1. Loading Dataset from ScikitLearn

Load `iris` dataset from ScikitLearn using `load_iris()`. Assign the dataset to `X` and the target values to `y`.

In [1]:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Print the dataset keys using `iris_dataset.keys()`

In [2]:

```
print("Keys of iris_dataset:\n", iris_dataset.keys())
```

```
Keys of iris_dataset:
 dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'featur
e_names', 'filename'])
```

Print the names of the categories in the target file

In [3]:

```
print("Target names:", iris_dataset['target_names'])
```

```
Target names: ['setosa' 'versicolor' 'virginica']
```

Print the feature names in the Iris dataset

In [4]:

```
print("Feature names:\n", iris_dataset['feature_names'])
```

```
Feature names:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
```

Print the type of the Iris dataset.

In [15]:

```
print("Type of data:", type(iris_dataset['data']))
```

```
Type of data: <class 'numpy.ndarray'>
```

Print the shape of the Iris dataset.

In [6]:

```
print("Shape of data:", iris_dataset['data'].shape)
```

```
Shape of data: (150, 4)
```

Print the first five rows of the Iris dataset.

In [8]:

```
print("First five rows of data:\n", iris_dataset['data'][:5])
```

```
First five rows of data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

Print the type of the target variable of Iris dataset.

In [9]:

```
print("Type of target:", type(iris_dataset['target']))
```

```
Type of target: <class 'numpy.ndarray'>
```

Print the shape of the target variable of Iris dataset.

In [10]:

```
print("Shape of target:", iris_dataset['target'].shape)
```

```
Shape of target: (150,)
```

Print the entire target variable values of the Iris dataset.

In [12]:

```
print("Target:\n", iris_dataset['target'])
```

```
Target:
 [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 1 1 1
 1 1 1 1 1 1 1 1 1 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Import numpy with `import numpy as np` and use the `numpy.unique()` function to print the unique values of the target variable of Iris dataset

In [14]:

```
import numpy as np
print(np.unique(iris_dataset['target']))
```

```
[0 1 2]
```

Split dataset into train and test datasets using `from sklearn.model_selection import train_test_split`

In [15]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

Print the shape of train/test datasets and the train/test target variables.

In [17]:

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (112, 4)
y_train shape: (112,)
```

In [18]:

```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

Build your K-neighbors classifier for nearest neighbor of 1 using `from sklearn.neighbors import KNeighborsClassifier` , fit the model to your train dataset and make a prediction for teh data point of `[5, 1.9, 1, 0.2]` . Print your prediction class value as an integer and also the corresponding string label.

In [23]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [24]:

```
knn.fit(X_train, y_train)
```

Out[24]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [25]:

```
X_new = np.array([[5, 1.9, 1, 0.2]])
print("X_new.shape:", X_new.shape)
```

```
X_new.shape: (1, 4)
```

In [26]:

```
prediction = knn.predict(X_new)
print("Prediction:", prediction)
print("Predicted target name:",
       iris_dataset['target_names'][prediction])
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

Evaluate your model on the test dataste, print the accuracy of the model

In [27]:

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n", y_pred)
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2
1 0
 2]
```

In [28]:

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

In [29]:

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

2. Loading a Dataset and exploring it

import the modules needed to explore the data

In [30]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Import `auto_mpg.csv` dataset using pandas' `read_csv` function. Print the first three samples from your dataset, print the index range of the observations, and print the column names of your dataset

In [31]:

```
#filename = '/Users/nihalsahan/Documents/DataScience/AutoMPG/auto-mpg.csv'
#data = pd.read_csv(filename, index_col = 'car name')
data = pd.read_csv('auto_mpg.csv')
print(data.head(3))
print(data.index)
print(data.columns)

output = data.iloc[:,0]
features = data.iloc[:, 1:8]

X = features.values
y = output.values
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0

   model year  origin                   car name
0          70       1  chevrolet chevelle malibu
1          70       1          buick skylark 320
2          70       1         plymouth satellite
RangeIndex(start=0, stop=398, step=1)
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model year', 'origin', 'car name'],
      dtype='object')
```

Assign `mpg` column as output and name it as `y` and the rest of the data as the features and assign it to `X` .

In [32]:

```
output = data.iloc[:,0]
features = data.iloc[:, 1:8]

X = features.values
y = output.values
```

Print the shape of the dataset.

In [33]:

```
data.shape
```

Out[33]:

```
(398, 9)
```

Bonus: Check the datasetif there are any missing values in any of the columns using `isnull().any()` functions.

In [34]:

```
data.isnull().any()
```

Out[34]:

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
model year      False
origin          False
car name        False
dtype: bool
```

Check the data types of each feature. Which columns are continuous and which are categorical?

In [36]:

```
# in some datasets, horsepower is object and it needs to be converted to object
# data.horsepower = data.horsepower.astype('float')
data.dtypes
```

Out[36]:

```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

- mpg: continuous
- cylinders: multi-valued discrete
- displacement: continuous
- horsepower: continuous
- weight: continuous
- acceleration: continuous
- model year: multi-valued discrete
- origin: multi-valued discrete
- car name: string (unique for each instance)

Look at the unique elements of horsepower

In [37]:

```
print(data.horsepower.unique())
```

```
[130. 165. 150. 140. 198. 220. 215. 225. 190. 170. 160.  95.  97.  85.
  88.  46.  87.  90. 113. 200. 210. 193. 104. 100. 105. 175. 153. 180.
 110.  72.  86.  70.  76.  65.  69.  60.  80.  54. 208. 155. 112.  92.
 145. 137. 158. 167.  94. 107. 230.  49.  75.  91. 122.  67.  83.  78.
  52.  61.  93. 148. 129.  96.  71.  98. 115.  53.  81.  79. 120. 152.
 102. 108.  68.  58. 149.  89.  63.  48.  66. 139. 103. 125. 133. 138.
 135. 142.  77.  62. 132.  84.  64.  74. 116.  82.]
```

Let's describe data since everything looks in order.

- See the statistical details of the dataset using `describe` and `info` methods.

In [40]:

```
data.describe()
```

Out[40]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.462312	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.199230	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	76.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	95.000000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

In [41]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(4), int64(4), object(1)
memory usage: 28.1+ KB
```

Let's specifically look at the description of the mpg feature

In [42]:

```
pd.set_option('precision', 2)
data.mpg.describe()
```

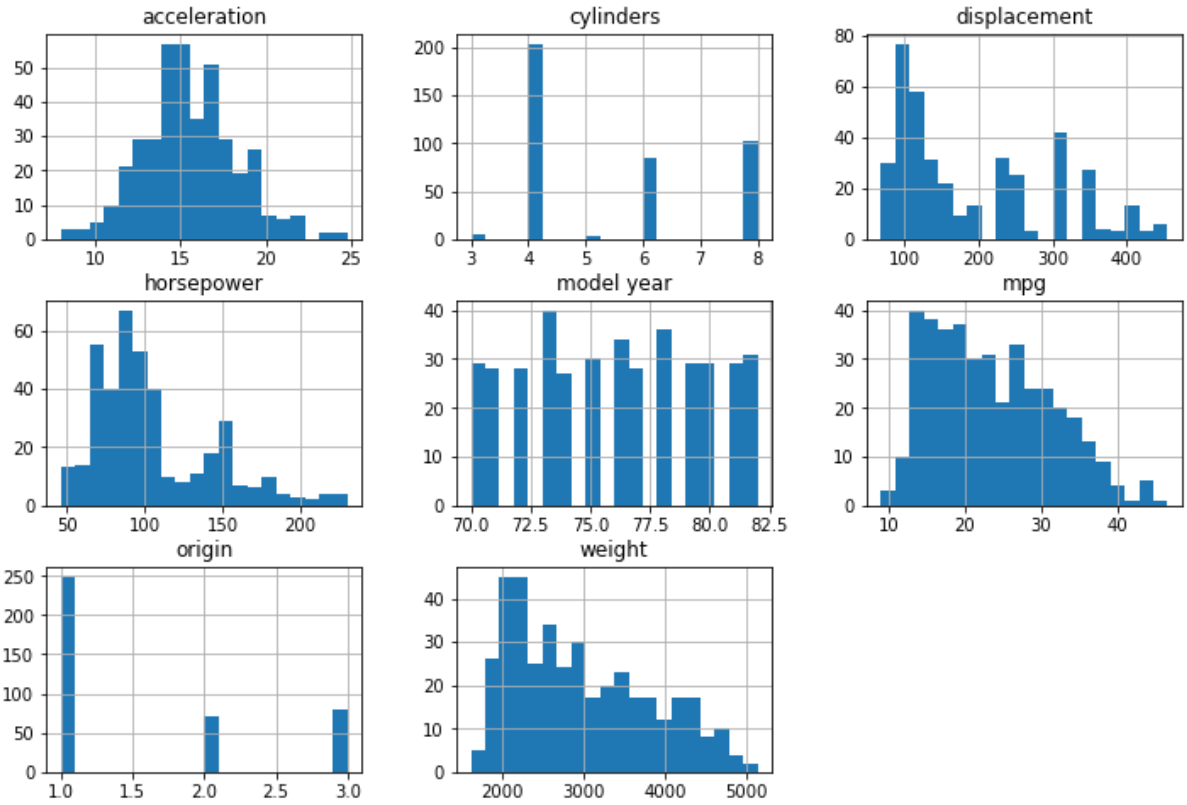
Out[42]:

```
count    398.00
mean      23.51
std        7.82
min        9.00
25%       17.50
50%       23.00
75%       29.00
max       46.60
Name: mpg, dtype: float64
```

Visualize the distribution of the features of the data using `hist` method, use `bins=20`.

In [43]:

```
data.hist(figsize=(12,8),bins=20)
plt.show()
```



- We can see that our variables mpg is skewed to the right. We can also see that our variables are not on the same scale.

BONUS: Visualize the relationships between these data points.

- Create a function to scale your dataset by using the formula $b=\frac{x-min}{max-min}$.
- Using this function, scale `displacement`, `horsepower`, `acceleration`, weight `,` `and` mpg`.
- Create a boxplot of `mpg` for different `origin` values before and after scaling.

In [44]:

```
#In order to visualize some relationship between data points, scale them between 0, 1
def scale(a):
    b = (a-a.min())/(a.max()-a.min())
    return b

# Use a copy of the original dataset
data_scale = data.copy()

data_scale ['displacement'] = scale(data_scale['displacement'])
data_scale['horsepower'] = scale(data_scale['horsepower'])
data_scale ['acceleration'] = scale(data_scale['acceleration'])
data_scale ['weight'] = scale(data_scale['weight'])
data_scale['mpg'] = scale(data_scale['mpg'])

data_scale.head()
```

Out[44]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	0.24	8	0.62	0.46	0.54	0.24	70	1	chevrolet chevelle malibu
1	0.16	8	0.73	0.65	0.59	0.21	70	1	buick skylark 320
2	0.24	8	0.65	0.57	0.52	0.18	70	1	plymouth satellite
3	0.19	8	0.61	0.57	0.52	0.24	70	1	amc rebel sst
4	0.21	8	0.60	0.51	0.52	0.15	70	1	ford torino

In [45]:

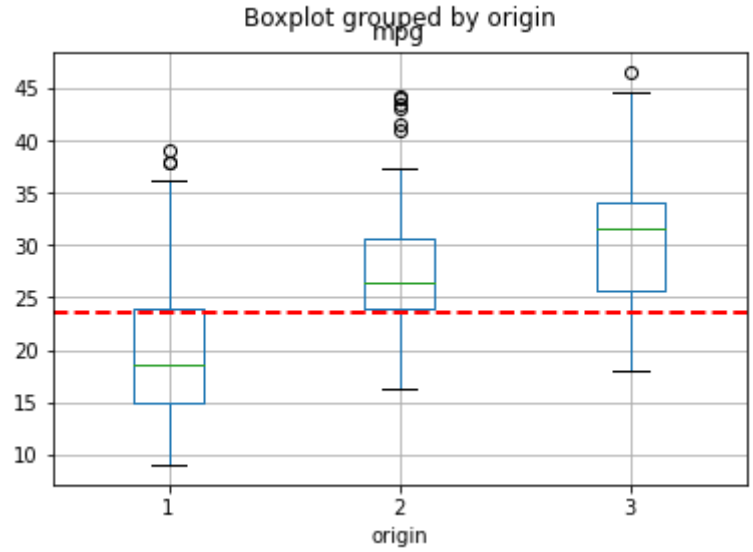
```
#boxplot before the scaling

fig1 = data.boxplot(column = 'mpg', by='origin')
#fig1.axis(ymin=0, ymax=1)
plt.axhline(data.mpg.mean(), color='r', linestyle='dashed', linewidth = 2)
```

```
/Users/gceran/opt/anaconda3/envs/tensorflow_env/lib/python3.6/site-pack
ages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated.
If you meant to do this, you must specify 'dtype=object' when creating
the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[45]:

```
<matplotlib.lines.Line2D at 0x1a150a47b8>
```



In [46]:

```
#boxplot after the scaling
var = 'origin'
data_plt = pd.concat([data_scale['mpg'], data_scale[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="mpg", data=data_plt)
fig.axis(ymin=0, ymax=1)
plt.axhline(data_scale.mpg.mean(),color='r',linestyle='dashed',linewidth=2)
```

Out[46]:

```
<matplotlib.lines.Line2D at 0x1a15538278>
```