Exercise: Regression and Classification Machine Learning

In this exercise, we'll dive deeper into the ML concepts by creating a regression and classification model.

Your tasks for this exercise are:

1. Load the iris dataset into a dataframe
2. Create a LinearRegression model and fit it to the dataset
3. Score the regression model on the dataset and predict it's values
4. Create a RidgeClassifier model and fit it to the dataset, use `alpha=3.0` when initializing the model
5. Score the classification model on the dataset and predict it's values

In [1]:

```
import numpy as np
import pandas as pd
import sklearn
from sklearn import datasets
```

In [2]:

```
# Load in the iris dataset
iris = datasets.load_iris()
```

In [3]:

```
# Create the iris `data` dataset as a dataframe and name the columns with `feature_names`
df = pd.DataFrame(iris["data"], columns=iris["feature_names"])

# Include the target as well
df['target'] = iris["target"]
```

In [4]:

```
# Check your dataframe by `.head()`
df.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Regression ML

In [5]:

```
from sklearn.linear_model import LinearRegression
```

In [6]:

```
# Fit a standard regression model, we've done this in other exercises
reg = LinearRegression().fit(df[iris["feature_names"]], df["target"])
```

In [7]:

```
# Score the model on the same dataset
reg.score(df[iris["feature_names"]], df["target"])
```

Out[7]:

```
0.93042236753315966
```

In [8]:

```
# Predicting values shows they are not that useful to a classification model
reg.predict(df[iris["feature_names"]])
```

Out[8]:

```
array([ -8.26582725e-02,  -3.85897565e-02,  -4.81896914e-02,
         1.26087761e-02,  -7.61081708e-02,   5.68023484e-02,
         3.76259158e-02,  -4.45599433e-02,   2.07050198e-02,
        -8.13030749e-02,  -1.01728663e-01,   8.84875996e-05,
        -8.86050221e-02,  -1.01834705e-01,  -2.26997797e-01,
        -4.36405904e-02,  -3.39982044e-02,  -2.16688605e-02,
        -3.26854579e-02,  -1.22408563e-02,  -4.30562522e-02,
         5.31726003e-02,  -1.23012138e-01,   1.77258467e-01,
         6.81889023e-02,  -4.16362637e-03,   1.00119019e-01,
        -7.09322806e-02,  -8.92083742e-02,   1.99107233e-02,
         1.33606216e-02,   3.35222953e-02,  -1.58465961e-01,
        -1.57523171e-01,  -8.13030749e-02,  -1.03812269e-01,
        -1.49254996e-01,  -8.13030749e-02,  -6.41916305e-03,
        -5.55340896e-02,  -3.33948524e-02,   7.45644153e-02,
        -1.52672524e-02,   2.17673798e-01,   1.39549109e-01,
         3.33738018e-02,  -5.05301301e-02,  -1.45154068e-02,
        -9.07545163e-02,  -6.28360368e-02,   1.20308259e+00,
         1.28451660e+00,   1.32487047e+00,   1.18762080e+00,
         1.31393877e+00,   1.25705298e+00,   1.39745639e+00,
         9.07172433e-01,   1.17656176e+00,   1.24113634e+00,
         9.59294742e-01,   1.28013501e+00,   9.54205881e-01,
         1.31512204e+00,   1.05930184e+00,   1.17232866e+00,
         1.38115786e+00,   9.76734088e-01,   1.35070534e+00,
         1.02311961e+00,   1.59045598e+00,   1.09965570e+00,
         1.41725961e+00,   1.19756726e+00,   1.13040963e+00,
         1.18772685e+00,   1.26542720e+00,   1.49592176e+00,
         1.34168532e+00,   8.55931450e-01,   1.01581766e+00,
         9.32128108e-01,   1.05331264e+00,   1.54772365e+00,
         1.40310615e+00,   1.38055451e+00,   1.30141848e+00,
         1.19062819e+00,   1.16837848e+00,   1.17877271e+00,
         1.20415981e+00,   1.28799785e+00,   1.08043682e+00,
         9.00622332e-01,   1.20435076e+00,   1.11911506e+00,
         1.18452852e+00,   1.15235793e+00,   8.73689093e-01,
         1.16625243e+00,   2.24146289e+00,   1.75264018e+00,
         1.90028407e+00,   1.74143264e+00,   2.00441822e+00,
         2.00431431e+00,   1.60207593e+00,   1.79059214e+00,
         1.76063251e+00,   2.15212358e+00,   1.71469034e+00,
         1.73219558e+00,   1.84240596e+00,   1.81075169e+00,
         2.05316319e+00,   1.95403300e+00,   1.69236016e+00,
         2.04163735e+00,   2.20111558e+00,   1.48615432e+00,
         1.98996282e+00,   1.78575356e+00,   1.96389898e+00,
```

```
        1.59137976e+00,   1.88550825e+00,   1.72019374e+00,
        1.57522972e+00,   1.60005592e+00,   1.91785077e+00,
        1.56166273e+00,   1.79963117e+00,   1.82960982e+00,
        1.97884018e+00,   1.44938775e+00,   1.53269542e+00,
        2.00181829e+00,   2.08524888e+00,   1.69891026e+00,
        1.58832992e+00,   1.80430763e+00,   2.05462443e+00,
        1.85818604e+00,   1.75264018e+00,   2.04633725e+00,
        2.12946589e+00,   1.90725851e+00,   1.68391740e+00,
        1.74623857e+00,   1.98983334e+00,   1.66740449e+00])
```

In [9]:

```
# If we really wanted to, we could do something like round each regression value to an int
# and have it "act" like a classification model
# This is not required, but something to keep in mind for future reference
reg_cls = np.abs(np.rint(reg.predict(df[iris["feature_names"]])))
reg_cls
```

Out[9]:

```
array([ 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.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  2.,  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.,  1.,  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.])
```

In [10]:

```
# Evaluate accuracy
sum(reg_cls == df["target"]) / df.shape[0]
```

Out[10]:

```
0.9733333333333334
```

Classification ML

In [11]:

```
from sklearn.linear_model import RidgeClassifier
```

In [12]:

```
# Fit a ridge classifier, which matches with the problem space of being a classification problem
clf = RidgeClassifier(alpha=3.0).fit(df[iris["feature_names"]], df["target"])
```

In [13]:

```
# Score the model
clf.score(df[iris["feature_names"]], df["target"])
```

Out[13]:

```
0.8599999999999999
```

In [14]:

```
# Predict the class values for the dataset, these will look much better!
clf.predict(df[iris["feature_names"]])
```

Out[14]:

```
array([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, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```