Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "VBM"
COLLABORATORS = ""
```

Build a Custom KNN Classifier on a Dataset

In this exercise you will build a custom KNN classifier on a provided dataset. Along the way, you will explore different aspects of your data as a way to understand its characteristics. Let's get started!

In [2]:
```
from sklearn import neighbors, datasets
from sklearn.model_selection import train_test_split
# Define your data splits
iris = datasets.load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris["data"], iris["target"]
                                                    test_size=0.2, random_state=
```

Exploratory Data Analysis

Exploratory data analysis (EDA) is a preparation step in modelling that is designed to help you understand characteristics of the data. While `exploratory` is in the phrase, the goal of EDA is to answer the question of whether the data you have is sufficient for a predictive task. In other words, are the features you can extract from the data "good enough" to actually learn a model of your target variable.

For our dataset, let us first begin by understanding the shape of our data. Write code to answer the following questions:

1. How many datapoints do we have in our train and test splits?
2. How many features do we have?
3. How many distinct labels do we have?

In [3]:
```
# Perform data shape analysis
# YOUR CODE HERE
print(X_train.shape)
print(X_test.shape)
# We also have 3 possible output labels
print(set(y_train) | set(y_test))
```

```
(120, 4)
(30, 4)
{0, 1, 2}
```

Understanding Feature Relationships

In our provided dataset, the four features represent *sepal length*, *sepal width*, *petal length*, and *petal width* of a collection of flowers. Sometimes features in a dataset may not be completely independent, and may change in predictable ways.

The **Pearson Correlation** is one way of quantifying the extent of a linear relationship between a pair of random variables. Given two random variables `X` and `Y`, the Pearson correlation is defined as:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

where

$$cov(X,Y)$$

represents the covariance between the variables and

$$\sigma$$

represents a variable's variance.

Compute the Pearson Correlation coefficients of the features in our dataset.

Tip: a heatmap is a useful way to inspect correlation values.

In [4]:

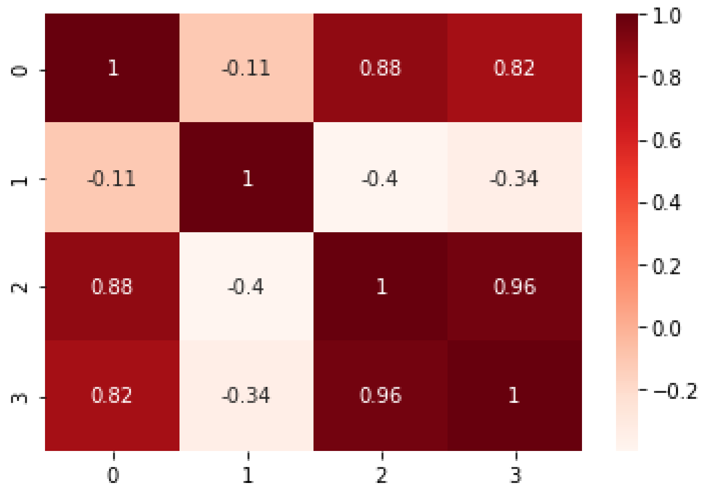
```python
# Compute correlation coefficients
# YOUR CODE HERE

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
corr = np.corrcoef(X_train, rowvar=False)

# Note how sepal length is highly correlated with petal length and petal width.
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
```

Out[4]:

```
<AxesSubplot:>
```



Dimensionality Reduction

Given our previous correlation analysis, it seems that some of our features are redundant. In other words, it seems that we could build a machine learning model with fewer features rather than the 4 provided in the original dataset without losing too much predictive performance.

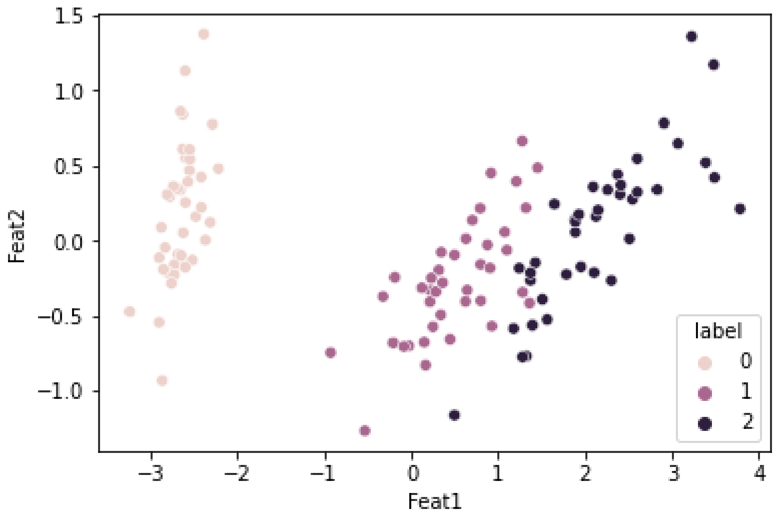
Let's put this to the test by performing what is called **dimensionality reduction**. Here we will take our original feature space of 4 features and see if we can extract a lower-dimensional feature space that still captures the bulk of information in our dataset.

We will do this by performing **principal components analysis** or **PCA**. Check this link for a refresher on PCA.

Run PCA on the train portion of our dataset to extract a 2-dimensional feature subspace. After you have reduced the feature space, plot the projections of these features onto the principal components. Additionally color the points by their labels.

In [5]:
```
# Run PCA
# YOUR CODE HERE
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X_train)
df = pd.DataFrame(X_reduced,columns=["Feat1", "Feat2"])
df["label"] = y_train
# Note here how even after we have reduced the features into a lower-dimensional
# by their labels, indicating that we can still capture much of the predictive p
sns.scatterplot(data=df, x="Feat1", y="Feat2", hue="label")
```

Out[5]:
```
<AxesSubplot:xlabel='Feat1', ylabel='Feat2'>
```



In [6]:
```
class MyKNN(object):
    def __init__(self):
        self.model = neighbors.KNeighborsClassifier(2)

    def train(self, X, y):
        self.model.fit(X, y)

    def predict(self, X):
        return self.model.predict(X)
```

In [7]:
```
from sklearn.metrics import accuracy_score
def compute_accuracy(model, test_x, test_y):
    predicted = model.predict(test_x)
    return accuracy_score(test_y, predicted)
```

In [8]:
```
model = MyKNN()
model.train(X_train, y_train)
assert compute_accuracy(model, X_test, y_test) > 0.9
```

Error Analysis

Now that we have built a custom KNN classifier for our dataset, it is standard to perform *error analysis* to understand what mistakes our model is making and how we can further improve our model.

Using your trained model, find the points that the model incorrectly predicts. Do you see any trends in the mistakes the model makes?

In [9]:
```
# Perform error analysis
# YOUR CODE HERE
predictions = model.predict(X_test)
# Which labels we get wrong
print(y_test[predictions != y_test])
```

```
[2 2]
```

```
In [ ]:
```