Solutions 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')

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure()
sns.scatterplot(x='EngDispl', y='FE', data=cars)
plt.show()
```

- What would we expect a model between these two variables to tell us?

```
## An average descrease in fuel economy for increasing engine size
```

- 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
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train, y_train)
```

- What is the average decrease in fuel economy for each 1 litre increase in displacement according to this model?

```
model.coef_
```

```
## array([-4.52092928])
```

- Draw a scatter plot with the fitted model line

```
plt.figure()
sns.scatterplot(x='EngDispl', y='FE', data=cars)
fitted = model.predict(x_train)
sns.lineplot(x='EngDispl', y=fitted, data=cars)
plt.show()
```

- Create a plot of model fitted values against residuals

```
import numpy as np
resid = y_train - fitted
resid = (resid - np.mean(resid))/np.std(resid, ddof=1)

fig, ax = plt.subplots(1, 1)
ax.scatter(fitted, resid)
ax.set_ylabel('Residuals')
ax.set_xlabel('Fitted Values')
ax.hlines([-2, 0, 2], xmin=np.min(fitted), xmax=np.max(fitted))
plt.show()
```

- What does this plot tell us?

```
##  residuals show us structure that the model has not accommodated.
##  there appears to be some trend here, the curved shape indicates that we
##  potentially require some transformation of variables
##  a squared term might help
##
```

- Plot the fitted values against the true observations

```
fig, ax = plt.subplots(1, 1)
ax.scatter(y_train, fitted)
ax.set_ylabel('Fitted Values')
ax.set_xlabel('Observed Values')
ax.plot([15, 50], [15, 50])
plt.show()
```

- What does this plot tell us about the predictive performance of the model across the range of the response?

```
fig, ax = plt.subplots(1, 1)
ax.scatter(x_train, resid)
ax.set_ylabel('Residuals')
ax.set_xlabel('Engine Displacement')
```

```
ax.hlines([-2, 0, 2], xmin=np.min(x_train), xmax=np.max(x_train))
plt.show()

# We seem to overestimate more often than not in the 25-35 range.
# At the upper end we consistently under estimate the true values
```

- 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])

model.fit(x_train, y_train)

plt.figure()
sns.scatterplot(x='EngDispl', y='FE', data=cars)
fitted = model.predict(x_train)
sns.lineplot(x='EngDispl', y=fitted, data=cars)
plt.show()
```

- 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

```
from sklearn.pipeline import Pipeline
```

```
model2 = Pipeline([
  ('prep', preprocessor),
  ('reg', LinearRegression())
])

model2.fit(x_train, y_train)
```

- 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

```
mean_squared_error(y_train, model2.predict(x_train))
```

```
## 16.713690021471173
```

```
mean_squared_error(y_train, fitted)

# The model with engine displacement,
# engine displacement squared and transmission inputs
```

```
## 17.933750031324223
```