Practical 1

Jumping Rivers

We will build a linear regression model for predicting the fuel economy of a vehicle given some other attributes on that vehicle. The data can be access from the **jrpyml** package

```
import jrpyml

cars = jrpyml.datasets.cars.load_data()
```

- Begin by creating a scatter plot of fuel economy, (the 'FE' variable) against engine displacement ('EngDispl')
- What would we expect a model between these two variables to tell us?
- Fit a simple linear regression model with fuel economy as the response variable and engine displacement as the input. **sklearn** expects separate array objects for the predictors and the response of a model. The following code should get you started with shaping the inputs and outputs as necessary

```
X, y = cars.drop('FE', axis=1), cars['FE']

# remember to reshape input to a 2d array
x_train = X['EngDispl'].values.reshape(-1, 1)
y_train = y
```

- What is the average decrease in fuel economy for each 1 litre increase in displacement according to this model?
- Draw a scatter plot with the fitted model line
- Create a plot of model fitted values against residuals
- What does this plot tell us?
- Plot the fitted values against the true observations
- What does this plot tell us about the predictive performance of the model across the range of the response?
- We will refit the model with a square term for the engine displacement variable. A handy way to go from our original single predictor to one that includes both a linear and a square term is to use `np.hstack()` (horizontal stacking of arrays). Fit the same model with the new input and look at the scatter plot with model line.

```
import numpy as np
x_train = np.hstack([x_train, x_train*x_train])
```

- Now we wish to add the transmission (‘Tranmission’) variable to our model. This variable is categorical so we will require some preprocessing prior to fitting the model. The following will create a column transformer which will standardise the numeric variables and one hot encode the categorical variable

```
x_train = np.hstack([
  X[['EngDispl']],
  X[['EngDispl']]*X[['EngDispl']],
  X[['Transmission']]
])
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

preprocessor = ColumnTransformer([
  ('num', StandardScaler(), [0, 1]),
  ('cat', OneHotEncoder(), [2])
])
```

- Create a pipline that will run the preprocessor and fit a linear regression model
- We can assess which model gave us the smallest overall mean squared error using the `mean_squared_error` function from the **sklearn.metrics** module.

```
from sklearn.metrics import mean_squared_error
```

- Which model gave better performance