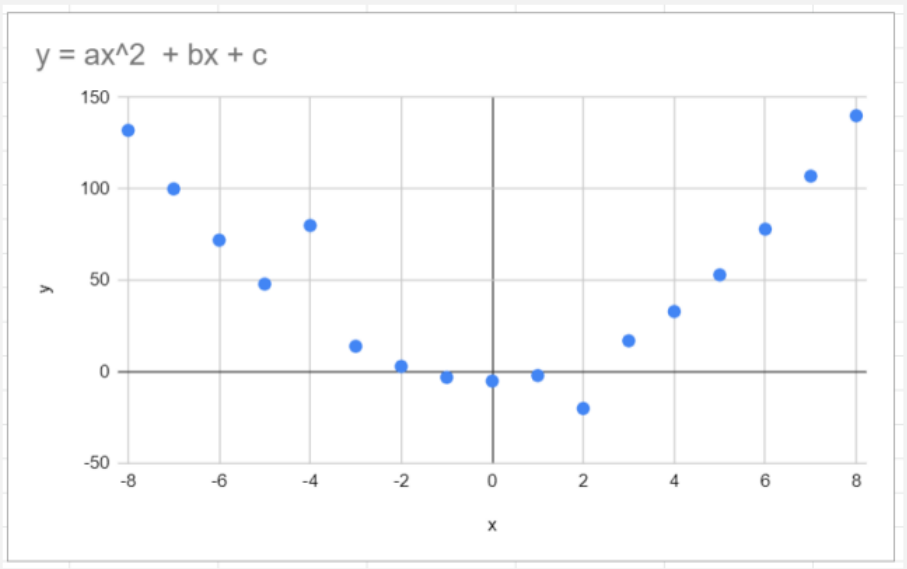
นาย สัณหณัฐ พรมจรรย์ 63070501069

Quiz 9

1. ***20 points.*** **2 hours. RANSAC Regression**. Use RANSAC to find a, b, c for the following dataset where points (x_i, y_i) are discrete samples from a function $f(x) = ax^2 + bx + c$ with 2 outliers. *Hint*: You should get a, b, and c close to 2.2, 0.5, -4.5, respectively.

x_i	y_i
-8	132
-7	100
-6	72
-5	48
-4	80
-3	14
-2	3
-1	-3
0	-5
1	-2
2	-20
3	17
4	33
5	53
6	78
7	107
8	140



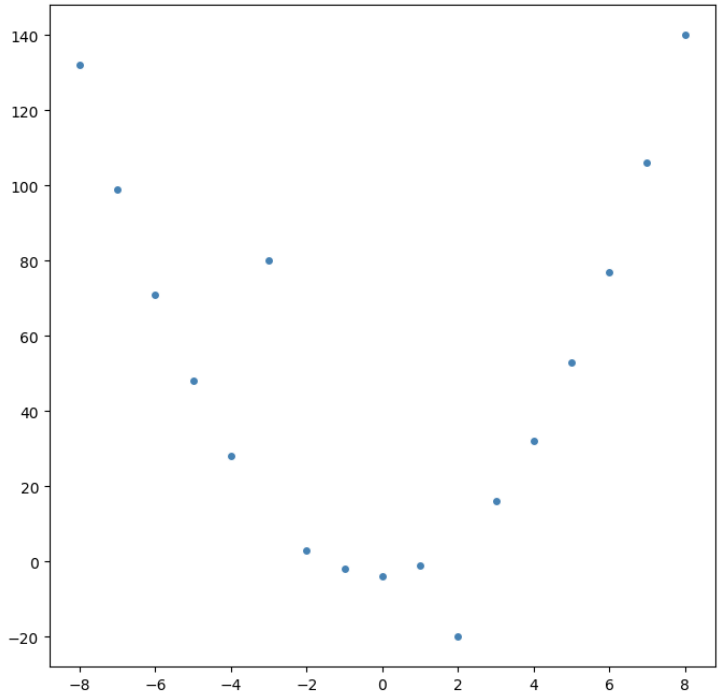
Link: https://colab.research.google.com/drive/19yYWmq_iwGvI_EV9CoikddZyTMfYwbNp?usp=sharing

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, RANSACRegressor
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import random
```

```
# Create dataset with some random perturbation
x_list = []
y_list = []
a=2.2
b=0.5
c=-4.5
for x in range (-8, 9): #x = -8, -7, ..., 7, 8
  x_list.append(x)
  y_hat = int(a*x**2 + b*x + c)
  y_list.append(y_hat)

X = np.array(x_list)
y = np.array(y_list)
```

```
# Add 2 outlier points
y[2-9] = -20
y[-4+9] = 80
```

```
# Create scatter plot for Input Data
plt.figure(figsize=(8, 8))
plt.scatter(X, y,c='steelblue', edgecolor='white',marker='o', label='Input')
plt.show()
```




```
# Select Avg. No of rooms per dwelling as feature
# and fit the model
#
X = X.reshape(-1, 1)
print("X = ", X)
print("y = ", y)
```

```
X =  [[-8]
 [-7]
 [-6]
 [-5]
 [-4]
 [-3]
 [-2]
 [-1]
 [ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]]
y =  [132  99  71  48  28  80   3  -2  -4  -1 -20  16  32  53  77 106 140]
```

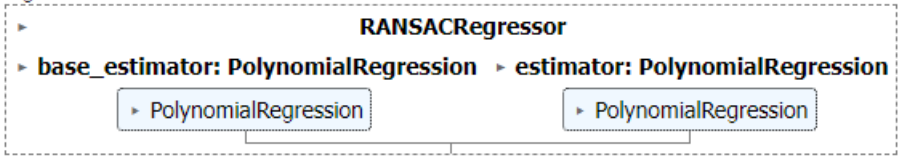
```python
import warnings
import numpy as np
from sklearn.linear_model import RANSACRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_regression
warnings.filterwarnings('ignore')

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))
```

```python
# Also try residual_threshold = 20
ransac = RANSACRegressor(
    base_estimator=PolynomialRegression(degree=2),
    residual_threshold= 20,
    random_state=0,
    min_samples=5
)
```

```
Degree: 2
```

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

```
Degree: 2
```




```python
print("Estimated a b c: ", ransac.estimator_.coeffs)
print("Actual a b c:    ", np.array([a, b, c]))
```

```
Estimated a b c:  [ 2.1870699   0.4958774  -4.42589906]
Actual a b c:     [ 2.2  0.5 -4.5]
```

```
1 X[inlier_mask]
```

```
array([[-8],
       [-7],
       [-6],
       [-5],
       [-4],
       [-2],
       [-1],
       [ 0],
       [ 1],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8]])
```

```
1 y[inlier_mask]
```

```
array([132,  99,  71,  48,  28,   3,  -2,  -4,  -1,  16,  32,  53,  77,
       106, 140])
```

```
1 X[outlier_mask]
```

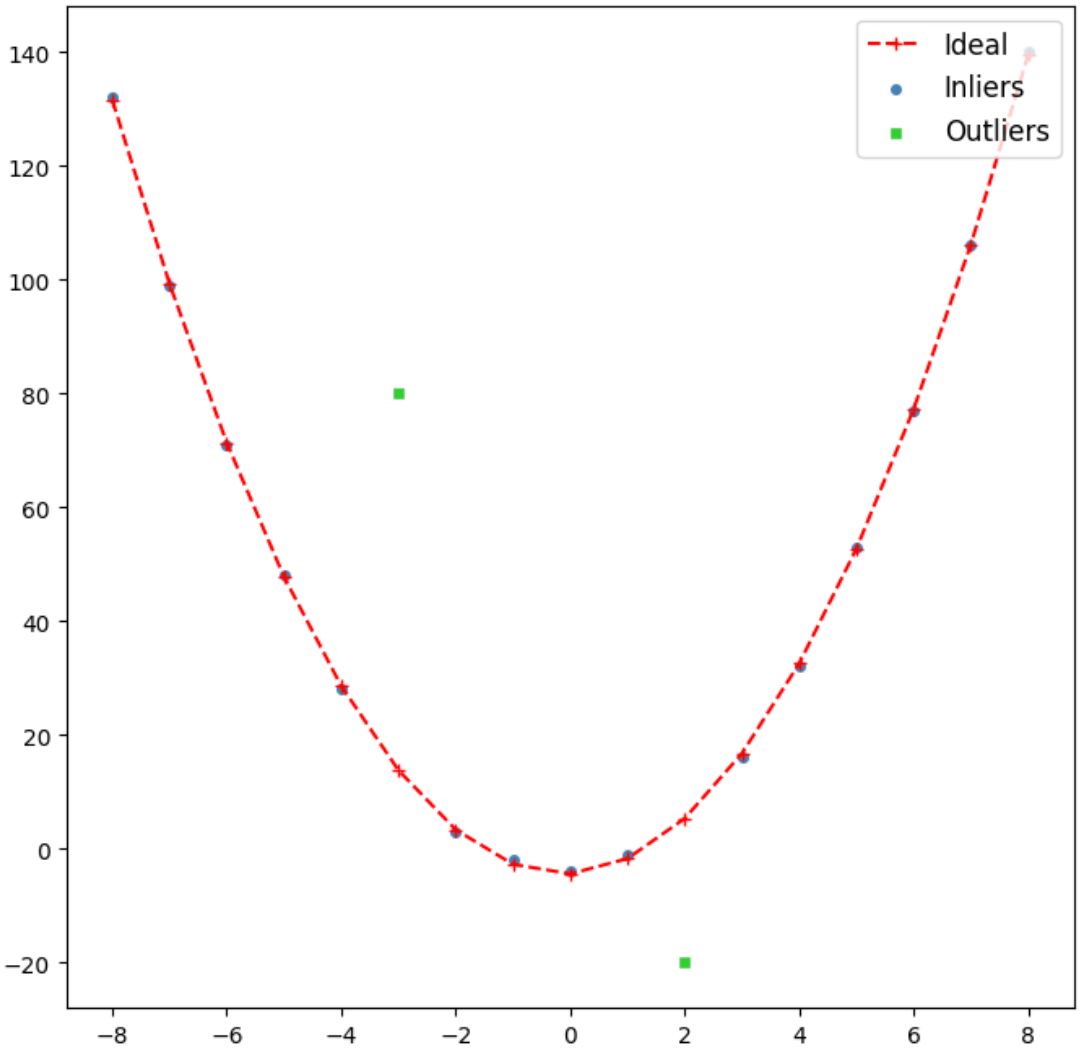
```
array([[-3],
       [ 2]])
```

```
1 y[outlier_mask]
```

```
array([ 80, -20])
```

```
1 x_list = []
2 y_list = []
3 co = ransac.estimator_.coeffs
4 for i in X:
5   y_hat = co[0]*i**2 + co[1]*i + co[2]
6   x_list.append(i)
7   y_list.append(y_hat)
8 X_perfect = np.array(x_list)
9 y_perfect = np.array(y_list)
```

```python
# Draw
#
# Create scatter plot for inlier datset
plt.figure(figsize=(8, 8))

# perfect curve
plt.plot(X_perfect, y_perfect,
         c='red', marker='+', linestyle= 'dashed', label='Ideal')

# Inlier scatter plot
plt.scatter(X[inlier_mask], y[inlier_mask],
            c='steelblue', edgecolor='white',
            marker='o', label='Inliers')

# Create scatter plot for outlier datset
plt.scatter(X[outlier_mask], y[outlier_mask],
            c='limegreen', edgecolor='white',
            marker='s', label='Outliers')
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



2. Use K Means clustering on the IRIS dataset.

Link: https://colab.research.google.com/drive/16psaSjAJXaYoqzPRx2BoujsN7awBB4uh?usp=sharing

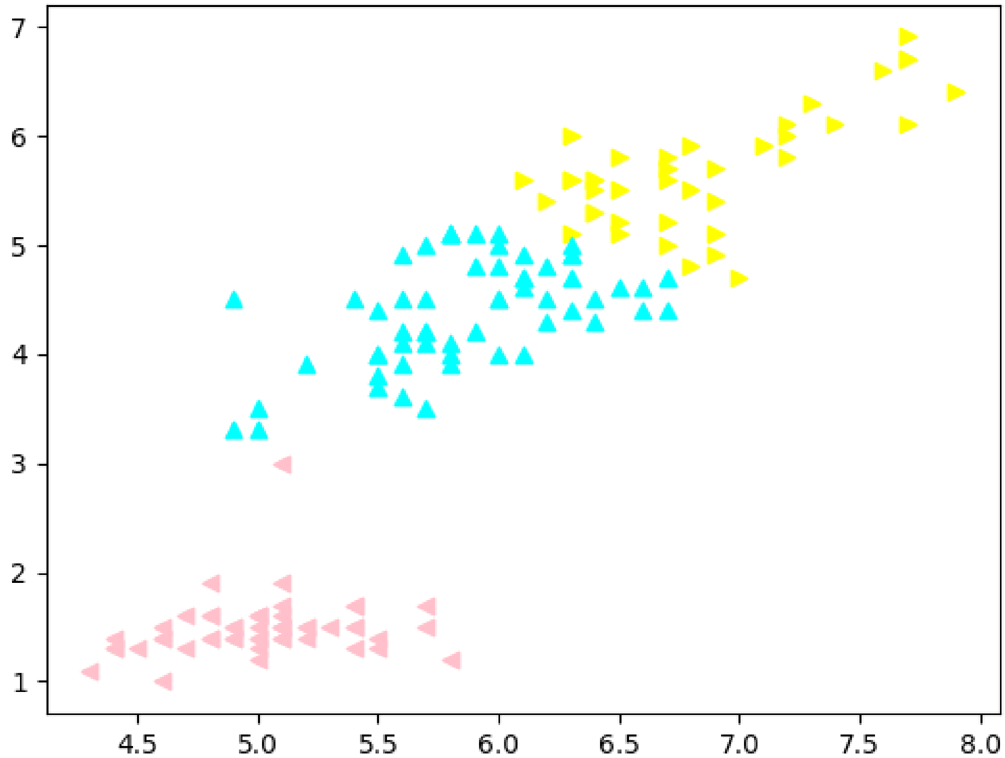
```
#copied from https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
import matplotlib.pyplot as plt
from matplotlib.image import imread
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler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

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.

```
# Load the IRIS dataset
iris = load_iris()
X = iris.data[:, [0, 2]]  # Select features 1 and 3
y = iris.target

# Run the Kmeans algorithm using all 1 3 features
km = KMeans(n_clusters=3, n_init= "auto")
labels = km.fit_predict(X)
print(labels)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
```

```
# Plotting the results of clustering
# Plot of actual class labels using <, >, ^.
plt.scatter(X[labels == 0][:,0] , X[labels == 0][:,1] , marker = "<", color = 'pink')
plt.scatter(X[labels == 1][:,0] , X[labels == 1][:,1], marker = ">" , color = 'yellow')
plt.scatter(X[labels == 2][:,0] , X[labels == 2][:,1] , marker = "^", color = 'cyan')
plt.show()
```



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

```
# Calculate misclassification percentage
misclassified = np.sum(labels != y)
misclassification_percentage = (misclassified / len(y)) * 100

print(f"Misclassification percentage: {misclassification_percentage:.2f}%")
```

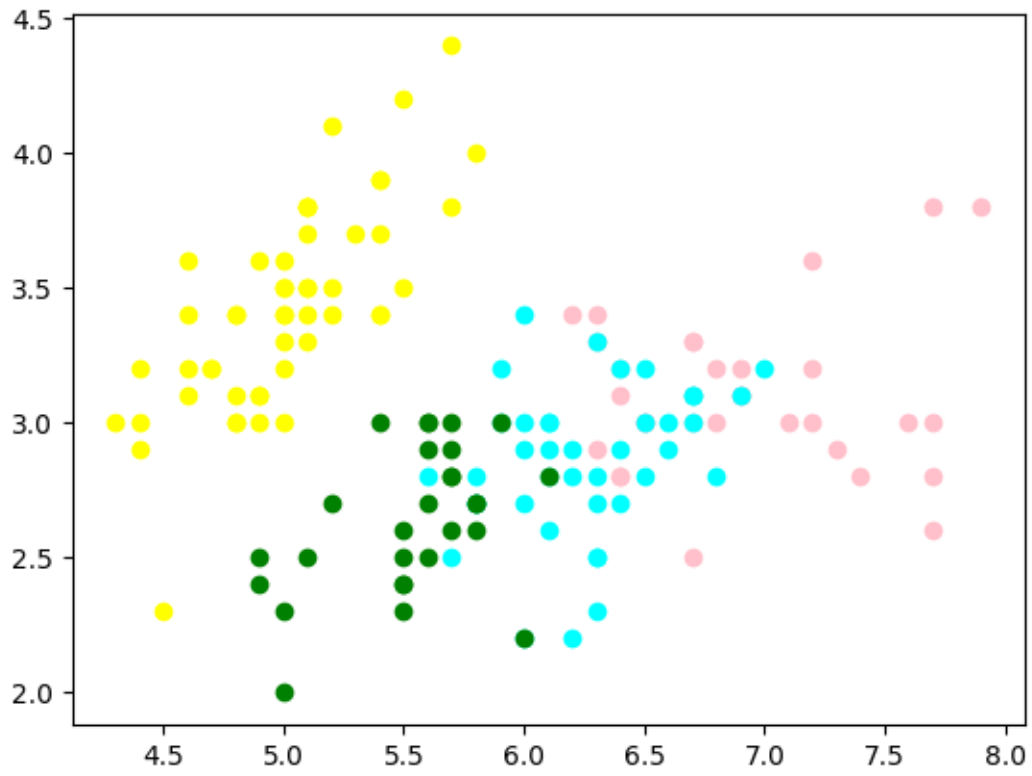
```
Misclassification percentage: 55.33%
```

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

```
# Run the Kmeans algorithm using all 4 features
X = iris.data[:,:]
km = KMeans(n_clusters=4, n_init= "auto")
label = km.fit_predict(X)
print(label)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 2 3 2 2
 2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 3 2 3 3 0 2 0 0 0 0 3 0 0 0 2
 2 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 2 0 0 0 2 0 0 0 2 0 0 0 2 2
 0 2]
```

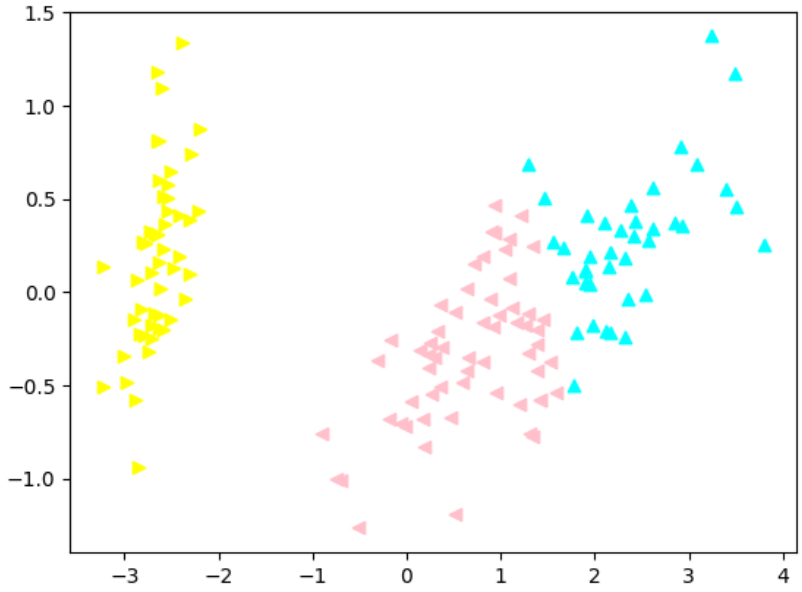
```
#Plotting the results of clustering
plt.scatter(X[label == 0][:,0] , X[label == 0][:,1] , color = 'pink')
plt.scatter(X[label == 1][:,0] , X[label == 1][:,1] , color = 'yellow')
plt.scatter(X[label == 2][:,0] , X[label == 2][:,1] , color = 'cyan')
plt.scatter(X[label == 3][:,0] , X[label == 3][:,1] , color = 'green')
plt.show()
```



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.

```
# Apply PCA for dimensionality reduction
X = iris.data[:, :]
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

k = 3

# Perform K-means clustering on PCA-transformed data
km = KMeans(n_clusters=k, n_init= "auto")
labels = km.fit_predict(X_pca)
```

```
# Plotting the results of clustering
# Plot of actual class labels using <, >, ^.
plt.scatter(X_pca[labels == 0][:,0] , X_pca[labels == 0][:,1] , marker = "<", color = 'pink')
plt.scatter(X_pca[labels == 1][:,0] , X_pca[labels == 1][:,1], marker = ">" , color = 'yellow')
plt.scatter(X_pca[labels == 2][:,0] , X_pca[labels == 2][:,1] , marker = "^", color = 'cyan')
plt.show()
```




```
# Calculate misclassification percentage
misclassified = np.sum(labels != y)
misclassification_percentage = (misclassified / len(y)) * 100

print(f"Misclassification percentage: {misclassification_percentage:.2f}%")
```

Misclassification percentage: 76.00%

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.

```
def my_k_means(K, X, max_iterations, centroid_move_epsilon):
    # Random initialization of cluster centers within the range of each dimension
    min_vals = np.min(X, axis=0)
    max_vals = np.max(X, axis=0)
    centroids = np.random.uniform(min_vals, max_vals, size=(K, X.shape[1]))

    # Iterative K-means algorithm
    for _ in range(max_iterations):
        # Assign each data point to the nearest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=-1)
        labels = np.argmin(distances, axis=-1)

        # Move centroids to the mean of their assigned data points
        new_centroids = np.array([np.mean(X[labels == k], axis=0) for k in range(K)])

        # Check if centroids have moved by less than the threshold
        centroid_moves = np.linalg.norm(new_centroids - centroids, axis=-1)
        if np.all(centroid_moves < centroid_move_epsilon):
            break

        centroids = new_centroids

    return labels
```

```
# Run my_k_means
K = 3
max_iterations = 100
centroid_move_epsilon = 1e-4
labels = my_k_means(K, X, max_iterations, centroid_move_epsilon)

print(labels)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
```

```
km = KMeans(n_clusters=3, n_init="auto")
labels = km.fit_predict(X)
print(labels)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```

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.

Link: https://colab.research.google.com/drive/1bpDNMyA3W_AyvNaVzaQLeBz8USJzR33N?usp=sharing

```python
# Import scikit-learn dataset library
from sklearn import datasets

# Load the Wine dataset
wine = datasets.load_wine()
```

```python
# Print the label species
print(wine.target_names)

# Print the names of the features
print(wine.feature_names)
```

```
['class_0' 'class_1' 'class_2']
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

```python
# Print the wine data (top 5 records)
print(wine.data[0:5])

# Print the wine labels
print(wine.target)
```

```
[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
  3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
  2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
 [1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
  3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```python
# Creating a DataFrame of the wine dataset
import pandas as pd
data = pd.DataFrame({
    'alcohol': wine.data[:, 0],
    'malic_acid': wine.data[:, 1],
    'ash': wine.data[:, 2],
    'alcalinity_of_ash': wine.data[:, 3],
    'magnesium': wine.data[:, 4],
    'total_phenols': wine.data[:, 5],
    'flavanoids': wine.data[:, 6],
    'nonflavanoid_phenols': wine.data[:, 7],
    'proanthocyanins': wine.data[:, 8],
    'color_intensity': wine.data[:, 9],
    'hue': wine.data[:, 10],
    'od280/od315_of_diluted_wines': wine.data[:, 11],
    'proline': wine.data[:, 12],
    'species': wine.target
})
data.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	species
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

```python
# Import train_test_split function
from sklearn.model_selection import train_test_split

X=data[['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
        'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
        'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']]  # Features
y=data['species']  # Labels

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

```python
# Import the DecisionTreeClassifier and RandomForestClassifier Model
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Create a Decision Tree Classifier
clf_tree = DecisionTreeClassifier()

# Train the Decision Tree Classifier using the training sets
clf_tree.fit(X_train, y_train)

# Make predictions on the test set
y_pred_tree = clf_tree.predict(X_test)
```

```python
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Calculate the accuracy of the Decision Tree Classifier
accuracy_tree = metrics.accuracy_score(y_test, y_pred_tree)
print("Decision Tree Accuracy:", accuracy_tree)

# Create a Random Forest Classifier with 100 estimators
clf_forest = RandomForestClassifier(n_estimators=100)

# Train the Random Forest Classifier using the training sets
clf_forest.fit(X_train, y_train)

# Make predictions on the test set
y_pred_forest = clf_forest.predict(X_test)

# Calculate the accuracy of the Random Forest Classifier
accuracy_forest = metrics.accuracy_score(y_test, y_pred_forest)
print("Random Forest Accuracy:", accuracy_forest)
```

```
Decision Tree Accuracy: 0.8888888888888888
Random Forest Accuracy: 0.9629629629629629
```