COMP60711 Lab Introduction

The first week's lab will **not be assessed**, as the goal is for you to get set up and get used to the tools you'll need for the coursework.

If you have this file open, it's assumed you have gone through the `comp60711_part1_guide.html` file to get set up.

Part 1: Python

Python is the main programming language used in data science/machine learning, due to the vast array of popular and useful libraries. One such library is `scikit-learn`, which we will be using in the coursework.

Q1.1

For this question, we will be using `sklearn.datasets.load_diabetes` to load a dataset to experiment with.

Below you will find short question/instructions and code comments to guide your answer.

In [1]:
```
# These are the imports that you will need
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load the data and labels
data, labels = load_diabetes(return_X_y=True)
```

Split this data (and labels) into a train and test set, using a 75:25 split (75% training).

In [2]:
```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.25, random_state=42
)
```

Create and fit the model to the training data.

In [3]:
```
# Create the model
model = LinearRegression()
# Fit the model
model.fit(x_train, y_train)
```

Out[3]:
```
LinearRegression()
```

What is the `mean_squared_error` of the prediction on the test set?

In [4]:
```
# Get the predictions
y_pred = model.predict(x_test)
# Calculate the mean squared error
mean_squared_error(y_test, y_pred)
```

```
Out[4]: 2848.2953079329427
```

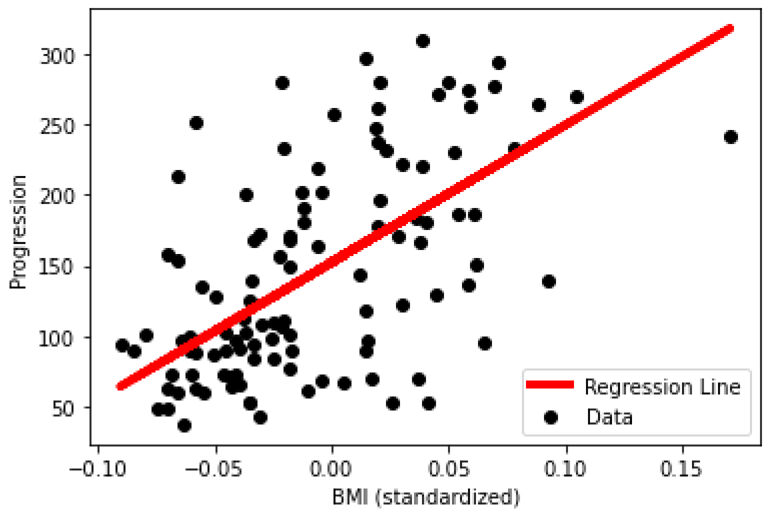
Now, wrap the above steps into a single for loop, where we take a different split of the data. Record the mean squared error across 10 runs, and then report the mean and standard deviation of these errors.

```
In [5]: # Create a list to hold results
mses = []
# Loop over 10 runs
for i in range(10):
    # Split using a different seed each time
    x_train, x_test, y_train, y_test = train_test_split(
        data, labels, test_size=0.25, random_state=i
    )
    # Create the model
    model = LinearRegression()
    # Fit the model
    model.fit(x_train, y_train)
    # Get the predictions
    y_pred = model.predict(x_test)
    # Calculate the mean squared error
    mses.append(mean_squared_error(y_test, y_pred))
# Print the mean and standard deviation
print(f"Mean MSE:    {np.mean(mses)}")
print(f"Std dev MSE: {np.std(mses)}")
```

```
Mean MSE:    2967.1647857368616
Std dev MSE: 217.34186770438617
```

Now we will fit the model using only a single feature, so that we can plotting the resulting line. The code below selects the relevant feature for you. Partial code for some steps is given, you will need to add the relevant variables, and fill in the missing lines (as specified by the comments).

```
In [6]: # Re-load and select a single feature
data, labels = load_diabetes(return_X_y=True)
data = data[:, np.newaxis, 2]
# Split the data and labels again into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(
        data, labels, test_size=0.25, random_state=42
    )
# Create the model
model = LinearRegression()
# Fit the model
model.fit(x_train, y_train)
# Predict on the test set
y_pred = model.predict(x_test)
# Create figure and axis
fig, ax = plt.subplots()
# Add the x and y values for the data below
ax.scatter(x_test, y_test, color="black", label="Data")
# Add the x and y values for the regression line below
ax.plot(x_test, y_pred, color="red", linewidth=4, label="Regression Line")
# Label axes
ax.set_xlabel("BMI (standardized)")
ax.set_ylabel("Progression")
# Create the legend
_ = ax.legend()
```



Q1.2

In this question, we will instead look at clustering, on the famous iris dataset.

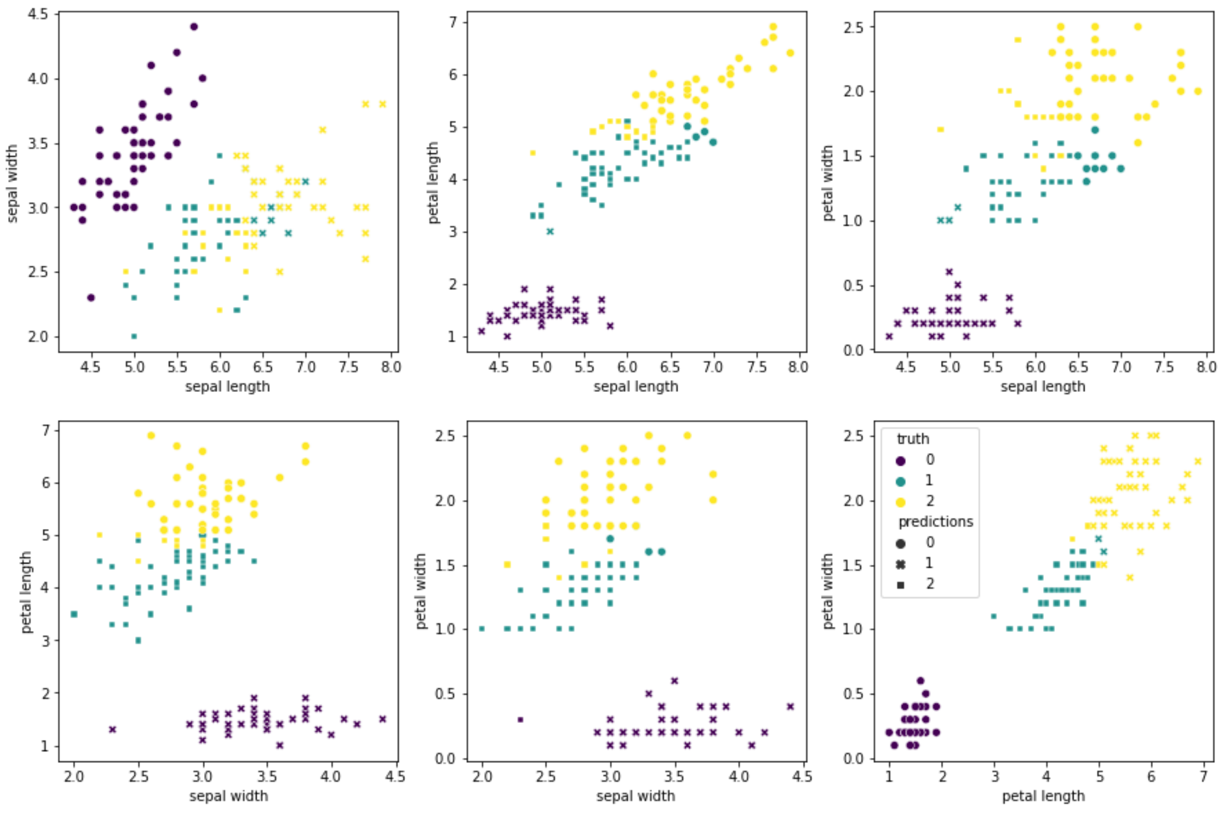
Visualizing the data can help us better understand it. For a dataset with 4 features, one way to do this could be to plot every 2D combination of features.

In [7]:
```python
# Relevant imports
from itertools import combinations
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import seaborn as sns
# To get the combinations of the 4 features
combs = list(combinations(range(4), 2))
```

Now, we need to loop over the data to plot each combination of features. We want to compare both the real classes and the cluster assignments to visualize how they compare. For this, we will use marker styles and marker colour to differentiate between the two. To do that, we will use `seaborn` as it makes this simple. We need to store our results into a `pandas` DataFrame to enable this.

Below, we have provided code to load the data and the start of the for loop is provided. Finish the code using the comments as a guide.

In [8]:
```python
# Import seaborn
import seaborn as sns
# Create a grid of plots
# As `len(combs)=6`, a 2x3 grid makes sense.
fig, axes = plt.subplots(2,3, figsize=(15,10))
# Load the data
data, labels = load_iris(return_X_y=True)
# The names for each feature
feature_names = ["sepal length", "sepal width",
                 "petal length", "petal width"]
# For loop
# This gives us a counter (i), an axes (ax),
# and the indexes of the features (feature_pair)
for i, (ax, feature_pair) in enumerate(zip(axes.flatten(), combs)):
    # Select the relevant columns of the data
    subset = data[:, feature_pair]
    # Create the model (using 3 clusters)
    km = KMeans(n_clusters=3)
    # Fit the model
    km.fit(subset)
    # Get feature names
    x_name, y_name = feature_names[feature_pair[0]], feature_names[feature_pa
    # Create the dataframe of results
    # You will need to fill in the relevant variables
    df = pd.DataFrame({
        "predictions": km.labels_,
        "truth": labels,
        x_name: subset[:, 0],
        y_name: subset[:, 1]
    })
    # Create the scatter plot
    # Only create the legend for the final plot
    if i == len(combs)-1:
        sns.scatterplot(
            x=x_name,
            y=y_name,
            data=df,
            hue="truth",
            style="predictions",
            ax=ax,
            legend="brief",
            palette="viridis"
        )
    else:
        sns.scatterplot(
            x=x_name,
            y=y_name,
            data=df,
            hue="truth",
            style="predictions",
            ax=ax,
            legend=False,
            palette="viridis"
        )
```



Inspect the graphs to see how well the clusters match.

Part 2: Weka

There are many graphical tools that are seeing both increased utility and prevalence, so it is useful to also have some experience with using these. Therefore, in addition to Python we will also be using Weka, which you'll need to install. When you first open Weka, you'll have options such as "Explorer" and "Experimenter", which opens separate interfaces.

Q2.1

Start by opening the "Explorer" part of Weka. Then, open `weather.nominal.arff`. For this dataset, we are deciding whether to play tennis or not based on past weather conditions.

In the "Classify" tab, use the following learning schemes, with the default settings to analyse the weather data (in `weather.arff`): `ZeroR` (majority class), `OneR`, `NaiveBayesSimple`, `J48`. For each, select "Use training set", and then do the same again using the default "Percentage Split" of 66%. Look at the accuracy for each of the models. What is the cause of the difference?

Which classifier are you more likely to trust when determining whether to play?

Q2.1 Answer

Q2.1 Possible Response

The cause of the difference is the amount of data used for training. Performance will of course be better when using all of the data, and worse when trying to generalize from a smaller set to an unseen set of data.

In terms of classifier trustworthiness, it depends on the data and so the reasoning is more important than the answer. Of course, `ZeroR` has no learning involved, and is just hoping that what has previously occurred more often will happen again. The more complex models actually learn from the data, so as long as the humidity etc. are predictive of whether it will rain, then they should perform better. `J48` has the advantage of being easily inspected, to see how a decision was made, which should increase trustworthiness.

Q2.2

For this, we will use the "Experimenter" tab to show how Weka can set up larger, more complex experiments, while collecting performance statistics and testing the significance of the results.

First, click "New" to start a new experiment. By default, 10-fold cross-validation should be selected, and the experiment should be repeated 10 times. Add the "breast-cancer" and "segment-challenge" datasets, and the J48 and ZeroR algorithms. Run the experiment (in the "Run" tab).

Then, in the "Analyse" tabs, click "Experiment" and then "Perform test" to load the results from the experiment, and then perform testing on the results.

J48 should achieve an accuracy of 95.71% on the "segment-challenge" datasets. Also, the asterisk ("*") shows that ZeroR performed significantly worse on the "segment-challenge" dataset. Add a screenshot of this table below, making sure it's clearly legible!

Then, analyse the results again but this time showing the standard deviations, so that we can see how robust the reported accuracies are. Does this change your interpretation of the results?

Q2.2 Answer

Q2.2 Possible Response

This is mostly about them getting to grips with the experimenter part of Weka, so encourage them to play around with different algorithms and different datasets (as this should not take long).

Narrower standard deviations indicate more robustness of the algorithm, i.e. less sensitivity to the particular split. This may indicate that either the model is not overfitting, or that it is not learning at all! For the latter, very simple models such as `ZeroR` can give us a worst-case baseline, which can help identify underfitting.

If there's time

If you have extra time, we strongly encourage you to play around with both the Python code in Q1 and Weka, as familiarity with these (particularly Python) will be helpful for the coursework. Some example questions are shown below, but what is most important is gaining an understanding of the topics on this course, and the tools available to you for using and analysing these models.

For example, how else could you visualize the different features of the iris dataset? A pairwise scatter plot? 3D plots?

Do you get a lower mean squared error when using a polynomial regression? At what order does the model overfit?