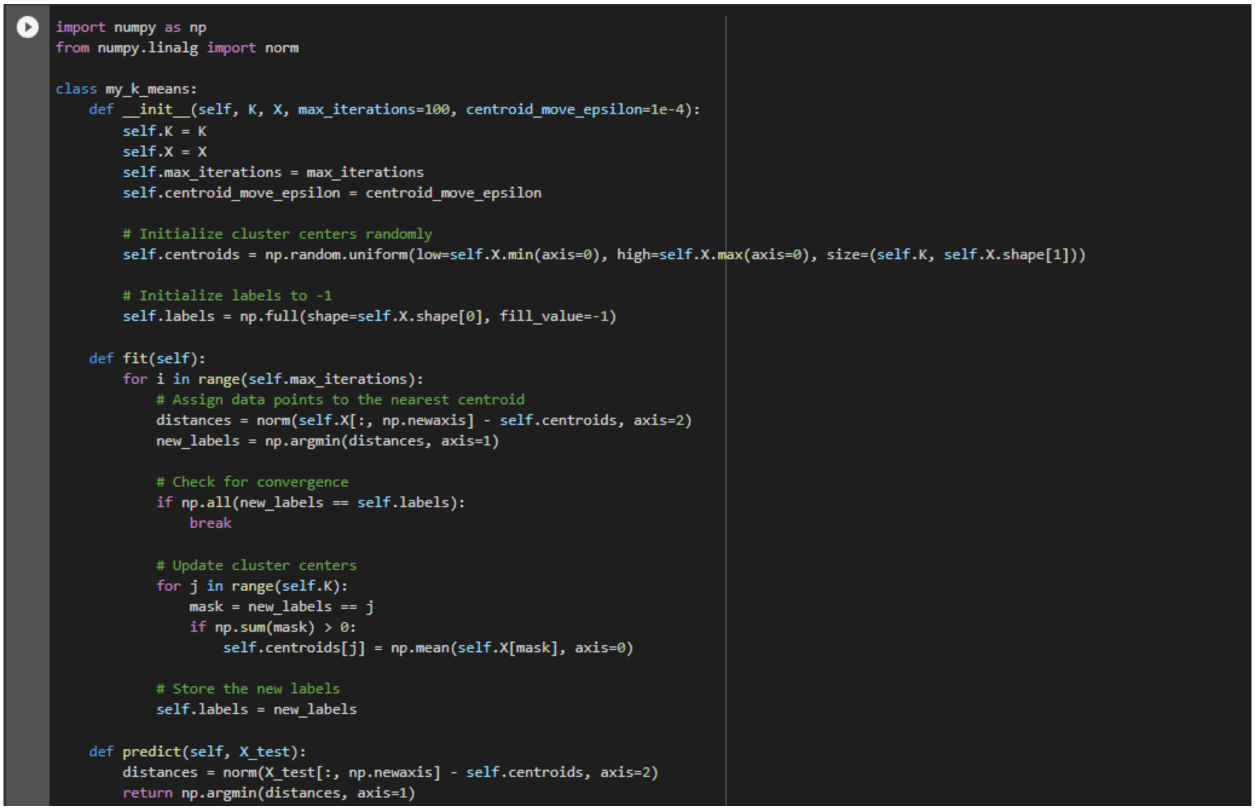


```
from sklearn.decomposition import PCA

# Reduce the 4 features into 2 PCA features
pca = PCA(n_components=2)
X_pca = pca.fit_transform(iris.data)

# Cluster the data using KMeans with K=3
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_pca)
labels = kmeans.labels_

# Plot the clustered data with known training classes
plt.scatter(X_pca[labels==0][:,0], X_pca[labels==0][:,1], color='pink')
plt.scatter(X_pca[labels==1][:,0], X_pca[labels==1][:,1], color='yellow')
plt.scatter(X_pca[labels==2][:,0], X_pca[labels==2][:,1], color='cyan')
plt.scatter(X_pca[iris.target==0][:,0], X_pca[iris.target==0][:,1], marker='<', color='black', s=10)
plt.scatter(X_pca[iris.target==1][:,0], X_pca[iris.target==1][:,1], marker='>', color='black', s=10)
plt.scatter(X_pca[iris.target==2][:,0], X_pca[iris.target==2][:,1], marker='^', color='black', s=10)

# Add axis labels and title
plt.xlabel('PCA feature 1')
plt.ylabel('PCA feature 2')
plt.title('KMeans clustering of Iris dataset (2 PCA features)')

plt.show()
# Calculate percent misclassified
accuracy = accuracy_score(iris.target, labels)
error_rate = 1 - accuracy
print("Accuracy of KMeans clustering with 2 features: {:.2f}%".format(accuracy*100))
print("Percentage of misclassified data points: {:.2f}%".format(error_rate*100))
```

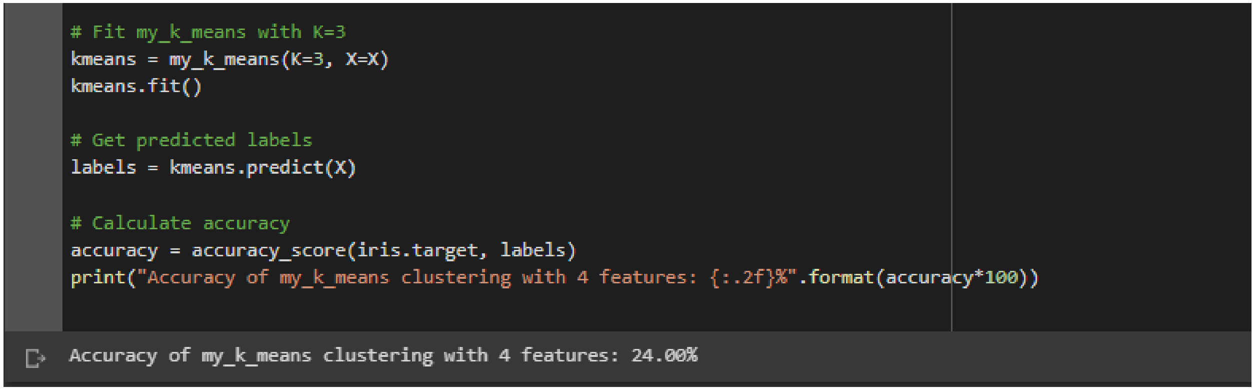


```
Accuracy of KMeans clustering with 2 features: 9.33%
Percentage of misclassified data points: 90.67%
```

2.5 20 points. Redo the example in class with all 4 features and K = 3, but using your own class or function' my_k_means in Python that has initialization parameters: K, X, max_iterations, centroid_move_epsilon and returns y as a 1-D array of integer labels of 1, 2, ..., K.. Each input N-dimensional data X[i] will have a 1-dimensional output label y[i] for i = 1..M where M is the number of data points. The algorithm should start by assigning K cluster centers based on random values from the (min, max) range of each dimension in the N-dimensional data X. It should stop when all centers have moved by less than the centroid_move_epsilon or when the max_iterations is reached. Make sure your results are similar to the K Means library class.



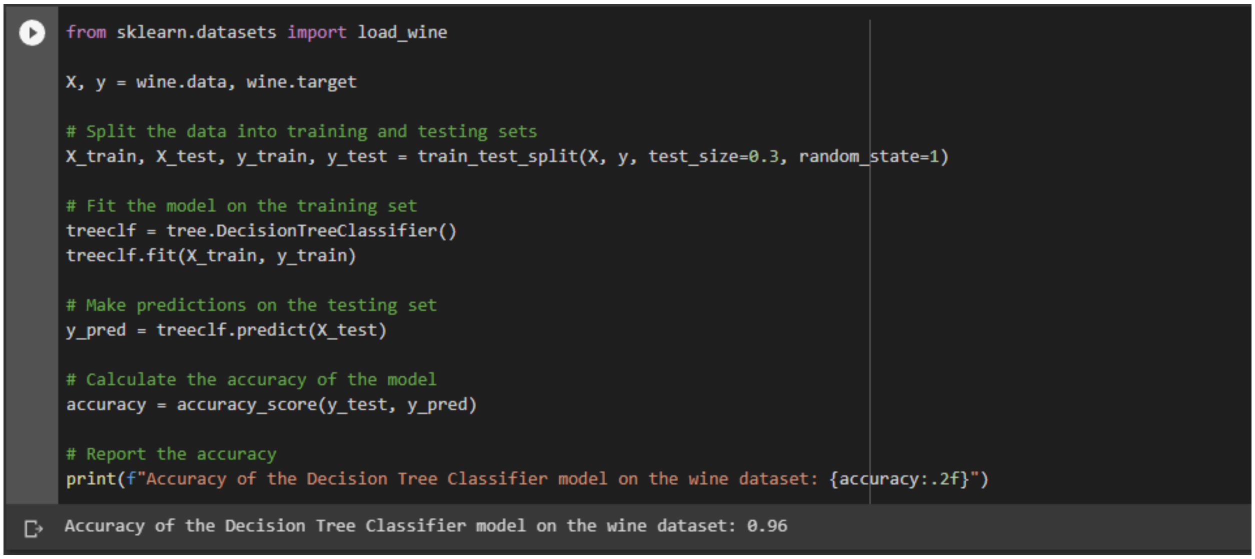
```
import numpy as np
from numpy.linalg import norm

class my_k_means:
    def __init__(self, K, X, max_iterations=100, centroid_move_epsilon=1e-4):
        self.K = K
        self.X = X
        self.max_iterations = max_iterations
        self.centroid_move_epsilon = centroid_move_epsilon

        # Initialize cluster centers randomly
        self.centroids = np.random.uniform(low=self.X.min(axis=0), high=self.X.max(axis=0), size=(self.K, self.X.shape[1]))

        # Initialize labels to -1
        self.labels = np.full(shape=self.X.shape[0], fill_value=-1)

    def fit(self):
        for i in range(self.max_iterations):
            # Assign data points to the nearest centroid
            distances = norm(self.X[:, np.newaxis] - self.centroids, axis=2)
            new_labels = np.argmin(distances, axis=1)

            # Check for convergence
            if np.all(new_labels == self.labels):
                break

            # Update cluster centers
            for j in range(self.K):
                mask = new_labels == j
                if np.sum(mask) > 0:
                    self.centroids[j] = np.mean(self.X[mask], axis=0)

            # Store the new labels
            self.labels = new_labels

    def predict(self, X_test):
        distances = norm(X_test[:, np.newaxis] - self.centroids, axis=2)
        return np.argmin(distances, axis=1)
```



```
# Fit my_k_means with K=3
kmeans = my_k_means(K=3, X=X)
kmeans.fit()

# Get predicted labels
labels = kmeans.predict(X)

# Calculate accuracy
accuracy = accuracy_score(iris.target, labels)
print("Accuracy of my_k_means clustering with 4 features: {:.2f}%".format(accuracy*100))
```

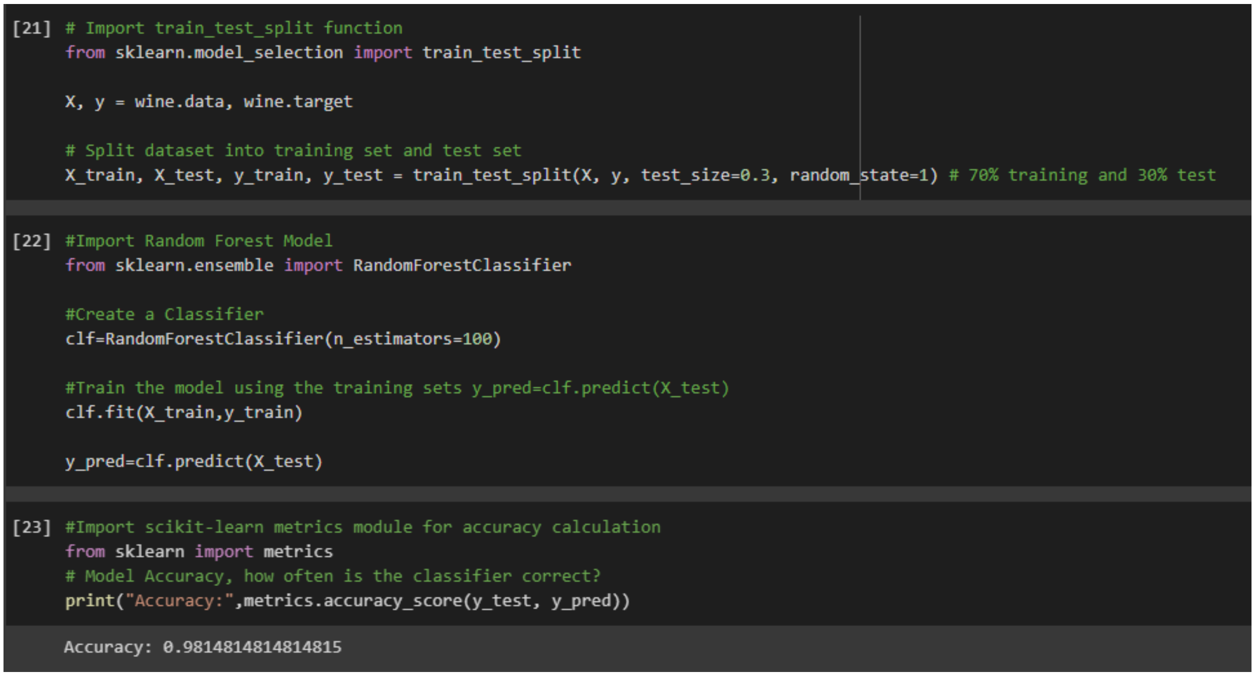
```
Accuracy of my_k_means clustering with 4 features: 24.00%
```

3. 15 points. 1 hour. Decision Trees. Change the “IRIS Decision Tree.ipynb” shown in class, to use SKlearn’s Wine Recognition Dataset instead. Report the classification accuracy % for a single tree using 70%training samples and for a random forest with 100 estimators



```
from sklearn.datasets import load_wine

X, y = wine.data, wine.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Fit the model on the training set
treeclf = tree.DecisionTreeClassifier()
treeclf.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = treeclf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Report the accuracy
print(f"Accuracy of the Decision Tree Classifier model on the wine dataset: {accuracy:.2f}")
```

```
Accuracy of the Decision Tree Classifier model on the wine dataset: 0.96
```



```
[21] # Import train_test_split function
     from sklearn.model_selection import train_test_split

     X, y = wine.data, wine.target

     # Split dataset into training set and test set
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

```
[22] #Import Random Forest Model
     from sklearn.ensemble import RandomForestClassifier

     #Create a Classifier
     clf=RandomForestClassifier(n_estimators=100)

     #Train the model using the training sets y_pred=clf.predict(X_test)
     clf.fit(X_train,y_train)

     y_pred=clf.predict(X_test)
```

```
[23] #Import scikit-learn metrics module for accuracy calculation
     from sklearn import metrics
     # Model Accuracy, how often is the classifier correct?
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
     Accuracy: 0.9814814814814815
```

The Random Forest Classifier model with 100 estimators has an accuracy of 0.9814814814814815, which is approximately 2.7% better than the Decision Tree Classifier model with an accuracy of 0.96 on the wine dataset.