Machine Learning with Python

Week 1: Introduction to Machine Learning

Topics:

- regression
- classification
- clustering
- scikit learn
- scipy

Introduction to Machine Learning

Machine learning:

- the subfield of computer science that gives computers the ability to learn without being explicitly programmed

Major machine learning techniques:

- regression/estimation
  - used to predict continuous values (price of house, CO2 emission)
- classification
  - predict the item class / category of a case
- clustering
  - finding the structures of data; summarization
- associations
  - associating frequent co-occurring items / events
- anomaly detection
  - discovering abnormal and unusual cases
- sequence mining
  - predicting next events; click stream (markov model,HMM)
- dimension reduction
  - reducing the size of data (PCA)
- recommendation systems
  - recommending items

Python for Machine Learning

Python libraries for machine learning:

- NumPy: math library to work with n-dimensional arrays in python
- SciPy: numerical algorithm and domain-specific toolboxes
- Matplotlib: 2D, 3D plotting

- Pandas: high performance data structures
- scikit-learn: algorithms and tools for machine learning
  - classification, regression, and clustering algorithms
  - works with numpy and scipy

Supervised vs Unsupervised

Supervised Learning:

- we 'teach the model', then with that knowledge it can predict unknown or future instances
- teach the model with labeled data
- has more evaluation methods than unsupervised learning
- controlled environment
- two types of supervised learning techniques:
  - classification: the process of predicting discrete class labels or categories
  - regression: the process of predicting continuous values

Unsupervised Learning:

- the model works on its own to discover information.
- the model works on the unlabeled dataset to discover information that may not be visible to the human eye.
- usually has more difficult algorithms than supervised learning
- unsupervised learning techniques:
  - dimension reduction: aka feature selection reduces redundant features
  - density estimation: explores the data to find structure within it
  - market basket analysis: modeling technique based on theory that if you buy one set of items, you're more likely to buy a different set of items too
  - clustering: clustering is grouping of data points or objects that are somehow similar by:
    - discovering structure
    - summarization
    - anomaly detection

Week 2: Introduction to Regression

Introduction to Regression

What is Regression?

- regression is the process of predicting a continuous value
- two types of variables in regression analysis
  - dependent variable (y) - the state, target, or goal variable we want to predict
    - must be a continuous value
  - independent variable(s) (x) - the explanatory variables that cause the state variable's value
    - can be continuous, discrete, or categorical

- Regression modelling: use historical data to create a model, which will predict the expected target variable value.

Simple Regression:

- one independent variable is used to estimate a dependent/target variable.
- two types (dependent on the nature of the relationship between the dependent and independent variables):
  - simple linear regression
  - simple non-linear regression

Multiple Regression:

- More than one independent variable is used to estimate a dependent variable.
- two types (dependent on the nature of the relationship between the dependent and indepdendent variables):
  - multiple linear regression
  - multiple non-linear regression

Regression algorithms:

- ordinal regression
- poisson regression
- fast forest quantile regression
- linear, polynomial, lasso, stepwise, ridge regression
- bayesian linear regression
- neural network regression
- decision forest regression
- boosted decision tree regression
- k-nearest neighbors (KNN)

Simple Linear Regression

Regression / Fitting Line:

- ${\hat{y} = b + mx}$
  - ${\hat{y}}$: predicted value
  - ${x}$: predictor / independent variable
  - ${m}$: slope/gradient of the line
  - ${b}$: intercept
- linear regression estimates the coefficients ${m, b}$ of the line
- we need to calculate the coefficients such that we get the line that best fits the data

Finding the best fit:

- ${Residual Error = y - \hat{y}}$
- The mean of all residual errors tells us how well or how poorly the line fits the data. We want the smallest MSE.

    - ${MSE = (1/n)\sum ({y}_{i} - \hat{y}_{i})^2}$
- Two options to find smallest MSE
    - Mathematical
        - calculate ${b, m}$ using mathematical formulas where you get the mean values ${\bar{x}}$ and ${\bar{y}}$ then subtract all observations of ${\bar{x}}$ and ${\bar{y}}$ from x and y respectively
    - Optimization

Model Evaluation in Regression Models

Evaluation:

- methods:
    - train and test on the same dataset
        - the whole dataset is used for training, and then a small portion of the dataset is used to test the model for accuracy
        - results in high 'training accuracy', but low 'out of sample accuracy'
            - high training accuracy could be the result of overfitting, which means the model is overly trained to the dataset, which may capture noise and produce a non-generalized model
    - train/test split
        - the dataset is split so some of it is used for training and the rest is used for testing.
        - more accurate evaluation on 'out of sample accuracy'.
        - highly dependent on the datasets on which the model was trained and tested
    - k-fold cross validation
        - resolves most of the issues with train/test splits.
        - the dataset is split into k 'folds'. each fold has the chance to be used as the testing dataset and as the training dataset.
        - then the average accuracy of each grouping is taken.
        - it is essentially multiple train/test splits used on the same data.

Evaluation Metrics in Regression Models

Accuracy Metrics:

- Error: measure how far the data is from the fitted regression line
- Mean Absolute Error:
    - simplest error to understand
    - ${MAE = (1/n)\sum |{y}_{i} - \hat{y}_{i}|}$
- Mean Squared Error:
    - more popular than MAE as it is geared towards larger errors
    - ${MSE = (1/n)\sum ({y}_{i} - \hat{y}_{i})^2}$
- Root Mean Squared Error:
    - square root of MSE.
    - is interpretable in the same units as the response vector or y units, which makes it easy to relate the information to the data.

    - ${RMSE = (\sqrt(1/n)\sum ({y}_{i} - \hat{y}_{i})^2}))$
- Relative Absolute Error / Residual Sum of Square:
    - takes the total absolute error and normalizes
    - ${MSE = (\sum ({y}_{i} - \hat{y}_{i}))) / (\sum ({y}_{i} - \bar{y})$
- Relative Squared Error:
    - similar to relative absolute error
    - widely adopted by data science community since it is used for calculating R-squared
        - R-squared is not an error persay, but a popular metric for accuracy. It represents how closely the data values are fitted to the regression line. The higher R-squared, the better the model fits.
        - ${R^2 = 1-RSE}$
    - ${RSE = (\sum ({y}_{i} - \hat{y}_{i})^2}) / (\sum ({y}_{i} - \bar{y})^2$

Lab: Simple Linear Regression

In [1]:
```
# import packages
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

In [2]:
```
# download the data
!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-storage.appdoma
```

```
--2021-08-11 17:28:41--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/FuelConsumptio
nCo2.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72629 (71K) [text/csv]
Saving to: 'FuelConsumption.csv'

     0K .......... .......... .......... .......... .......... 70%  638K 0s
    50K .......... ..........                                  100% 30.2M=0.08s

2021-08-11 17:28:41 (897 KB/s) - 'FuelConsumption.csv' saved [72629/72629]
```

In [3]:
```
# read the data
df = pd.read_csv("FuelConsumption.csv")

# take a look at the dataset
df.head()
```

Out[3]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244

In [4]:
```
# summarize the data
df.describe()
```

Out[4]:

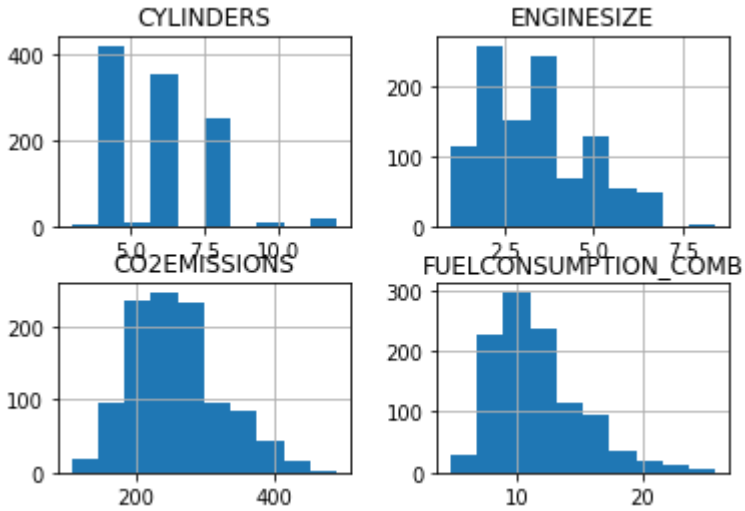
	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000
mean	2014.0	3.346298	5.794752	13.296532	9.474602	11.580881	26.441425	256.228679
std	0.0	1.415895	1.797447	4.101253	2.794510	3.485595	7.468702	63.372304
min	2014.0	1.000000	3.000000	4.600000	4.900000	4.700000	11.000000	108.000000
25%	2014.0	2.000000	4.000000	10.250000	7.500000	9.000000	21.000000	207.000000
50%	2014.0	3.400000	6.000000	12.600000	8.800000	10.900000	26.000000	251.000000
75%	2014.0	4.300000	8.000000	15.550000	10.850000	13.350000	31.000000	294.000000
max	2014.0	8.400000	12.000000	30.200000	20.500000	25.800000	60.000000	488.000000

In [5]:
```
# explore some features
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

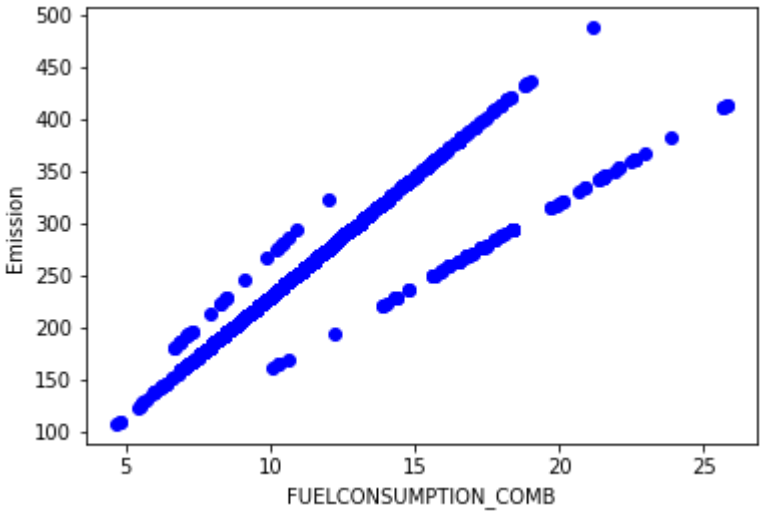
Out[5]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

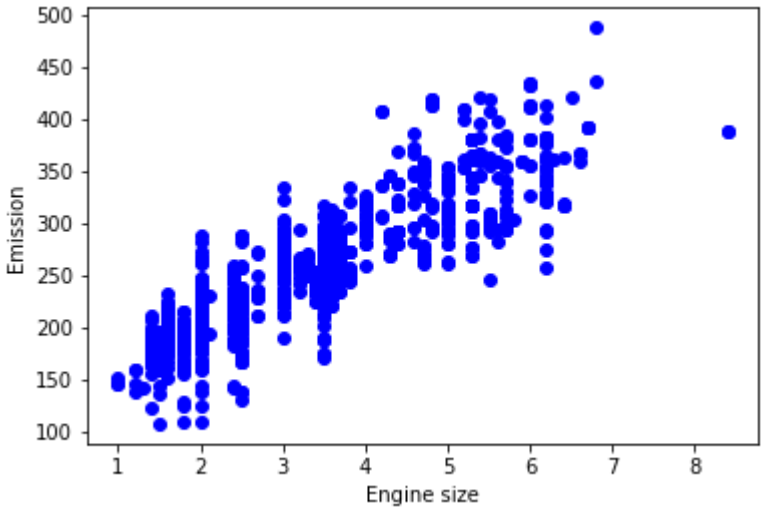
In [6]:
```
# plot the features
viz = cdf[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION_COMB']]
viz.hist()
plt.show()
```



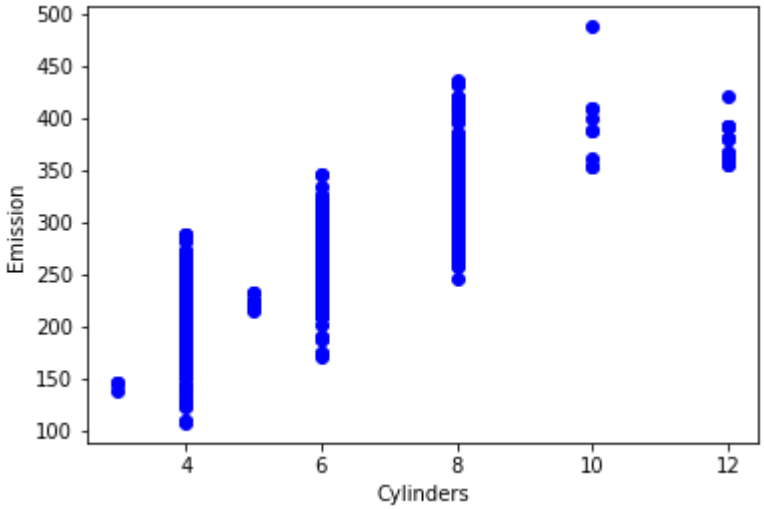
In [7]:
```
# plot fuelconsumption against emission
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```



In [8]:
```
# plot enginesize against emissions
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```

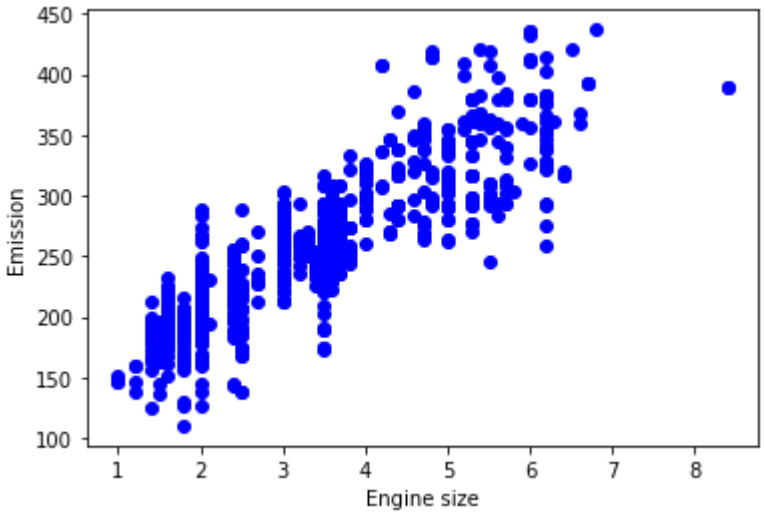


In [9]:
```
# plot cylinders against emissions
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Cylinders")
plt.ylabel("Emission")
plt.show()
```



In [10]:
```
# split the dataset into train (80%) and test sets (20%)
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

In [11]:
```
# train the data and plot it
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



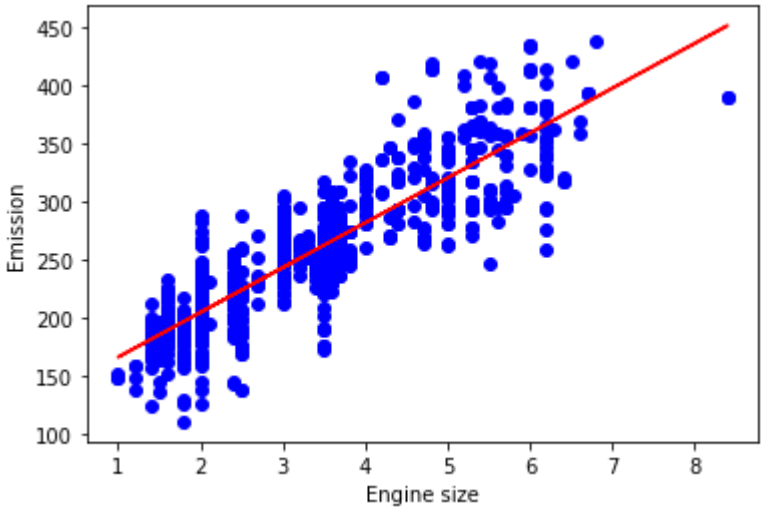
In [12]:
```
# use sklearn to model data
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[38.58212014]]
Intercept:  [126.89417394]
```

As mentioned before, **Coefficient** and **Intercept** in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data. Notice that all of the data must be available to traverse and calculate the parameters.

In [13]:
```
# plot the fit line over the data
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[13]:
```
Text(0, 0.5, 'Emission')
```



Evaluation

We compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, lets use MSE here to calculate the accuracy of our model based on the test set:

- Mean Absolute Error: It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just average error.
- Mean Squared Error (MSE): Mean Squared Error (MSE) is the mean of the squared error. It's more popular than Mean Absolute Error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.
- Root Mean Squared Error (RMSE).
- R-squared is not an error, but rather a popular metric to measure the performance of your regression model. It represents how close the data points are to the fitted regression line. The higher the R-squared value, the better the model fits your data. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

In [15]:
```
# get the MAE, MSE, and R-squared
from sklearn.metrics import r2_score

test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])
test_y_ = regr.predict(test_x)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , test_y_) )
```

```
Mean absolute error: 24.52
Residual sum of squares (MSE): 1048.91
```

```
R2-score: 0.77
```

Multiple Linear Regression

Applications for MLR:

- independent variables' effectiveness on prediction
- predicting impacts of changes

Find optimized parameters for the model:

- Ordinary Least Squares
  - linear algebra operations estimate the optimal values
  - takes a long time for large datasets (over 10k rows)
- Optimization algorithm
  - gradient descent
    - good approach if you have a very large dataset

Making predictions with MLR:

- solve the equation ${\hat{y} = \theta^TX}$
- MLR estimates the relative importance / impact of each predictor.

Lab: Multiple Linear Regression

In [16]:
```
# import packages
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

In [17]:
```
# download data
!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-storage.appdoma
```

```
--2021-08-11 17:51:38--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/FuelConsumptio
nCo2.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72629 (71K) [text/csv]
Saving to: 'FuelConsumption.csv'

     0K .......... .......... .......... .......... .......... 70%  639K 0s
    50K .......... ..........                                   100% 38.7M=0.08s

2021-08-11 17:51:39 (901 KB/s) - 'FuelConsumption.csv' saved [72629/72629]
```

In [18]:
```
# read data
df = pd.read_csv("FuelConsumption.csv")
```

```
# take a look at the dataset
df.head()
```

Out[18]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244

In [19]:

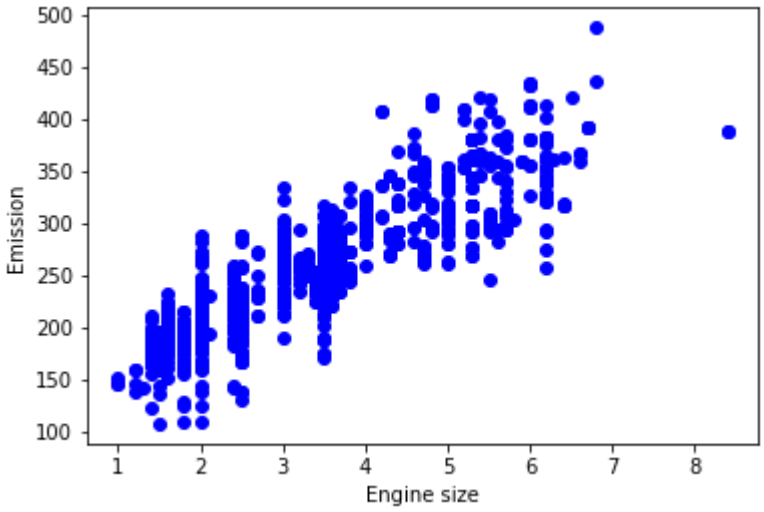
```
# select features we want to use for regression
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCO
cdf.head(9)
```

Out[19]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	9.9	6.7	8.5	196
1	2.4	4	11.2	7.7	9.6	221
2	1.5	4	6.0	5.8	5.9	136
3	3.5	6	12.7	9.1	11.1	255
4	3.5	6	12.1	8.7	10.6	244
5	3.5	6	11.9	7.7	10.0	230
6	3.5	6	11.8	8.1	10.1	232
7	3.7	6	12.8	9.0	11.1	255
8	3.7	6	13.4	9.5	11.6	267

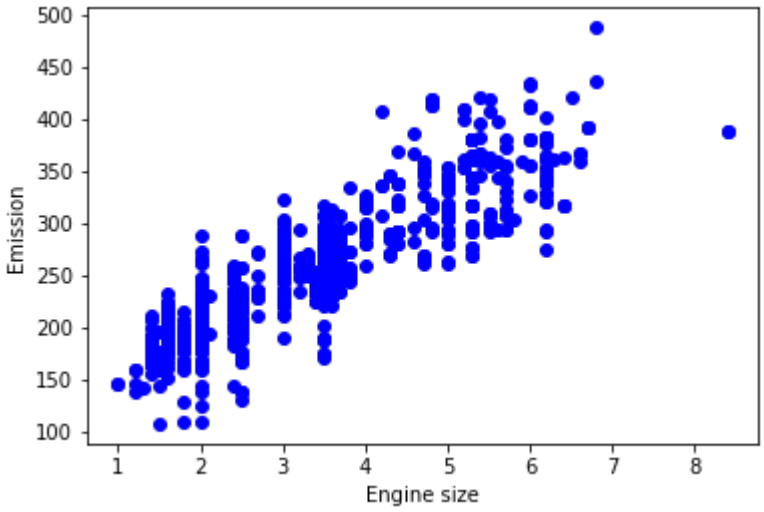
In [20]:

```
# plot emissions with respect to engine size
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



In [21]:
```
# split the dataset into training and test sets
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

In [22]:
```
# plot the training set
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



In [23]:
```
# train and fit the model
from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (x, y)
# The coefficients
print ('Coefficients: ', regr.coef_)
```

```
Coefficients:  [[11.04172732  7.78307829  9.43270081]]
```

In reality, there are multiple variables that impact the Co2emission. When more than one

independent variable is present, the process is called multiple linear regression. An example of multiple linear regression is predicting co2emission using the features FUELCONSUMPTION_COMB, EngineSize and Cylinders of cars. The good thing here is that multiple linear regression model is the extension of the simple linear regression model.

As mentioned before, **Coefficient** and **Intercept** are the parameters of the fitted line. Given that it is a multiple linear regression model with 3 parameters and that the parameters are the intercept and coefficients of the hyperplane, sklearn can estimate them from our data. Scikit-learn uses plain Ordinary Least Squares method to solve this problem.

Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by minimizing the sum of the squares of the differences between the target dependent variable and those predicted by the linear function. In other words, it tries to minimizes the sum of squared errors (SSE) or mean squared error (MSE) between the target variable (y) and our predicted output ($\hat{y}$) over all samples in the dataset.

OLS can find the best parameters using of the following methods:

- Solving the model parameters analytically using closed-form equations
- Using an optimization algorithm (Gradient Descent, Stochastic Gradient Descent, Newton's Method, etc.)

In [25]:
```
# make a prediction and get the MSE and variance
y_hat= regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
x = np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
y = np.asanyarray(test[['CO2EMISSIONS']])
print("Residual sum of squares: %.2f"
      % np.mean((y_hat - y) ** 2))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x, y))
```

```
Residual sum of squares: 523.99
Variance score: 0.87
```

In [26]:
```
# use a multiple linear regression model using fuel consumption in city and fuel consum
# do these have better accuracy than fuel consumption combined?
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTI
y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (x, y)
print ('Coefficients: ', regr.coef_)
y_= regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION
x = np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTIO
y = np.asanyarray(test[['CO2EMISSIONS']])
print("Residual sum of squares: %.2f"% np.mean((y_ - y) ** 2))
print('Variance score: %.2f' % regr.score(x, y))
```

```
Coefficients:  [[11.25786246  7.18081555  6.52617665  2.46749732]]
Residual sum of squares: 529.82
```

```
Variance score: 0.86
```

Non-Linear Regression

Polynomial Regression:

- the relationship between independent x and dependent y is modeled as an nth degree polynomial in x.
- with many types of regression to choose from, there's a good chance than one will fit the data shape.
- curvy data can be modeled by polynomial regression
- a polynomial regression model can be transformed into a linear regression model.
- polynomial regressio models can fit using the methods of least squares
- to be a non-linear function, y-hat must be a non-linear function of the parameters, not necessarily the features x.

Lab: Polynomial Regression

In [27]:
```
# import libraries
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

In [28]:
```
# download data
!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-storage.appdoma
```

```
--2021-08-11 18:10:47--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/FuelConsumptio
nCo2.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72629 (71K) [text/csv]
Saving to: 'FuelConsumption.csv'

     0K .......... .......... .......... .......... .......... 70%  415K 0s
    50K .......... ..........                                   100% 37.7M=0.1s

2021-08-11 18:10:48 (586 KB/s) - 'FuelConsumption.csv' saved [72629/72629]
```

In [29]:
```
# read the data
df = pd.read_csv("FuelConsumption.csv")

# take a look at the dataset
df.head()
```

Out[29]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196

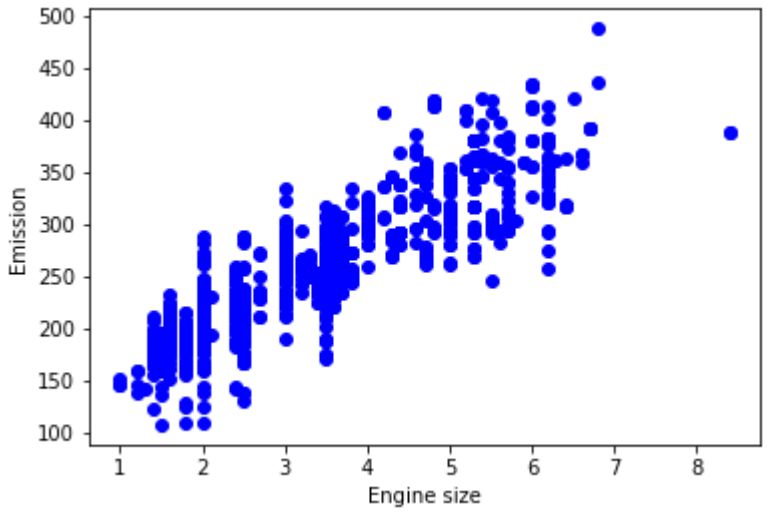
	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244

In [31]:
```
# select features to use for regression
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

Out[31]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

In [32]:
```
# plot emission values with respect to engine size
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



In [33]:
```
# create train and test datasets
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

Sometimes, the trend of data is not really linear, and looks curvy. In this case we can use Polynomial regression methods. In fact, many different regressions exist that can be used to fit whatever the dataset looks like, such as quadratic, cubic, and so on, and it can go on and on to infinite degrees.

In essence, we can call all of these, polynomial regression, where the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x. Lets say you want to have a polynomial regression (let's make 2 degree polynomial):

$$y = b + \theta\_1 x + \theta\_2 x^2$$
Now, the question is: how we can fit our data on this equation while we have only x values, such as **Engine Size**? Well, we can create a few additional features: 1, x, and x^2.

PolynomialFeatures() function in Scikit-learn library, drives a new feature sets from the original feature set. That is, a matrix will be generated consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, lets say the original feature set has only one feature, *ENGINESIZE*. Now, if we select the degree of the polynomial to be 2, then it generates 3 features, degree=0, degree=1 and degree=2:

fit_transform takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2).

The equation and the sample example is displayed below.

$$ \begin{bmatrix} v\_1\\\\ v\_2\\\\ \vdots\\\\ v_n \end{bmatrix}\longrightarrow \begin{bmatrix} \[1 & v\_1 & v\_1^2]\\\\ \[1 & v\_2 & v\_2^2]\\\\ \vdots & \vdots & \vdots\\\\ \[1 & v_n & v_n^2] \end{bmatrix} $$$$ \begin{bmatrix} 2.\\\\ 2.4\\\\ 1.5\\\\ \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} \[1 & 2. & 4.]\\\\ \[1 & 2.4 & 5.76]\\\\ \[1 & 1.5 & 2.25]\\\\ \vdots & \vdots & \vdots\\\\ \end{bmatrix} $$
It looks like feature sets for multiple linear regression analysis, right? Yes. It Does. Indeed, Polynomial regression is a special case of linear regression, with the main idea of how do you select your features. Just consider replacing the x with x_1, x_1^2 with x_2, and so on. Then the degree 2 equation would be turn into:

$$y = b + \theta\_1 x\_1 + \theta\_2 x\_2$$
Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.

so we can use **LinearRegression()** function to solve it:

In [34]:
```
# train and fit the model
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
```

```
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])

test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly
```

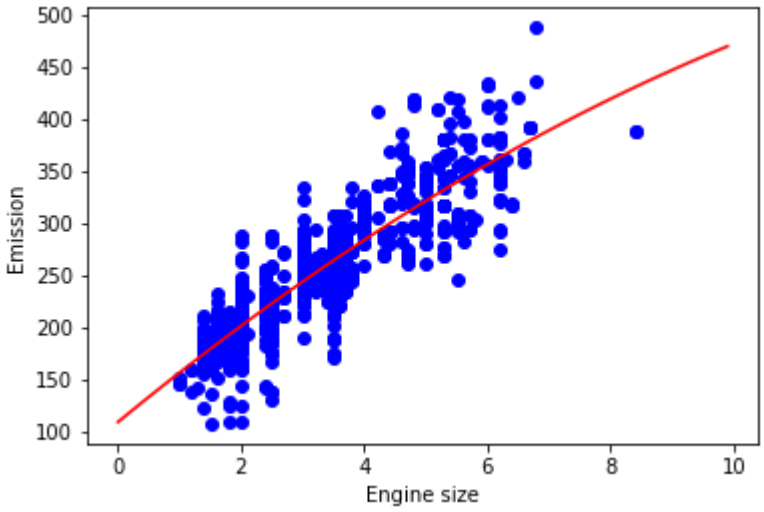
Out[34]:
```
array([[ 1.  ,  2.  ,  4.  ],
       [ 1.  ,  2.4 ,  5.76],
       [ 1.  ,  1.5 ,  2.25],
       ...,
       [ 1.  ,  3.  ,  9.  ],
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24]])
```

In [35]:
```
# get the coefficients and intercept
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

```
Coefficients:  [[ 0.         48.58609486 -1.23008598]]
Intercept:  [109.58215371]
```

In [36]:
```
# plot the model and regression line
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf.intercept_[0]+ clf.coef_[0][1]*XX+ clf.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[36]:
```
Text(0, 0.5, 'Emission')
```



In [37]:
```
# get the MAE, MSE, and R-squared
from sklearn.metrics import r2_score
```

```
test_x_poly = poly.fit_transform(test_x)
test_y_ = clf.predict(test_x_poly)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,test_y_ ) )
```

```
Mean absolute error: 22.41
Residual sum of squares (MSE): 874.29
R2-score: 0.78
```

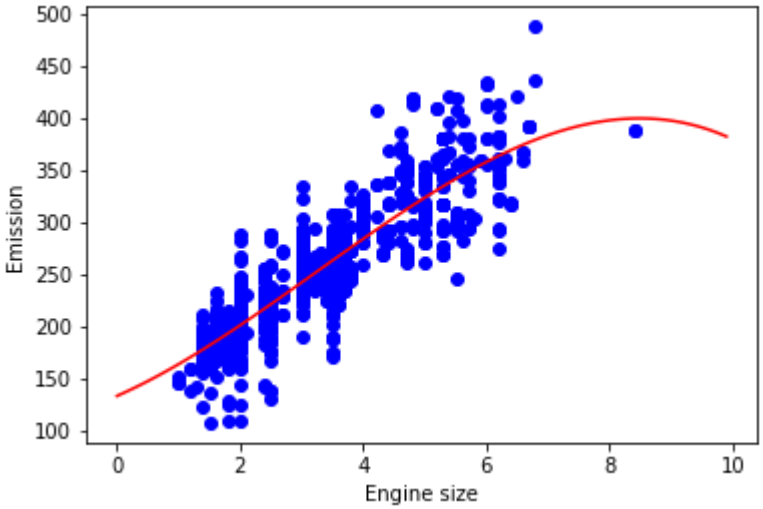
In [38]:

```
# use a polynomial regression with degree 3. Does it result in better accuracy?
poly3 = PolynomialFeatures(degree=3)
train_x_poly3 = poly3.fit_transform(train_x)
clf3 = linear_model.LinearRegression()
train_y3_ = clf3.fit(train_x_poly3, train_y)
```

In [39]:

```
# The coefficients
print ('Coefficients: ', clf3.coef_)
print ('Intercept: ',clf3.intercept_)
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf3.intercept_[0]+ clf3.coef_[0][1]*XX + clf3.coef_[0][2]*np.power(XX, 2) + clf3.

# plot
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
test_x_poly3 = poly3.fit_transform(test_x)
test_y3_ = clf3.predict(test_x_poly3)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y3_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y3_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,test_y3_ ) )
```

```
Coefficients:  [[ 0.         26.01379794  4.95707174 -0.50959903]]
Intercept:  [133.61463614]
Mean absolute error: 22.46
Residual sum of squares (MSE): 875.70
R2-score: 0.78
```



Non-Linear Regression

In [41]:
```
# import packages
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

In [42]:
```
# download data
!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-storage.appdoma
```

```
--2021-08-11 18:16:15--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/FuelConsumptio
nCo2.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72629 (71K) [text/csv]
Saving to: 'FuelConsumption.csv'

     0K .......... .......... .......... .......... .......... 70%  412K 0s
    50K .......... ..........                                   100% 32.4M=0.1s

2021-08-11 18:16:16 (581 KB/s) - 'FuelConsumption.csv' saved [72629/72629]
```

In [43]:
```
# read the data
df = pd.read_csv("FuelConsumption.csv")

# take a look at the dataset
df.head()
```

Out[43]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244

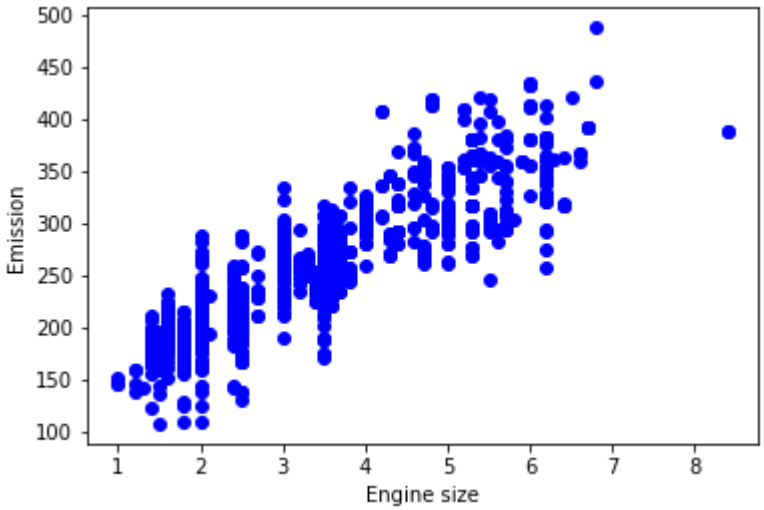
In [44]:
```
# select features to use for regression
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

Out[44]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

In [45]:
```
# plot emissions with respect to engine size
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



In [46]:
```
# create train and test datasets
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

Sometimes, the trend of data is not really linear, and looks curvy. In this case we can use Polynomial regression methods. In fact, many different regressions exist that can be used to fit whatever the dataset looks like, such as quadratic, cubic, and so on, and it can go on and on to infinite degrees.

In essence, we can call all of these, polynomial regression, where the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x. Lets say you want to have a polynomial regression (let's make 2 degree polynomial):

$$y = b + \theta\_1 x + \theta\_2 x^2$$
Now, the question is: how we can fit our data on this equation while we have only x values, such as **Engine Size**? Well, we can create a few additional features: 1, x, and x^2.

PolynomialFeatures() function in Scikit-learn library, drives a new feature sets from the original feature set. That is, a matrix will be generated consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, lets say the original feature set has only one feature, *ENGINESIZE*. Now, if we select the degree of the polynomial to be 2, then it generates 3 features, degree=0, degree=1 and degree=2:

In [47]:
```
# train and fit the model
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])

test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly
```

Out[47]:
```
array([[ 1.  ,  2.  ,  4.  ],
       [ 1.  ,  2.4 ,  5.76],
       [ 1.  ,  1.5 ,  2.25],
       ...,
       [ 1.  ,  3.  ,  9.  ],
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24]])
```

fit_transform takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2).

The equation and the sample example is displayed below.

$$ \begin{bmatrix} v\_1\\\\ v\_2\\\\ \vdots\\\\ v_n \end{bmatrix}\longrightarrow \begin{bmatrix} \[1 & v\_1 & v\_1^2]\\\\ \[1 & v\_2 & v\_2^2]\\\\ \vdots & \vdots & \vdots\\\\ \[1 & v_n & v_n^2] \end{bmatrix} $$$$ \begin{bmatrix} 2.\\\\ 2.4\\\\ 1.5\\\\ \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} \[1 & 2. & 4.]\\\\ \[1 & 2.4 & 5.76]\\\\ \[1 & 1.5 & 2.25]\\\\ \vdots & \vdots & \vdots\\\\ \end{bmatrix} $$

It looks like feature sets for multiple linear regression analysis, right? Yes. It Does. Indeed, Polynomial regression is a special case of linear regression, with the main idea of how do you select your features. Just consider replacing the x with x_1, x_1^2 with x_2, and so on. Then the degree 2 equation would be turn into:

$$y = b + \theta\_1 x\_1 + \theta\_2 x\_2$$

Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.

so we can use **LinearRegression()** function to solve it:

In [48]:
```
# solve the equation
clf = linear_model.LinearRegression()
```

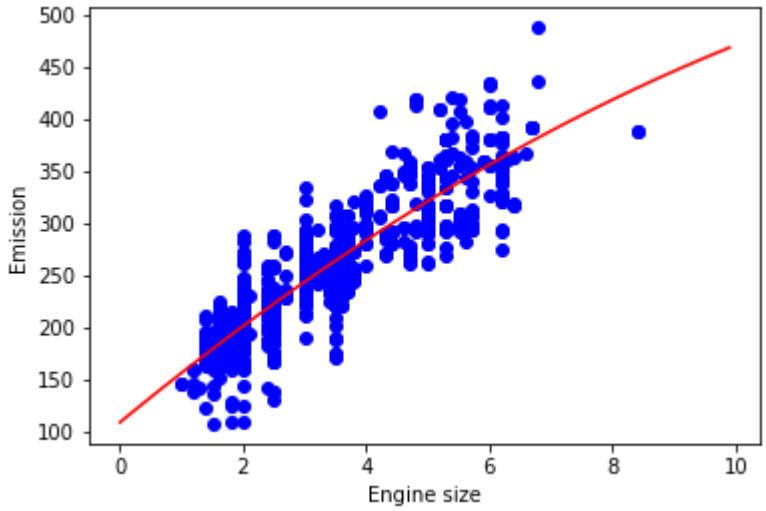
```
train_y_ = clf.fit(train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

```
Coefficients:  [[ 0.         48.68561762 -1.24991389]]
Intercept:  [109.30437796]
```

As mentioned before, **Coefficient** and **Intercept** , are the parameters of the fit curvy line. Given that it is a typical multiple linear regression, with 3 parameters, and knowing that the parameters are the intercept and coefficients of hyperplane, sklearn has estimated them from our new set of feature sets. Lets plot it:

In [49]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf.intercept_[0]+ clf.coef_[0][1]*XX+ clf.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[49]:
```
Text(0, 0.5, 'Emission')
```



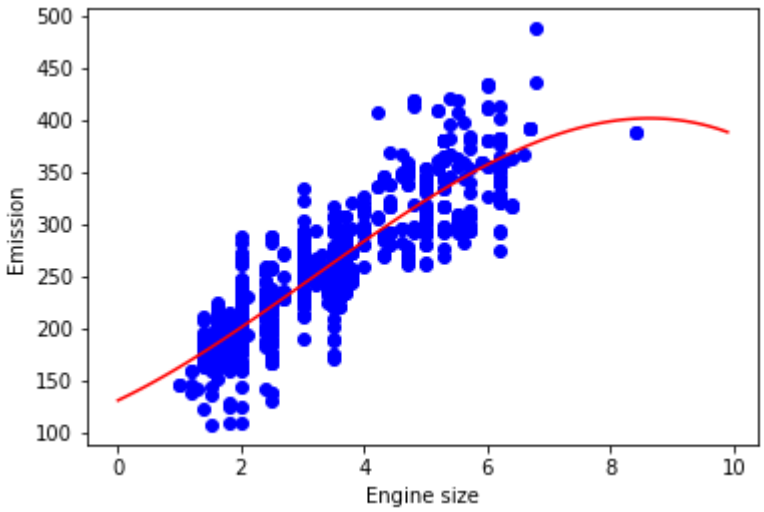
In [50]:
```
# evaluate the model
from sklearn.metrics import r2_score

test_x_poly = poly.fit_transform(test_x)
test_y_ = clf.predict(test_x_poly)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,test_y_ ) )
```

```
Mean absolute error: 23.43
Residual sum of squares (MSE): 962.75
R2-score: 0.74
```

In [51]:
```
# use a polynomial regression with the dataset with degree 3. does it have better accur
poly3 = PolynomialFeatures(degree=3)
train_x_poly3 = poly3.fit_transform(train_x)
clf3 = linear_model.LinearRegression()
train_y3_ = clf3.fit(train_x_poly3, train_y)
```

```
# The coefficients
print ('Coefficients: ', clf3.coef_)
print ('Intercept: ',clf3.intercept_)
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf3.intercept_[0]+ clf3.coef_[0][1]*XX + clf3.coef_[0][2]*np.power(XX, 2) + clf3.
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
test_x_poly3 = poly3.fit_transform(test_x)
test_y3_ = clf3.predict(test_x_poly3)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y3_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y3_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,test_y3_ ) )
```

```
Coefficients:  [[ 0.         28.08579145  4.38934375 -0.46466733]]
Intercept:  [131.30907963]
Mean absolute error: 23.39
Residual sum of squares (MSE): 960.30
R2-score: 0.75
```



R-squared is a value between 0 and 1. The closer to 1, the more correlated the variables are (and lower variance).

MSE: the larger the MSE, the larger the error.

Week 3: Classification

Introduction to Classification

Classification:

- a supervised learning approach
- categorizing some unknown items into discrete sets of categories or classes
- the target attribute is a categorical variable
- classification determines the class label for an unlabeled test case
- classification can be binary or multi-class

Use cases:

- which category a customer belongs to
- whether to switch to another provider / brand
- whether a customer will respond to a campaign

Classification Algorithms in Machine Learning:

- decision trees (ID3, C4.5, C5.0)
- naive bayes
- linear disciminant analysis
- k-nearest neighbor
- logistic regression
- neural networks
- support vector machines (SVM)

K-Nearest Neighbors

KNN:

- classify incoming / unknown observations the same as the nearest observations (graphically speaking), or take the mode of the k-nearest neighbors
- KNN is a method for classifying cases based on their similarity to other cases
- cases that are near each other are said to be 'neighbors'
- KNN is based on similar cases with the same class labels are near each other.
- KNN can also be used for regression

KNN Algorithm:

1. pick value for k
    - how to find the best k for your dataset:
        - reserve part of the data for testing the accuracy of the model
        - then choose K = 1 and use hte training part for modeling and calculate the accuracy of prediction using all samples in the test set.
        - Repeat this process increasing K and see which K is the best fit for the model.
2. calculate distance of unknown case from all cases
    - euclidean distance
3. select the k-observation in the training data that are 'nearest' to the unknown data point
4. predict the response of the unknown data point using the most popular response value from the k nearest neighbors.

Evaluation Metrics for Classification

Classification Accuracy Measurements:

- Jaccard Index
    - intersection of actual & predicted values / union of actual & predicted values
- F1-Score
    - confusion matrix

- true positive / negatives, false positive/negatives count
- precision = TP / (TP+FP)
- recall = TP / (TP+FN)
- F1score = 2(precision x recall)/(precision + recall)
  - score = 1 is perfect accuracy

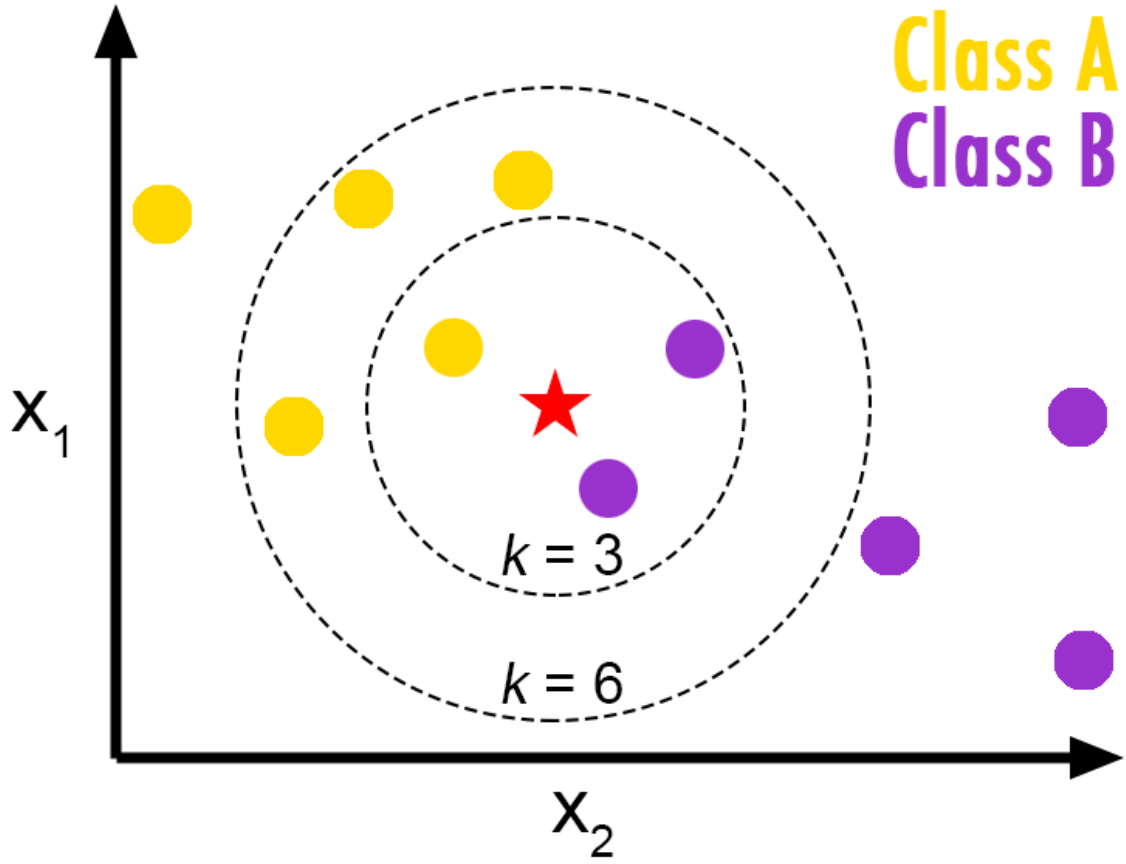
- Log Loss
  - perfromance of a classifier where the predicted output is a probability value between 0 and 1, where 0 means higher accuracy

Lab: KNN

In this Lab you will load a customer dataset, fit the data, and use K-Nearest Neighbors to predict a data point. But what is **K-Nearest Neighbors**?

K-Nearest Neighbors is a supervised learning algorithm. Where the data is 'trained' with data points corresponding to their classification. To predict the class of a given data point, it takes into account the classes of the 'K' nearest data points and chooses the class in which the majority of the 'K' nearest data points belong to as the predicted class.

Here's an visualization of the K-Nearest Neighbors algorithm



In this case, we have data points of Class A and B. We want to predict what the star (test data point) is. If we consider a k value of 3 (3 nearest data points), we will obtain a prediction of Class B. Yet if we consider a k value of 6, we will obtain a prediction of Class A.

In this sense, it is important to consider the value of k. Hopefully from this diagram, you should get a sense of what the K-Nearest Neighbors algorithm is. It considers the 'K' Nearest Neighbors (data points) when it predicts the classification of the test point.

In [1]:
```
# load libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
%matplotlib inline
```

In [2]:
```
# download dataset
!wget -O teleCust1000t.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain
```

```
--2021-08-12 15:55:21--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/teleCust1000t.
csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 36047 (35K) [text/csv]
Saving to: 'teleCust1000t.csv'

     0K .......... .......... .......... .....                 100%  873K=0.04s

2021-08-12 15:55:21 (873 KB/s) - 'teleCust1000t.csv' saved [36047/36047]
```

In [3]:
```
# load into dataframe
df = pd.read_csv('teleCust1000t.csv')
df.head()
```

Out[3]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

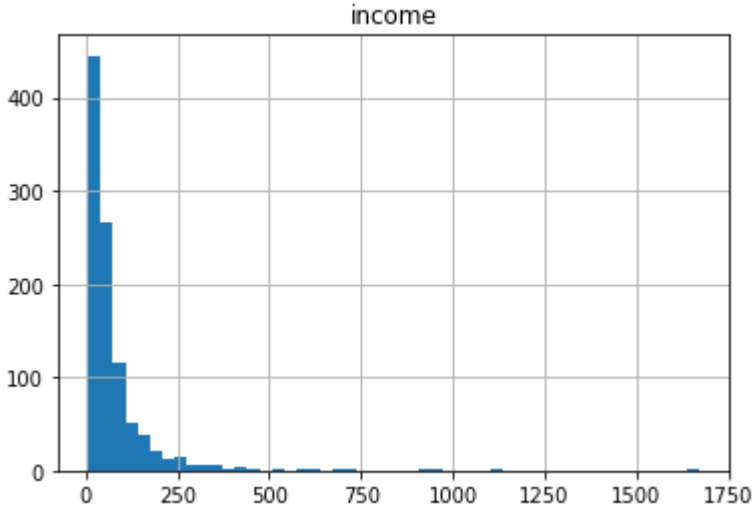
In [4]:
```
# see how many of each class is in the set
df['custcat'].value_counts()
```

Out[4]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

In [5]:
```
# explore data with a histogram
```

```
df.hist(column='income', bins=50)
```

Out[5]:
```
array([[<AxesSubplot:title={'center':'income'}>]], dtype=object)
```



In [10]:
```
df.columns
```

Out[10]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

In [8]:
```
#define the feature sets X
X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retir
X[0:5]
```

Out[8]:
```
array([[  2.,  13.,  44.,   1.,   9.,  64.,   4.,   5.,   0.,   0.,   2.],
       [  3.,  11.,  33.,   1.,   7., 136.,   5.,   5.,   0.,   0.,   6.],
       [  3.,  68.,  52.,   1.,  24., 116.,   1.,  29.,   0.,   1.,   2.],
       [  2.,  33.,  33.,   0.,  12.,  33.,   2.,   0.,   0.,   1.,   1.],
       [  2.,  23.,  30.,   1.,   9.,  30.,   1.,   2.,   0.,   0.,   4.]])
```

In [13]:
```
# get labels
y = df['custcat'].values
y[0:5]
```

Out[13]:
```
array([1, 4, 3, 1, 3], dtype=int64)
```

In [11]:
```
# normalize the data
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[11]:
```
array([[-0.02696767, -1.055125  ,  0.18450456,  1.0100505 , -0.25303431,
        -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
        -0.23065004],
       [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
         0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
         2.55666158],
       [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
         0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
        -0.23065004],
       [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
```

```
       -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
       -0.92747794],
      [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
       -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
        1.16300577]])
```

In [14]:
```
# split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (800, 11) (800,)
Test set: (200, 11) (200,)
```

In [15]:
```
# get KNN
from sklearn.neighbors import KNeighborsClassifier
```

In [16]:
```
# start with k = 4
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[16]:
```
KNeighborsClassifier(n_neighbors=4)
```

In [17]:
```
# make predictions on the test set
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[17]:
```
array([1, 1, 3, 2, 4], dtype=int64)
```

In [18]:
```
# compute accuracy with accuracy classification score (which is the jaccard score)
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

In [19]:
```
# try again with k = 6
k = 6
neigh6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = neigh6.predict(X_test)
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh6.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat6))
```

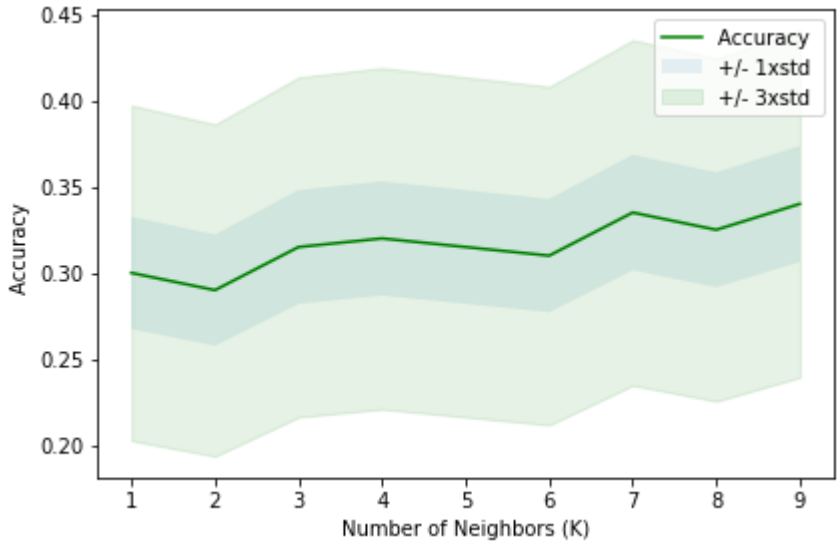
```
Train set Accuracy:  0.51625
Test set Accuracy:  0.31
```

In [20]:
```
# try other K values
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
```

```
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Out[20]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

In [21]:
```
# plot model accuracy for different number of neighbors
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [22]:
```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.34 with k= 9
```

Introduction to Decision Trees

Decision Tree:

- The basic intuition behind a decision tree is to map out all the possible decision paths that form the tree
- The tree is a set of categorical variables
- One node of the tree contains all of or most of one category of the data
- they are built by splitting the training set into distinct nodes

- each internal node corresponds to a test
- each branch corresponds tot he result of the test
- each leaf node assigns the classification

Decision Tree algorithm:

1. choose an attribute from the dataset
2. calculate the significance of attribute in splitting of data
    - finding the 'best' or 'most predictive' attribute:
        - choose the feature with the lowest impurity and entropy, i.e. the nodes all fall into a specific category of the target field. Minimize the impurity of nodes.
        - Entropy: measure of randomness or uncertainty. The lower the entropy, the less uniform the distribution, the purer the node.
            - entropy = 1 means the nodes are evenly split in categories.
                - calculate the entropy for all attributes and choose the one that has the smallest entropy & highest information gain
                - information gain is the info that can increase the level of certainty after splitting.
                    - IG = (entropy before split) - (weighted entropy after split)
3. split data based on value of the best attribute
4. go to step 1 for the rest of the attributes

Lab: Decision Trees

In [23]:
```
# import libraries
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
# download data
!wget -O drug200.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
```

```
--2021-08-12 16:18:57--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/drug200.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5827 (5.7K) [text/csv]
Saving to: 'drug200.csv'

     0K .....                                                 100% 4.93G=0s

2021-08-12 16:18:57 (4.93 GB/s) - 'drug200.csv' saved [5827/5827]
```

In [25]:
```
# put data into dataframe
my_data = pd.read_csv("drug200.csv", delimiter=",")
my_data[0:5]
```

Out[25]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

In [26]:
```
# get size of data
my_data.shape
```

Out[26]:
```
(200, 6)
```

In [27]:
```
# declare x as the feature data and y as response vector.
X = my_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

Out[27]:
```
array([[23, 'F', 'HIGH', 'HIGH', 25.355],
       [47, 'M', 'LOW', 'HIGH', 13.093],
       [47, 'M', 'LOW', 'HIGH', 10.114],
       [28, 'F', 'NORMAL', 'HIGH', 7.798],
       [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

In [28]:
```
# convert sex and BP to numerical values since decision trees from sklearn can't handle
from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])


le_BP = preprocessing.LabelEncoder()
le_BP.fit([ 'LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_BP.transform(X[:,2])


le_Chol = preprocessing.LabelEncoder()
le_Chol.fit([ 'NORMAL', 'HIGH'])
X[:,3] = le_Chol.transform(X[:,3])

X[0:5]
```

Out[28]:
```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.114],
       [28, 0, 2, 0, 7.798],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

In [29]:
```
# fill target variable
y = my_data["Drug"]
y[0:5]
```

Out[29]:
```
0    drugY
1    drugC
```

```
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

In [30]:
```
# split dataset into train and test sets
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, ra
```

In [31]:
```
# get shape of the training sets
print('Shape of X training set {}'.format(X_trainset.shape),'&',' Size of Y training se
```

```
Shape of X training set (140, 5) &  Size of Y training set (140,)
```

In [32]:
```
# get shape of test sets
print('Shape of X test set {}'.format(X_testset.shape),'&',' Size of Y test set {}'.for
```

```
Shape of X test set (60, 5) &  Size of Y test set (60,)
```

In [33]:
```
# create instance of decisiontreeclassifier
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # it shows the default parameters
# criterion = entropy allows us to see the information gain of each node
```

Out[33]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [34]:
```
# fit the model
drugTree.fit(X_trainset,y_trainset)
```

Out[34]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [35]:
```
# make predictions on the test set
predTree = drugTree.predict(X_testset)
```

In [36]:
```
# compare the predicted results to the actual values
print (predTree [0:5])
print (y_testset [0:5])
```

```
['drugY' 'drugX' 'drugX' 'drugX' 'drugX']
40     drugY
51     drugX
139    drugX
197    drugX
170    drugX
Name: Drug, dtype: object
```

In [37]:
```
# import metrics to check accuracy of model
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

```
DecisionTrees's Accuracy:  0.9833333333333333
```

Accuracy classification score computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.

In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

In [38]:
```
# ! pip install pydotplus
```

```
Collecting pydotplus
  Downloading pydotplus-2.0.2.tar.gz (278 kB)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\orgil\appdata\local\programs
\python\python39\lib\site-packages (from pydotplus) (2.4.7)
Using legacy 'setup.py install' for pydotplus, since package 'wheel' is not installed.
Installing collected packages: pydotplus
    Running setup.py install for pydotplus: started
    Running setup.py install for pydotplus: finished with status 'done'
Successfully installed pydotplus-2.0.2
```

In [40]:
```
# ! pip install graphviz
```

```
Collecting graphviz
  Downloading graphviz-0.17-py3-none-any.whl (18 kB)
Installing collected packages: graphviz
Successfully installed graphviz-0.17
```

In [41]:
```
# import graphing libraries
from  io import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [ ]:
```
# plot the tree
dot_data = StringIO()
filename = "drugtree.png"
featureNames = my_data.columns[0:5]
out=tree.export_graphviz(drugTree,feature_names=featureNames, out_file=dot_data, class_
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```

image

Introduction to Logistic Regression

Logistic Regression:

- a classification algorithm for categorical variables
- analogous to linear regression, but tries to predict a categorical variable rather than a numerical variable.

When to use logistic regression?

- if the data is binary
- if you need probabilistic results
- when you need a linear decision boundary
- if you need to understand the impact of a feature

Logistic Regression vs Linear Regression:

- linear regression cannot properly measure the probability of a case belonging to a class.

Training algorithm:

1. initialize parameters randomly
2. feed the cost function with training set and calculate the error
3. calculate the gradient of cost function
4. update weights with new values
5. repeat step 2 until cost is small enough
6. predict the new customer x

Find the best parameters for the model:

- minimize the cost function using gradient descent
- Gradient descent: a technique to use the derivative of a cost function to change the parameter values in order to minimize the cost

Lab: Logistic Regression

What is the difference between Linear and Logistic Regression?

While Linear Regression is suited for estimating continuous values (e.g. estimating house price), it is not the best tool for predicting the class of an observed data point. In order to estimate the class of a data point, we need some sort of guidance on what would be the **most probable class** for that data point. For this, we use **Logistic Regression**.

Recall linear regression:

As you know, **Linear regression** finds a function that relates a continuous dependent variable, **y**, to some predictors (independent variables x_1, x_2, etc.). For example, simple linear regression assumes a function of the form:

$$ y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots $$
and finds the values of parameters $\theta_0, \theta_1, \theta_2$, etc, where the term θ_0 is the "intercept". It can be generally shown as:

$$ $h_\theta(x)$ = \theta^TX $$

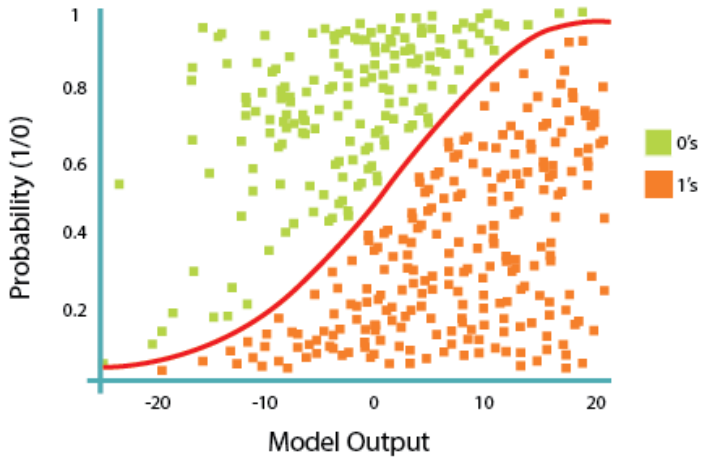
Logistic Regression is a variation of Linear Regression, useful when the observed dependent variable, **y**, is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables.

Logistic regression fits a special s-shaped curve by taking the linear regression function and transforming the numeric estimate into a probability with the following function, which is called the sigmoid function σ:

$$ $h\_\theta(x) = \sigma({\theta^TX}) = \frac {e^{(\theta\_0 + \theta\_1 x\_1 + \theta\_2 x\_2 +...)}}{1 + e^{(\theta\_0 + \theta\_1 x\_1 + \theta\_2 x\_2 +\cdots)}} $$
Or: $$ ProbabilityOfaClass\_1 = P(Y=1|X) = \sigma({\theta^TX}) = \frac{e^{\theta^TX}}{1+e^{\theta^TX}} $$

In this equation, ${\theta^TX}$ is the regression result (the sum of the variables weighted by the coefficients), exp is the exponential function and $\sigma(\theta^TX)$ is the sigmoid or logistic function, also called logistic curve. It is a common "S" shape (sigmoid curve).

So, briefly, Logistic Regression passes the input through the logistic/sigmoid but then treats the result as a probability:



The objective of the **Logistic Regression** algorithm, is to find the best parameters θ, for $$h\_\theta(x)$$ = $\sigma({\theta^TX})$, in such a way that the model best predicts the class of each case.

In [49]:
```
# import libraries
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

About the dataset

We will use a telecommunications dataset for predicting customer churn. This is a historical customer dataset where each row represents one customer. The data is relatively easy to

understand, and you may uncover insights you can use immediately. Typically it is less expensive to keep customers than acquire new ones, so the focus of this analysis is to predict the customers who will stay with the company.

This data set provides information to help you predict what behavior will help you to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

The dataset includes information about:

- Customers who left within the last month – the column is called Churn
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

Load the Telco churn data

Telco Churn is a hypothetical data file that concerns a telecommunications company's efforts to reduce turnover in its customer base. Each case corresponds to a separate customer and it records various demographic and service usage information. Before you can work with the data, you must use the URL to get the ChurnData.csv.

To download the data, we will use `!wget` to download it from IBM Object Storage.

In [50]:
```
#Click here and press Shift+Enter
!wget -O ChurnData.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
```

```
--2021-08-12 16:59:24--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/ChurnData.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 169.45.118.108
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|169.45.118.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35943 (35K) [text/csv]
Saving to: 'ChurnData.csv'

     0K .......... .......... .......... .....                 100% 1.73M=0.02s

2021-08-12 16:59:24 (1.73 MB/s) - 'ChurnData.csv' saved [35943/35943]
```

In [51]:
```
# store in dataframe
churn_df = pd.read_csv("ChurnData.csv")
churn_df.head()
```

Out[51]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	loglong	logtoll	lninc	custcat	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	1.0	1.0	0.0	1.482	3.033	4.913	4.0	1.0

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	loglong	logtoll	lninc	custcat	churn
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	0.0	0.0	0.0	2.246	3.240	3.497	1.0	1.0
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	0.0	1.0	0.0	1.841	3.240	3.401	3.0	0.0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	1.0	1.0	1.0	1.800	3.807	4.331	4.0	0.0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	1.0	1.0	0.0	1.960	3.091	4.382	3.0	0.0

5 rows × 28 columns

In [52]:
```
# select desired features for the model
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',   '

# update datatype to integer (a requirement by the algorithm)
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

Out[52]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

In [53]:
```
churn_df.shape
```

Out[53]:
```
(200, 10)
```

In [54]:
```
# define x
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']
X[0:5]
```

Out[54]:
```
array([[ 11.,  33.,   7., 136.,   5.,   5.,   0.],
       [ 33.,  33.,  12.,  33.,   2.,   0.,   0.],
       [ 23.,  30.,   9.,  30.,   1.,   2.,   0.],
       [ 38.,  35.,   5.,  76.,   2.,  10.,   1.],
       [  7.,  35.,  14.,  80.,   2.,  15.,   0.]])
```

In [55]:
```
# define y
y = np.asarray(churn_df['churn'])
y [0:5]
```

Out[55]:
```
array([1, 1, 0, 0, 0])
```

In [56]:
```
# normalize dataset
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[56]:
```
array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423 ,  1.6961288 ,
        -0.58477841, -0.85972695],
       [-0.11604313, -0.62595491,  0.03454064, -0.32886061, -0.6433592 ,
        -1.14437497, -0.85972695],
       [-0.57928917, -0.85594447, -0.261522  , -0.35227817, -1.42318853,
        -0.92053635, -0.85972695],
       [ 0.11557989, -0.47262854, -0.65627219,  0.00679109, -0.6433592 ,
        -0.02518185,  1.16316   ],
       [-1.32048283, -0.47262854,  0.23191574,  0.03801451, -0.6433592 ,
         0.53441472, -0.85972695]])
```

In [57]:
```
# split dataset into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (160, 7) (160,)
Test set: (40, 7) (40,)
```

Let's build our model using **LogisticRegression** from the Scikit-learn package. This function implements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers. You can find extensive information about the pros and cons of these optimizers if you search it in the internet.

The version of Logistic Regression in Scikit-learn, support regularization. Regularization is a technique used to solve the overfitting problem of machine learning models. **C** parameter indicates **inverse of regularization strength** which must be a positive float. Smaller values specify stronger regularization. Now let's fit our model with train set:

In [58]:
```
# fit the model with train set
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[58]:
```
LogisticRegression(C=0.01, solver='liblinear')
```

In [59]:
```
# make a prediction with test set
yhat = LR.predict(X_test)
yhat
```

Out[59]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [60]:
```
# use predict_proba to return estimates for all classes, ordered by the label of the cl
yhat_prob = LR.predict_proba(X_test)
yhat_prob
# col 1 = prob of class 0
# col 2 = prob of class 1
```

Out[60]:
```
array([[0.54132919, 0.45867081],
       [0.60593357, 0.39406643],
       [0.56277713, 0.43722287],
       [0.63432489, 0.36567511],
```

```
       [0.56431839, 0.43568161],
       [0.55386646, 0.44613354],
       [0.52237207, 0.47762793],
       [0.60514349, 0.39485651],
       [0.41069572, 0.58930428],
       [0.6333873 , 0.3666127 ],
       [0.58068791, 0.41931209],
       [0.62768628, 0.37231372],
       [0.47559883, 0.52440117],
       [0.4267593 , 0.5732407 ],
       [0.66172417, 0.33827583],
       [0.55092315, 0.44907685],
       [0.51749946, 0.48250054],
       [0.485743  , 0.514257  ],
       [0.49011451, 0.50988549],
       [0.52423349, 0.47576651],
       [0.61619519, 0.38380481],
       [0.52696302, 0.47303698],
       [0.63957168, 0.36042832],
       [0.52205164, 0.47794836],
       [0.50572852, 0.49427148],
       [0.70706202, 0.29293798],
       [0.55266286, 0.44733714],
       [0.52271594, 0.47728406],
       [0.51638863, 0.48361137],
       [0.71331391, 0.28668609],
       [0.67862111, 0.32137889],
       [0.50896403, 0.49103597],
       [0.42348082, 0.57651918],
       [0.71495838, 0.28504162],
       [0.59711064, 0.40288936],
       [0.63808839, 0.36191161],
       [0.39957895, 0.60042105],
       [0.52127638, 0.47872362],
       [0.65975464, 0.34024536],
       [0.5114172 , 0.4885828 ]])
```

In [61]:

```
# use jaccard index for accuracy evaluation
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=0)
# closer to 1 = higher accuracy
```

Out[61]: 0.7058823529411765

In [62]:

```
# use confusion matrix for accuracy evaluation
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
```

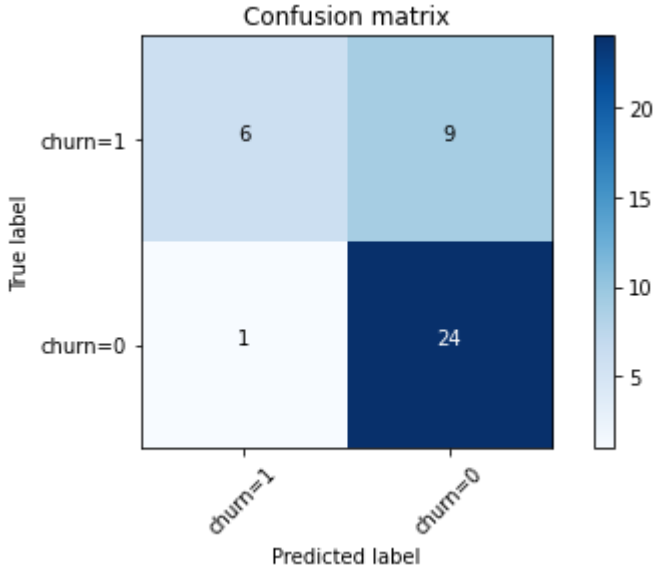
```
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[ 6  9]
 [ 1 24]]
```

In [63]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  titl
```

```
Confusion matrix, without normalization
[[ 6  9]
 [ 1 24]]
```



Look at first row. The first row is for customers whose actual churn value in the test set is 1. As you can calculate, out of 40 customers, the churn value of 15 of them is 1. Out of these 15 cases, the classifier correctly predicted 6 of them as 1, and 9 of them as 0.

This means, for 6 customers, the actual churn value was 1 in test set and classifier also correctly predicted those as 1. However, while the actual label of 9 customers was 1, the classifier predicted those as 0, which is not very good. We can consider it as the error of the model for first row.

What about the customers with churn value 0? Lets look at the second row. It looks like there were 25 customers whom their churn value were 0.

The classifier correctly predicted 24 of them as 0, and one of them wrongly as 1. So, it has done a good job in predicting the customers with churn value 0. A good thing about the confusion matrix is that it shows the model's ability to correctly predict or separate the classes. In a specific case of the binary classifier, such as this example, we can interpret these numbers as the count of true positives, false positives, true negatives, and false negatives.

In [64]:
```
# compute precision, recall, and f1 score
print (classification_report(y_test, yhat))
# perfect accuracy f1 score = 1
```

```
              precision    recall  f1-score   support

           0       0.73      0.96      0.83        25
           1       0.86      0.40      0.55        15

    accuracy                           0.75        40
   macro avg       0.79      0.68      0.69        40
weighted avg       0.78      0.75      0.72        40
```

Based on the count of each section, we can calculate precision and recall of each label:

- **Precision** is a measure of the accuracy provided that a class label has been predicted. It is defined by: precision = TP / (TP + FP)
- **Recall** is the true positive rate. It is defined as: Recall = TP / (TP + FN)

So, we can calculate the precision and recall of each class.

F1 score: Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifer has a good value for both recall and precision.

Finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.72 in our case.

log loss

Now, let's try **log loss** for evaluation. In logistic regression, the output can be the probability of customer churn is yes (or equals to 1). This probability is a value between 0 and 1. Log loss(Logarithmic loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1.

In [65]:
```
# import library
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
# closer to 0 means higher accuracy
```

Out[65]:
```
0.6017092478101185
```

In [66]:
```
# Try to build Logistic Regression model again for the same dataset, but this time, use
LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
LogLoss: : 0.61
```

Support Vector Machine (SVM)

SVM:

- a supervised algorithm that classifies cases by finding a separator

1. mapping data to a high dimensional feature space
2. finding a separator

- Mapping data to a higher dimensional space is called kernelling and can have different types
  - linear, polynomial, RBF, sigmoid

Using SVM to find the separator hyperplane:

- this is a line or plane that linearly separates the classes completely
- choose the hyperplane that has the largest margin possible, or the maximum distance between the support vectors

Pros and cons:

- accurate in high dimensional spaces
- memory efficient
- con: prone to overfitting
- con: no probability estimation
- con: better suited to small datasets

SVM applications:

- image recognition
- text category assignment
- detecting spam
- sentiment analysis
- gene expression classification
- regression, outlier detection, and clustering

Lab: SVM

In [67]:
```
# load libraries
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

In [68]:
```
# get data
!wget -O cell_samples.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
```

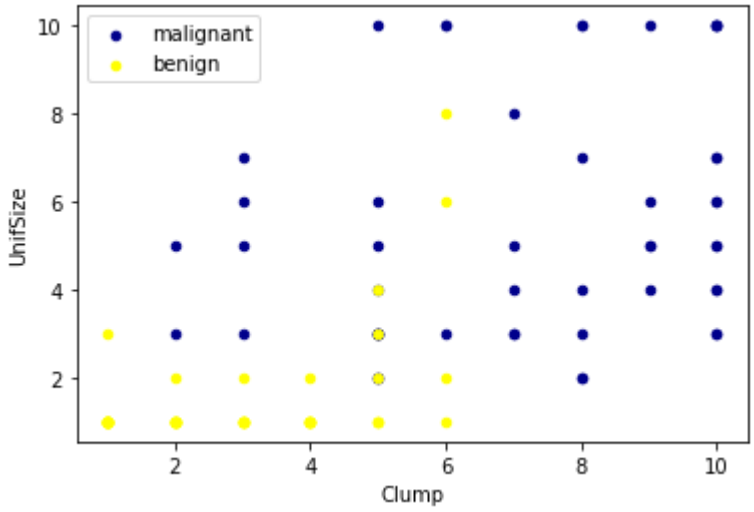
```
--2021-08-12 17:18:58--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/cell_samples.c
sv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 19975 (20K) [text/csv]
Saving to: 'cell_samples.csv'

     0K .......... .........                                  100%  511K=0.04s

2021-08-12 17:18:58 (511 KB/s) - 'cell_samples.csv' saved [19975/19975]
```

In [69]:
```
# load into a dataframe
cell_df = pd.read_csv("cell_samples.csv")
cell_df.head()
```

Out[69]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

In [70]:
```
# look at the distribution of classes based on clump thickness and uniformity of size
ax = cell_df[cell_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='UnifSize',
cell_df[cell_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize', colo
plt.show()
```



In [71]:
```
cell_df.dtypes
```

Out[71]:
```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc        object
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

In [72]:
```
# drop non-numerical rows
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes
```

Out[72]:
```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc         int32
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

In [73]:
```
# get features
feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'Bare
X = np.asarray(feature_df)
X[0:5]
```

Out[73]:
```
array([[ 5,  1,  1,  1,  2,  1,  3,  1,  1],
       [ 5,  4,  4,  5,  7, 10,  3,  2,  1],
       [ 3,  1,  1,  1,  2,  2,  3,  1,  1],
```

```
       [ 6,  8,  8,  1,  3,  4,  3,  7,  1],
       [ 4,  1,  1,  3,  2,  1,  3,  1,  1]], dtype=int64)
```

In [74]:
```
# change the class field's measurement level to reflect that it can only have 2 measure
# benign2 or malignant4
cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class'])
y [0:5]
```

Out[74]:
```
array([2, 2, 2, 2, 2])
```

In [75]:
```
# split the dataset
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (546, 9) (546,)
Test set: (137, 9) (137,)
```

The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as:

1. Linear
2. Polynomial
3. Radial basis function (RBF)
4. Sigmoid

Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy way of knowing which function performs best with any given dataset. We usually choose different functions in turn and compare the results. Let's just use the default, RBF (Radial Basis Function) for this lab.

In [76]:
```
# fit the data
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[76]:
```
SVC()
```

In [77]:
```
# predict new values
yhat = clf.predict(X_test)
yhat [0:5]
```

Out[77]:
```
array([2, 4, 2, 4, 2])
```

In [78]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

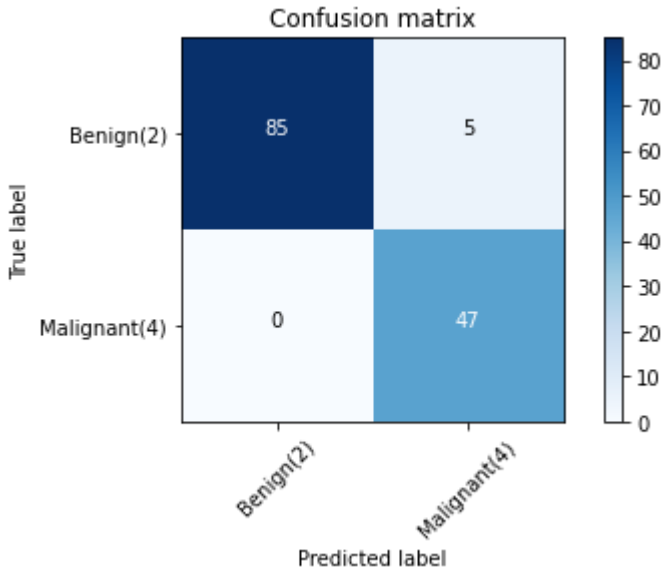
In [79]:
```
# create confusion matrix function
def plot_confusion_matrix(cm, classes,
```

```
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [80]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[2,4])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Benign(2)','Malignant(4)'],normalize= False
```

```
              precision    recall  f1-score   support

           2       1.00      0.94      0.97        90
           4       0.90      1.00      0.95        47

    accuracy                           0.96       137
   macro avg       0.95      0.97      0.96       137
weighted avg       0.97      0.96      0.96       137

Confusion matrix, without normalization
[[85  5]
 [ 0 47]]
```



In [81]:
```
# get f1score
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
# closer to 1 = higher accuracy
```

Out[81]: 0.9639038982104676

In [82]:
```
# get jaccard accuracy
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=2)
# closer to 1 = higher accuracy
```

Out[82]: 0.9444444444444444

In [85]:
```
# rebuild model but with a linear kernel. how does the accuracy change?
clf2 = svm.SVC(kernel='linear')
clf2.fit(X_train, y_train)
yhat2 = clf2.predict(X_test)
print("Avg F1-score: %.4f" % f1_score(y_test, yhat2, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(y_test, yhat2,pos_label=2))
```

```
Avg F1-score: 0.9639
Jaccard score: 0.9444
```

Week 4: Clustering

Introduction to Clustering

Clustering:

- finding clusters in a dataset unsupervised
- cluster: a group of objects that are similar to other objects in that cluster, and dissimilar to datapoints in other clusters

Clustering vs Classification:

- classification is supervised and labelled
- clustering is unsupervised and unlabelled

Applications:

- identify buying patterns
- recommend similar books/movies
- fraud detection
- identify clusters of customers
- insurance risk of customers
- auto-categorize news
- characterize patient behavior
- cluster genetic markers

Clustering Algorithms:

- partition based clustering
  - relatively efficient
  - kmeans, kmedian, fuzzy cmeans
- hierarchical clustering
  - produces trees of clusters
  - agglomerative, divisive
- density based clustering
  - dbscan

Introduction to K-Means

K-Means algorithms:

- partitioning clustering
- k-means divides the data into non-overlapping subsets (clusters) without any cluster internal structure or labels
- examples within a cluster are very similar, and examples across different clusters are very different
- minimize intra cluster distances and maximize inter cluster distances

Determine similiarity or dissimilarity:

- the (euclidean) distance of samples from each other is used to find the dissimilarity

K-means steps:

1. initialize k = 3 centroids either randomly or choosing 3 points in the dataset
2. calculate the distance of each datapoint from each centroid
3. assign each point to the closest centroid
4. compute new centroids for each cluster so they have less error. in this case, use the mean of the datapoints in its cluster
5. complete steps 2+ again until you get the point that the centroids no longer move.

Choosing K:

- run clustering across different values of k and find to what extent we can minimize the error of clustering (how dense the clusters are)
- increasing k will always decrease the error, so the value of the metric as a function of k is plotted and the elbow point of the graph where the rate of decrease sharply shifts is the right K for clustering

Lab: K-Means

In [87]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
%matplotlib inline
```

In [88]:
```
# set a random seed
np.random.seed(0)
```

Let's create our own dataset for this lab!

First we need to set a random seed. Use **numpy's random.seed()** function, where the seed will be set to 0.

Next we will be making *random clusters* of points by using the **make_blobs** class. The **make_blobs** class can take in many inputs, but we will be using these specific ones.

Input

- **n_samples**: The total number of points equally divided among clusters.
  - Value will be: 5000
- **centers**: The number of centers to generate, or the fixed center locations.
  - Value will be: [[4, 4], [-2, -1], [2, -3],[1,1]]
- **cluster_std**: The standard deviation of the clusters.
  - Value will be: 0.9

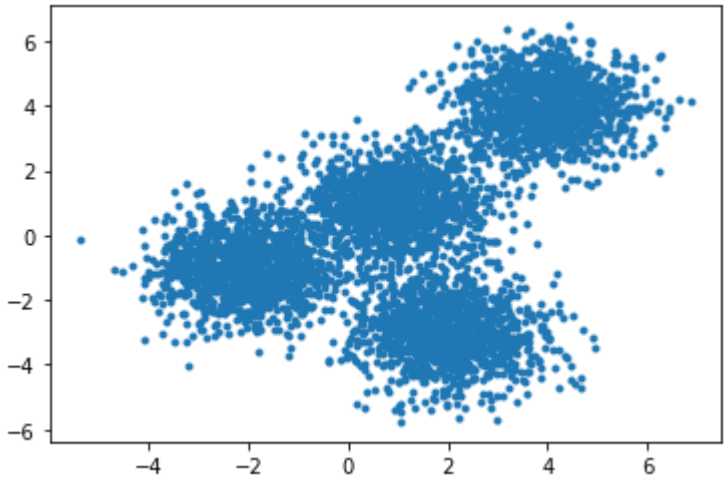
Output

- **X**: Array of shape [n_samples, n_features]. (Feature Matrix)
  - The generated samples.
- **y**: Array of shape [n_samples]. (Response Vector)
  - The integer labels for cluster membership of each sample.

In [89]:
```
# create random clusters
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_s
```

In [90]:
```
# display
plt.scatter(X[:, 0], X[:, 1], marker='.')
```

Out[90]:
```
<matplotlib.collections.PathCollection at 0x270d3a87a60>
```



The KMeans class has many parameters that can be used, but we will be using these three:

- **init**: Initialization method of the centroids.
  - Value will be: "k-means++"
  - k-means++: Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- **n_clusters**: The number of clusters to form as well as the number of centroids to generate.
  - Value will be: 4 (since we have 4 centers)
- **n_init**: Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
  - Value will be: 12

Initialize KMeans with these parameters, where the output parameter is called **k_means**.

In [91]:
```
# initialize kmeans
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```

In [92]:
```
# fit the model
k_means.fit(X)
```

Out[92]:
```
KMeans(n_clusters=4, n_init=12)
```

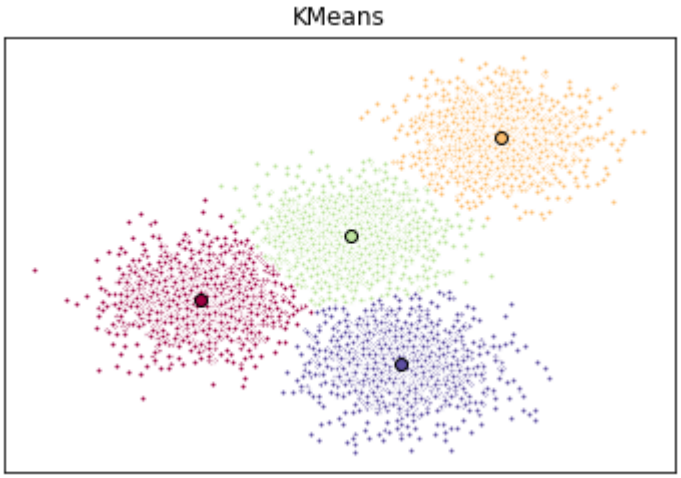
In [93]:
```
# get the labels for each point
k_means_labels = k_means.labels_
k_means_labels
```

Out[93]:
```
array([0, 3, 3, ..., 1, 0, 0])
```

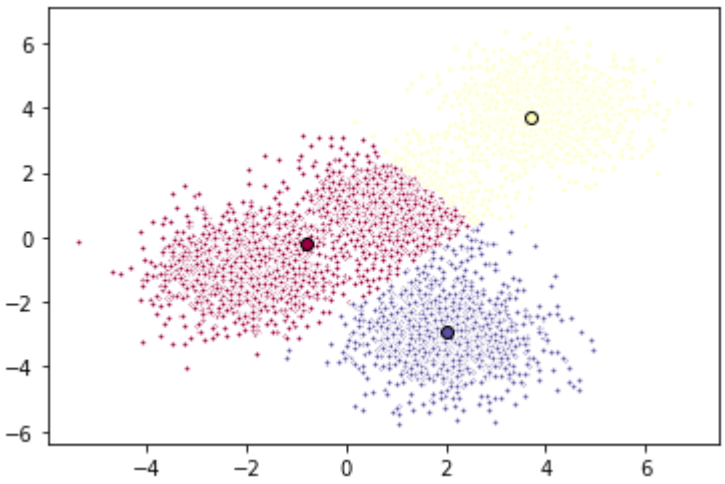
In [94]:
```
# get the coordinates of the cluster centers
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```

Out[94]:
```
array([[-2.04, -1.  ],
       [ 3.97,  3.99],
       [ 0.97,  0.98],
       [ 2.  , -3.02]])
```

In [95]:
```
# Initialize the plot with the specified dimensions.
fig = plt.figure(figsize=(6, 4))

# Colors uses a color map, which will produce an array of colors based on
# the number of labels there are. We use set(k_means_labels) to get the
# unique labels.
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels))))

# Create a plot
ax = fig.add_subplot(1, 1, 1)

# For loop that plots the data points and centroids.
# k will range from 0-3, which will match the possible clusters that each
# data point is in.
for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):

    # Create a list of all data points, where the data points that are
    # in the cluster (ex. cluster 0) are labeled as true, else they are
    # labeled as false.
    my_members = (k_means_labels == k)

    # Define the centroid, or cluster center.
    cluster_center = k_means_cluster_centers[k]

    # Plots the datapoints with color col.
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.')

    # Plots the centroids with specified color, but with a darker outline
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,  markeredge

# Title of the plot
ax.set_title('KMeans')

# Remove x-axis ticks
ax.set_xticks(())

# Remove y-axis ticks
ax.set_yticks(())

# Show the plot
plt.show()
```



In [96]:
```
# cluster the above dataset with 3 clusters
k_means3 = KMeans(init = "k-means++", n_clusters = 3, n_init = 12)
k_means3.fit(X)
fig = plt.figure(figsize=(6, 4))
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means3.labels_))))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(len(k_means3.cluster_centers_)), colors):
    my_members = (k_means3.labels_ == k)
    cluster_center = k_means3.cluster_centers_[k]
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.')
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,  markeredge
plt.show()
```



In [97]:
```
# let's try customer segmentation with k-means
# download dataset
!wget -O Cust_Segmentation.csv https://cf-courses-data.s3.us.cloud-object-storage.appdo
```

```
--2021-08-12 17:53:12--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/Cust_Segmentat
ion.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 169.45.118.108
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|169.45.118.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 33426 (33K) [text/csv]
```

```
Saving to: 'Cust_Segmentation.csv'

     0K .......... .......... .......... ..                   100% 1.47M=0.02s

2021-08-12 17:53:13 (1.47 MB/s) - 'Cust_Segmentation.csv' saved [33426/33426]
```

In [98]:
```
# load data
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

Out[98]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomel
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

In [99]:
```
# drop address
df = cust_df.drop('Address', axis=1)
df.head()
```

Out[99]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeR
0	1	41	2	6	19	0.124	1.073	0.0	6.3
1	2	47	1	26	100	4.582	8.218	0.0	12.8
2	3	33	2	10	57	6.111	5.802	1.0	20.9
3	4	29	2	4	19	0.681	0.516	0.0	6.3
4	5	47	1	31	253	9.308	8.908	0.0	7.2

In [100…
```
# normalize over the standard deviation
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Out[100…
```
array([[ 0.74,  0.31, -0.38, ..., -0.59, -0.52, -0.58],
       [ 1.49, -0.77,  2.57, ...,  1.51, -0.52,  0.39],
       [-0.25,  0.31,  0.21, ...,  0.8 ,  1.91,  1.6 ],
       ...,
       [-1.25,  2.47, -1.26, ...,  0.04,  1.91,  3.46],
       [-0.38, -0.77,  0.51, ..., -0.7 , -0.52, -1.08],
       [ 2.11, -0.77,  1.1 , ...,  0.16, -0.52, -0.23]])
```

In [101…
```
# apply k-means to the dataset and look at cluster labels
```

```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[0 2 0 0 1 2 0 2 0 2 2 0 0 0 0 0 0 2 0 0 0 0 2 2 2 0 0 2 0 2 0 0 0 0 0 0
 0 0 2 0 2 0 1 0 2 0 0 0 2 2 0 0 2 2 0 0 0 2 0 2 0 2 2 0 0 2 0 0 0 2 2 2 0
 0 0 0 0 2 0 2 2 1 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 2 0
 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 2 0
 0 0 0 0 0 0 2 0 2 2 0 2 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 2 0
 0 0 0 0 2 0 0 2 0 2 0 0 2 1 0 2 0 0 0 0 0 0 1 2 0 0 0 0 2 0 0 2 2 0 2 0 2
 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 2 0 0 0 0
 0 0 2 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 2 0 2 0 2 2 0 0 0 0 0
 0 0 0 2 2 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 2 0 2 2 0
 0 0 0 0 2 0 0 0 0 0 0 2 0 0 2 0 0 2 0 0 0 0 0 2 0 0 0 1 0 0 0 2 0 2 2 2 0
 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 0 2 0 0 2 0 0 0 0 2 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 1
 0 0 0 0 0 0 2 0 0 0 1 0 0 0 0 2 0 1 0 0 0 0 2 0 2 2 2 0 0 2 2 0 0 0 0 0 0
 0 2 0 0 0 0 2 0 0 0 2 0 2 0 0 0 2 0 0 0 0 2 2 0 0 0 0 2 0 0 0 0 2 0 0 0 0
 0 2 2 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 2 0 0 0 0 2 0 0 2 0 0 1 0 1 0
 0 1 0 0 0 0 0 0 0 0 0 2 0 2 0 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 2
 0 0 0 0 0 0 2 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2
 2 0 0 2 0 2 0 0 2 0 2 0 0 1 0 2 0 2 0 0 0 0 0 2 2 0 0 0 0 2 0 0 0 2 2 0 0
 2 0 0 0 2 0 1 0 0 2 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0
 0 0 2 0 0 2 0 2 0 2 2 0 0 0 2 0 2 0 0 0 0 0 2 0 0 0 0 2 2 0 0 2 2 0 0 0 0
 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 0 2 0 2 2 0 0 2 0 0 0 0 0 2 2
 0 0 0 0 0 0 0 2 0 0 0 0 0 0 1 2 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2]
```

In [102…

```
# assign labels to each row of the dataframe
df["Clus_km"] = labels
df.head(5)
```

Out[102…

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	0
1	2	47	1	26	100	4.582	8.218	0.0	12.8	2
2	3	33	2	10	57	6.111	5.802	1.0	20.9	0
3	4	29	2	4	19	0.681	0.516	0.0	6.3	0
4	5	47	1	31	253	9.308	8.908	0.0	7.2	1

In [103…

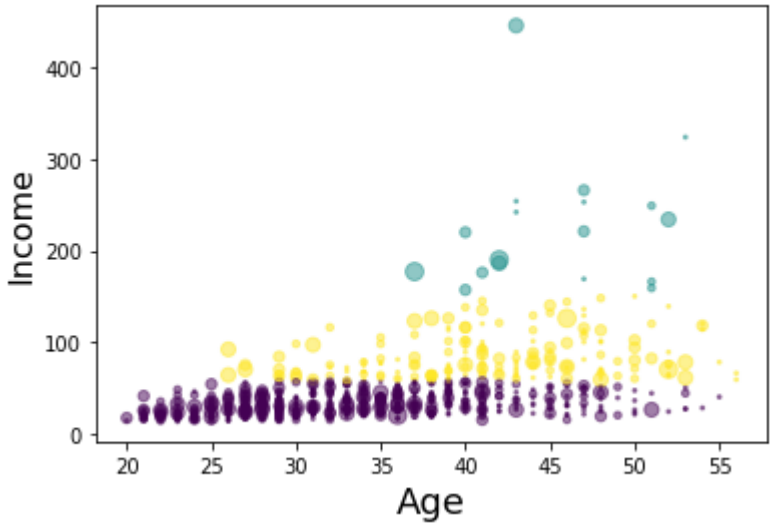
```
# check centroid values
df.groupby('Clus_km').mean()
```

Out[103…

Clus_km	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeR
0	432.006154	32.967692	1.613846	6.389231	31.204615	1.032711	2.108345	0.284658	10.095385
1	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714	7.322222
2	403.780220	41.368132	1.961538	15.252747	84.076923	3.114412	5.770352	0.172414	10.725824

In [104…
```
# check the distribution of customers based on age and income
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)

plt.show()
```

```
<ipython-input-104-18b5e2ff7a02>:3: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by itself. Doing this will
not modify any behavior and is safe. If you specifically wanted the numpy scalar type, u
se `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
```

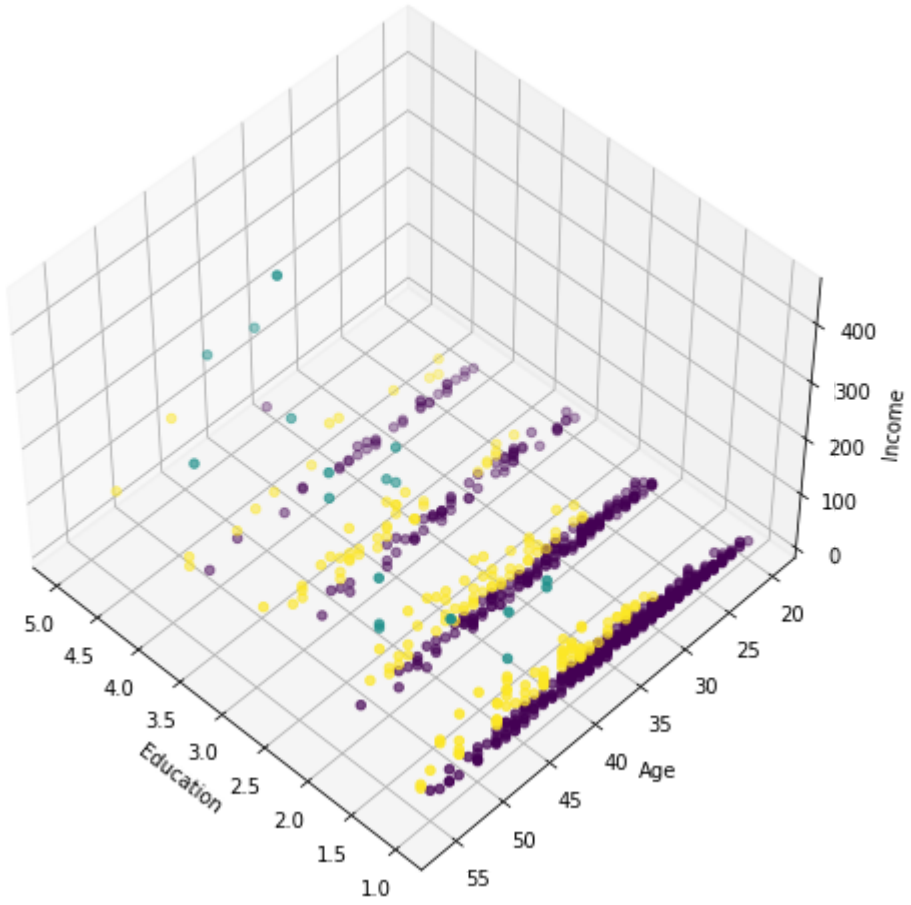


In [107…
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
# plt.ylabel('Age', fontsize=18)
# plt.xlabel('Income', fontsize=16)
# plt.zlabel('Education', fontsize=16)
ax.set_xlabel('Education')
ax.set_ylabel('Age')
ax.set_zlabel('Income')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

```
<ipython-input-107-13cd4ed72673>:4: MatplotlibDeprecationWarning: Axes3D(fig) adding its
elf to the figure is deprecated since 3.4. Pass the keyword argument auto_add_to_figure=
False and use fig.add_axes(ax) to suppress this warning. The default value of auto_add_t
o_figure will change to False in mpl3.5 and True values will no longer work in 3.6.  Thi
s is consistent with other Axes classes.
  ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
<ipython-input-107-13cd4ed72673>:14: DeprecationWarning: `np.float` is a deprecated alia
s for the builtin `float`. To silence this warning, use `float` by itself. Doing this wi
ll not modify any behavior and is safe. If you specifically wanted the numpy scalar typ
e, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
```

```
se/1.20.0-notes.html#deprecations
  ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

Out[107… `<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x270d3ecb310>`



Introduction to Hierarchical Clustering

Divisive Hierarchical Clustering:

- top down approach
- start with all observations in a large cluster and break it down into smaller pieces

Agglomerative Hierarchical Clustering:

- Bottom up approach
- each observation starts in its own cluster and pairs of clusters are merged together as they move up the hierarchy
- more popular approach

Agglomerative Algorithm:

1. create n clusters, one for each data point
2. compute the proximity matrix
3. repeat
    - merge the two closest clusters
    - update the proximity matrix
4. until only a single cluster remains

Criteria for distance between clusters:

- single linkage clustering: min distance between clusters
- complete linkage clustering: max distance between clusters
- average linking clustering: average distance between clusters
- centroid linkage clustering: distance between cluster centroids

Advantages:

- doesn't require number of clusters to be specified
- easy to implement
- produces dendrogram

Disadvantages:

- can never undo any previous steps of algorithm
- long runtimes
- sometimes difficult to identify number of clusters with dendrogram

Hierarchical Clustering vs K means:

- k means:
  - more efficient
  - requires the number of clusters to be specified
  - gives only one partitioning of the data based on number of clusters
  - potentially returns different clusters each time it is run due ot random initialization of clusters
- hierarchical:
  - slower
  - does not require the number of clusters
  - gives more than one partition depending on the resolution
  - always generates the same clusters

Lab: Agglomerative Clustering

Generating Random Data

We will be generating a set of data using the **make_blobs** class.

Input these parameters into make_blobs:

- **n_samples**: The total number of points equally divided among clusters.
  - Choose a number from 10-1500
- **centers**: The number of centers to generate, or the fixed center locations.
  - Choose arrays of x,y coordinates for generating the centers. Have 1-10 centers (ex. centers=[[1,1], [2,5]])
- **cluster_std**: The standard deviation of the clusters. The larger the number, the further apart the clusters

- Choose a number between 0.5-1.5

Save the result to **X1** and **y1**.

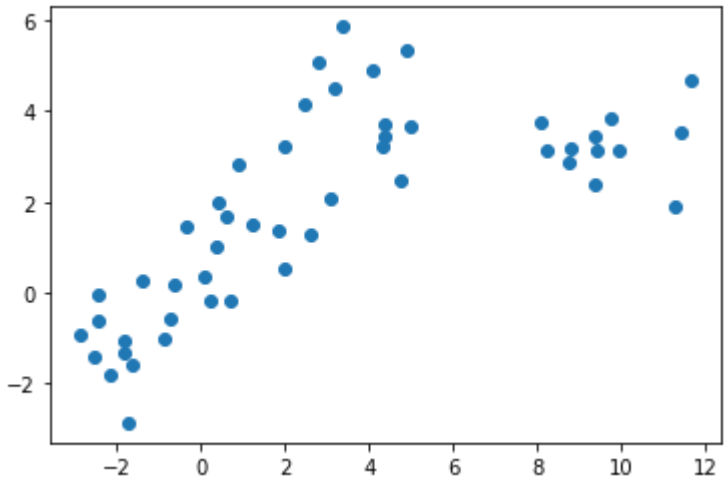
In [109…
```
import numpy as np
import pandas as pd
from scipy import ndimage
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
from matplotlib import pyplot as plt
from sklearn import manifold, datasets
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
%matplotlib inline
```

In [126…
```
import warnings
warnings.filterwarnings('ignore')
```

In [110…
```
# generate random data
X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster_st
```

In [111…
```
# plot the data
plt.scatter(X1[:, 0], X1[:, 1], marker='o')
```

Out[111… `<matplotlib.collections.PathCollection at 0x270d3989f40>`



The **Agglomerative Clustering** class will require two inputs:

- **n_clusters**: The number of clusters to form as well as the number of centroids to generate.
  - Value will be: 4
- **linkage**: Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.
  - Value will be: 'complete'
  - **Note**: It is recommended you try everything with 'average' as well

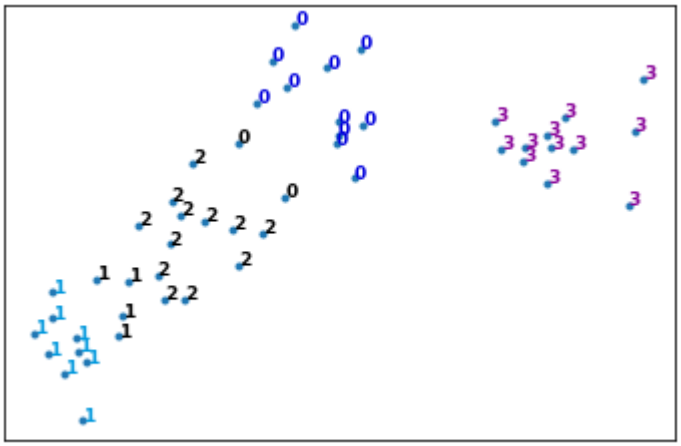
Save the result to a variable called **agglom** .

In [112…
```
agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
```

In [113…
```
# fit model
agglom.fit(X1,y1)
```

Out[113… AgglomerativeClustering(linkage='average', n_clusters=4)

In [114…
```
# Create a figure of size 6 inches by 4 inches.
plt.figure(figsize=(6,4))

# These two lines of code are used to scale the data points down,
# Or else the data points will be scattered very far apart.

# Create a minimum and maximum range of X1.
x_min, x_max = np.min(X1, axis=0), np.max(X1, axis=0)

# Get the average distance for X1.
X1 = (X1 - x_min) / (x_max - x_min)

# This loop displays all of the datapoints.
for i in range(X1.shape[0]):
    # Replace the data points with their respective cluster value
    # (ex. 0) and is color coded with a colormap (plt.cm.spectral)
    plt.text(X1[i, 0], X1[i, 1], str(y1[i]),
             color=plt.cm.nipy_spectral(agglom.labels_[i] / 10.),
             fontdict={'weight': 'bold', 'size': 9})

# Remove the x ticks, y ticks, x and y axis
plt.xticks([])
plt.yticks([])
#plt.axis('off')



# Display the plot of the original data before clustering
plt.scatter(X1[:, 0], X1[:, 1], marker='.')
# Display the plot
plt.show()
```



Dendrogram Associated for the Agglomerative Hierarchical Clustering

Remember that a **distance matrix** contains the **distance from each point to every other point of a dataset** .

Use the function **distance_matrix,** which requires **two inputs**. Use the Feature Matrix, **X1** as both inputs and save the distance matrix to a variable called **dist_matrix**

Remember that the distance values are symmetric, with a diagonal of 0's. This is one way of making sure your matrix is correct.
(print out dist_matrix to make sure it's correct)

In [115…
```
dist_matrix = distance_matrix(X1,X1)
print(dist_matrix)
```

```
[[0.   0.08 0.75 ... 0.38 0.75 0.35]
 [0.08 0.   0.82 ... 0.35 0.82 0.42]
 [0.75 0.82 0.   ... 0.9  0.09 0.4 ]
 ...
 [0.38 0.35 0.9  ... 0.   0.86 0.56]
 [0.75 0.82 0.09 ... 0.86 0.   0.41]
 [0.35 0.42 0.4  ... 0.56 0.41 0.  ]]
```

Using the **linkage** class from hierarchy, pass in the parameters:

- The distance matrix
- 'complete' for complete linkage

Save the result to a variable called **Z** .

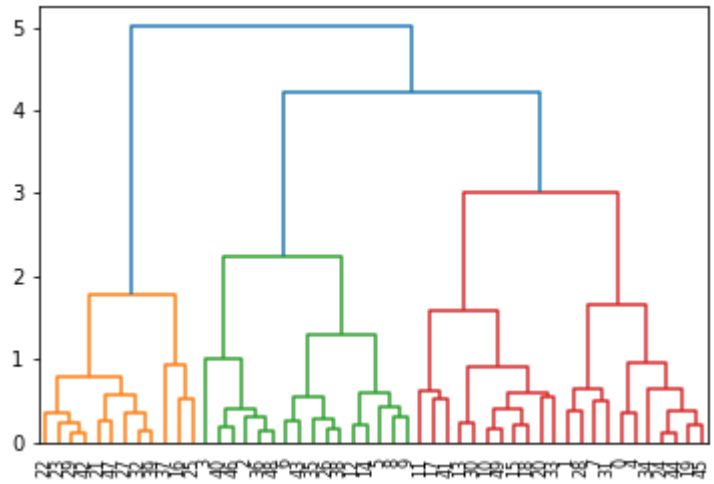
In [127…
```
Z = hierarchy.linkage(dist_matrix, 'complete')
```

A Hierarchical clustering is typically visualized as a dendrogram as shown in the following cell. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering.

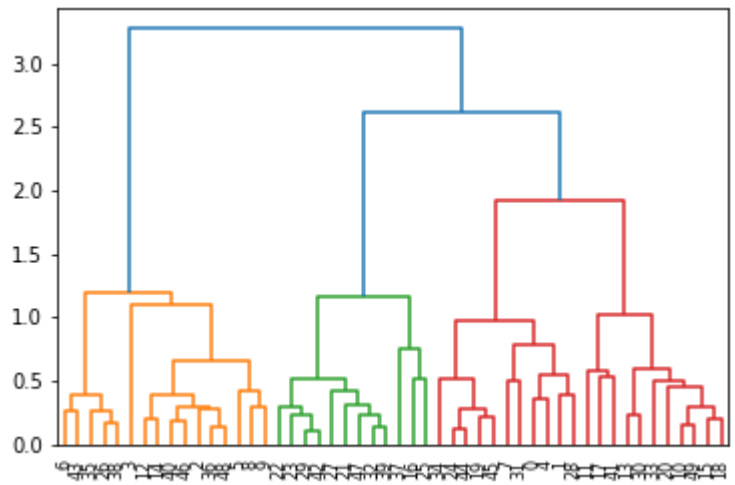
Next, we will save the dendrogram to a variable called **dendro**. In doing this, the dendrogram will also be displayed. Using the **dendrogram** class from hierarchy, pass in the parameter:

- Z

In [117…
```
dendro = hierarchy.dendrogram(Z)
```



In [128…
```
# use average linkage to see how the dendrogram changes
Z = hierarchy.linkage(dist_matrix, 'average')
dendro = hierarchy.dendrogram(Z)
```



In [129…
```
# cluster on a dataset
# download data
!wget -O cars_clus.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
```

```
--2021-08-12 18:15:48--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/cars_clus.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 169.45.118.108
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|169.45.118.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17774 (17K) [text/csv]
Saving to: 'cars_clus.csv'
```

```
    0K .......... .......                                   100% 2.88M=0.006s

2021-08-12 18:15:49 (2.88 MB/s) - 'cars_clus.csv' saved [17774/17774]
```

In [120…

```
# read and store data
filename = 'cars_clus.csv'

#Read csv
pdf = pd.read_csv(filename)
print ("Shape of dataset: ", pdf.shape)

pdf.head(5)
```

```
Shape of dataset:  (159, 16)
```

Out[120…

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition
0	Acura	Integra	16.919	16.360	0.000	21.500	1.800	140.000	101.200	67.300	172.400	2.639	13.200	28.000	2.828	0.0
1	Acura	TL	39.384	19.875	0.000	28.400	3.200	225.000	108.100	70.300	192.900	3.517	17.200	25.000	3.673	0.0
2	Acura	CL	14.114	18.225	0.000	$null$	3.200	225.000	106.900	70.600	192.000	3.470	17.200	26.000	2.647	0.0
3	Acura	RL	8.588	29.725	0.000	42.000	3.500	210.000	114.600	71.400	196.600	3.850	18.000	22.000	2.150	0.0
4	Audi	A4	20.397	22.255	0.000	23.990	1.800	150.000	102.600	68.200	178.000	2.998	16.400	27.000	3.015	0.0

In [121…

```
# clean data by dropping all rows with null values
print ("Shape of dataset before cleaning: ", pdf.size)
pdf[[ 'sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']] = pdf[['sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']].apply(pd.to_numeric, errors='coerce')
pdf = pdf.dropna()
pdf = pdf.reset_index(drop=True)
print ("Shape of dataset after cleaning: ", pdf.size)
pdf.head(5)
```

```
Shape of dataset before cleaning:  2544
Shape of dataset after cleaning:  1872
```

Out[121…

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition
0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0	2.828	0.0
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0	3.673	0.0
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0	2.150	0.0
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0	3.015	0.0
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561	18.5	22.0	2.933	0.0

In [122…

```
# select features
featureset = pdf[['engine_s',  'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', '
```

In [123…

```
# normalize the feature set
from sklearn.preprocessing import MinMaxScaler
```

```
x = featureset.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

Out[123…
```
array([[0.11, 0.22, 0.19, 0.28, 0.31, 0.23, 0.13, 0.43],
       [0.31, 0.43, 0.34, 0.46, 0.58, 0.5 , 0.32, 0.33],
       [0.36, 0.39, 0.48, 0.53, 0.63, 0.61, 0.35, 0.23],
       [0.11, 0.24, 0.22, 0.34, 0.38, 0.34, 0.28, 0.4 ],
       [0.26, 0.37, 0.35, 0.81, 0.57, 0.52, 0.38, 0.23]])
```

In [130…
```
# cluster the dataset
# first calculate distance matrix
import scipy
leng = feature_mtx.shape[0]
D = scipy.zeros([leng,leng])
for i in range(leng):
    for j in range(leng):
        D[i,j] = scipy.spatial.distance.euclidean(feature_mtx[i], feature_mtx[j])
D
```

Out[130…
```
array([[0.  , 0.58, 0.75, ..., 0.29, 0.25, 0.19],
       [0.58, 0.  , 0.23, ..., 0.36, 0.66, 0.62],
       [0.75, 0.23, 0.  , ..., 0.52, 0.82, 0.78],
       ...,
       [0.29, 0.36, 0.52, ..., 0.  , 0.42, 0.36],
       [0.25, 0.66, 0.82, ..., 0.42, 0.  , 0.15],
       [0.19, 0.62, 0.78, ..., 0.36, 0.15, 0.  ]])
```

In [131…
```
# cluster using complete distance
import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

In [132…
```
# use a cutting line
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

Out[132…
```
array([ 1,  5,  5,  6,  5,  4,  6,  5,  5,  5,  5,  5,  4,  4,  5,  1,  6,
        5,  5,  5,  4,  2, 11,  6,  6,  5,  6,  5,  1,  6,  6, 10,  9,  8,
        9,  3,  5,  1,  7,  6,  5,  3,  5,  3,  8,  7,  9,  2,  6,  6,  5,
        4,  2,  1,  6,  5,  2,  7,  5,  5,  5,  4,  4,  3,  2,  6,  6,  5,
        7,  4,  7,  6,  6,  5,  3,  5,  5,  6,  5,  4,  4,  1,  6,  5,  5,
        5,  6,  4,  5,  4,  1,  6,  5,  6,  6,  5,  5,  5,  7,  7,  7,  2,
        2,  1,  2,  6,  5,  1,  1,  1,  7,  8,  1,  1,  6,  1,  1],
      dtype=int32)
```

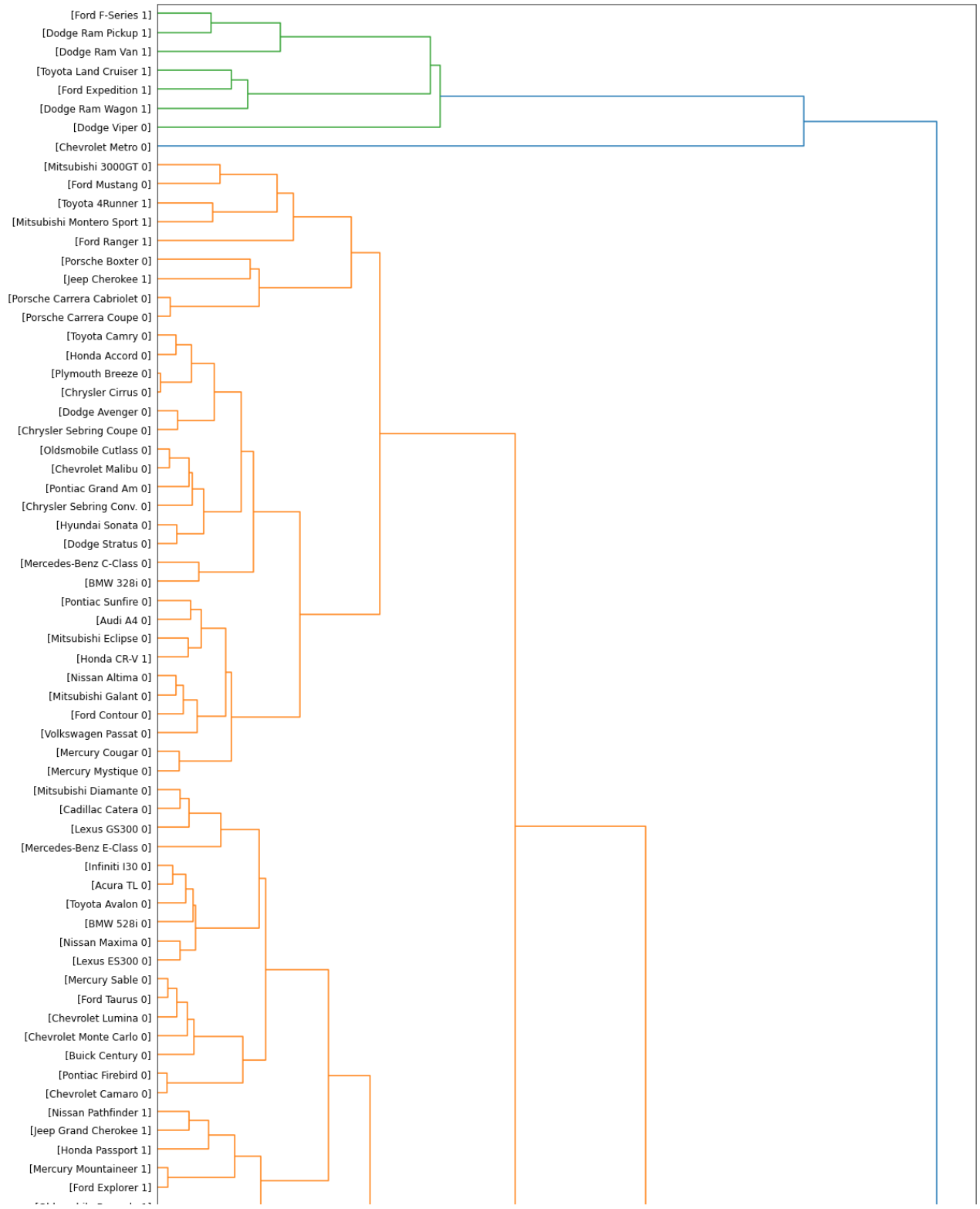
In [133…
```
# determine number of clusters
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

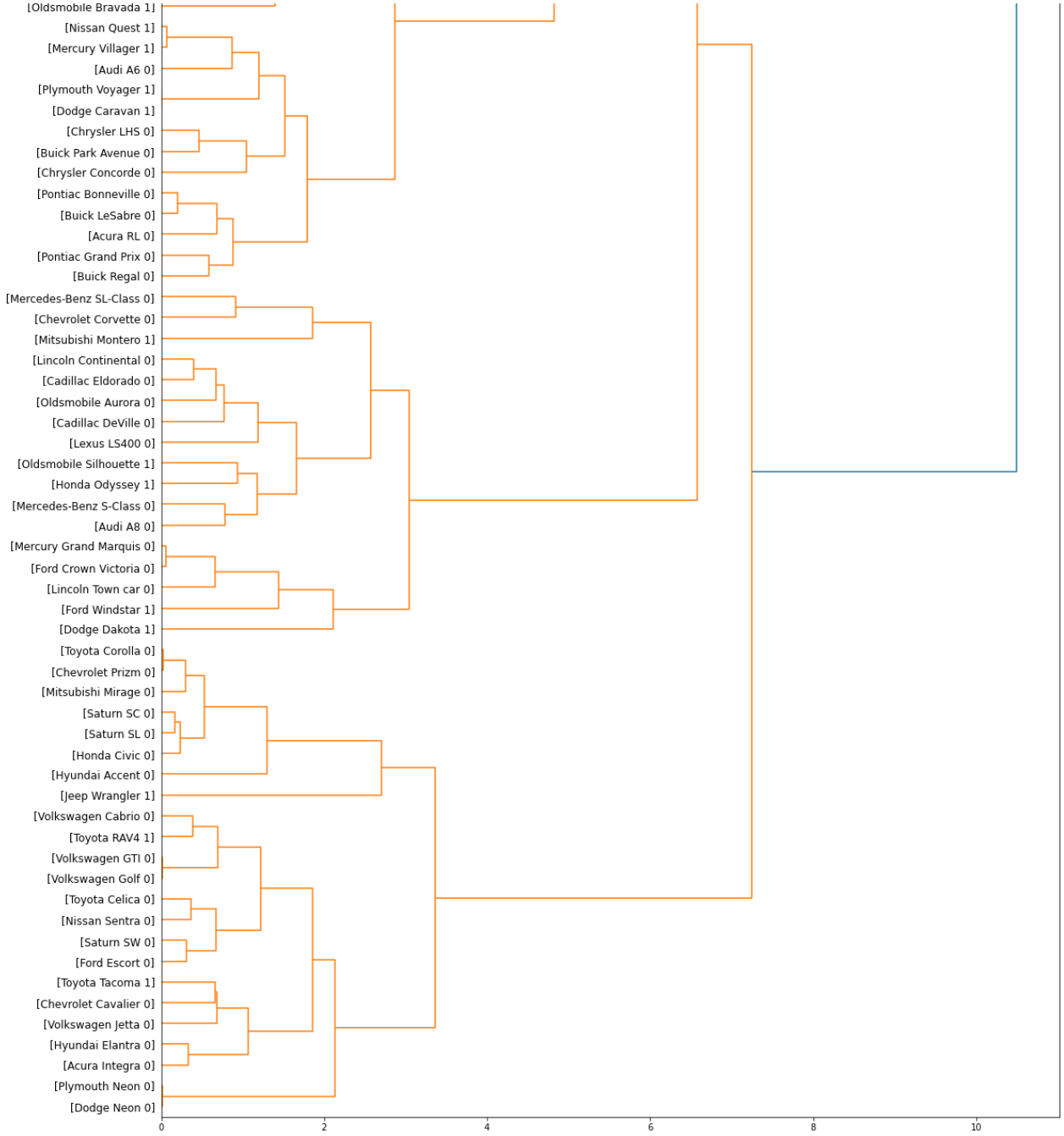
Out[133…
```
array([1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 2, 3, 1, 3, 3, 3, 3, 2, 1,
       5, 3, 3, 3, 3, 3, 1, 3, 3, 4, 4, 4, 4, 2, 3, 1, 3, 3, 3, 2, 3, 2,
```

```
       4, 3, 4, 1, 3, 3, 3, 2, 1, 1, 3, 3, 1, 3, 3, 3, 3, 2, 2, 2, 1, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 2, 1, 3, 3, 3, 3, 3, 2,
       3, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1,
       3, 4, 1, 1, 3, 1, 1], dtype=int32)
```

In [134…

```
# plot the dendrogram
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (pdf['manufact'][id], pdf['model'][id], int(float(pdf['type']

dendro = hierarchy.dendrogram(Z,  leaf_label_func=llf, leaf_rotation=0, leaf_font_size
```





In [135…

```
# redo, but this time using scikit learn
from sklearn.metrics.pairwise import euclidean_distances
dist_matrix = euclidean_distances(feature_mtx,feature_mtx)
print(dist_matrix)
```

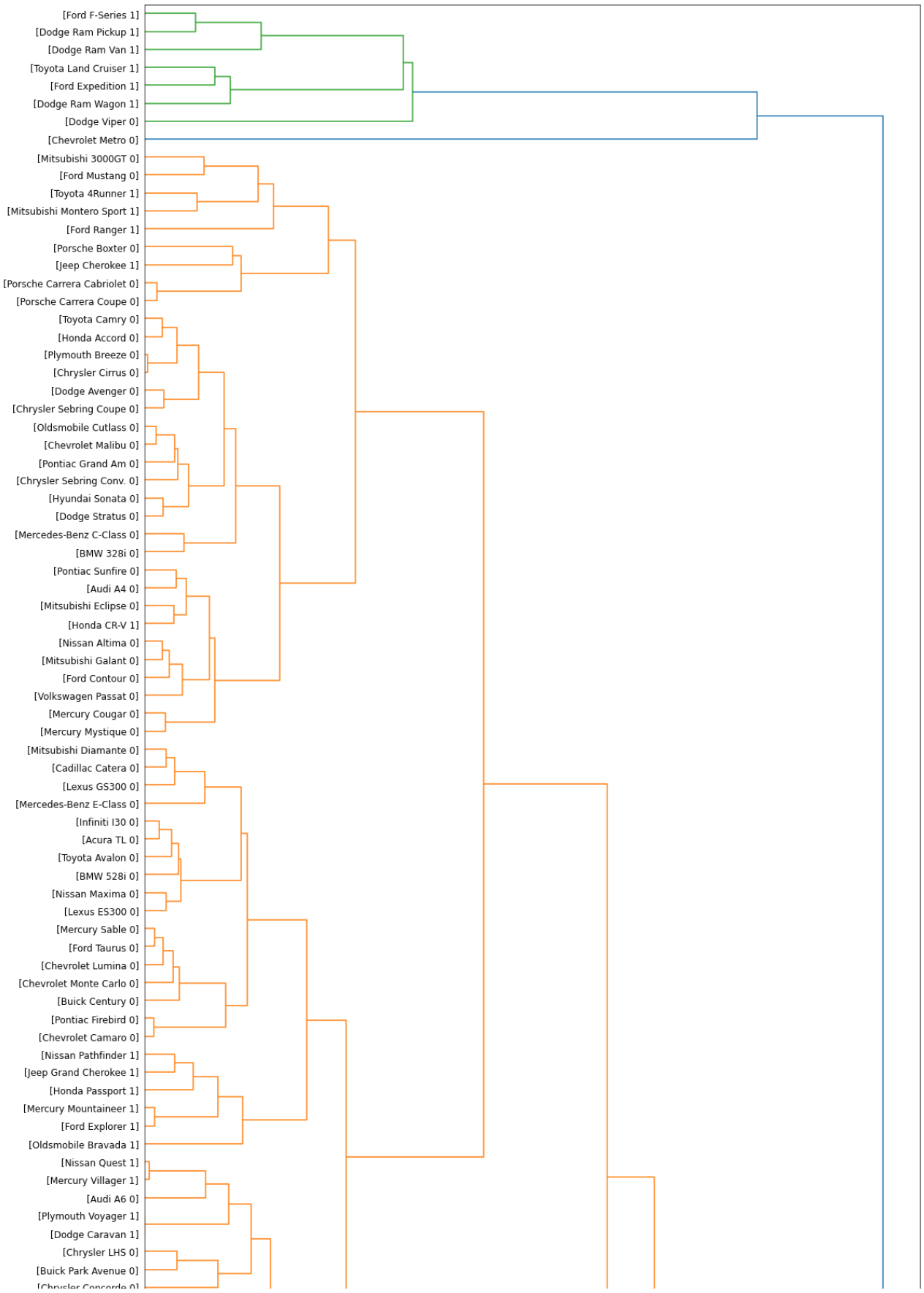
```
[[0.   0.58 0.75 ... 0.29 0.25 0.19]
 [0.58 0.   0.23 ... 0.36 0.66 0.62]
 [0.75 0.23 0.   ... 0.52 0.82 0.78]
 ...
 [0.29 0.36 0.52 ... 0.   0.42 0.36]
 [0.25 0.66 0.82 ... 0.42 0.   0.15]
 [0.19 0.62 0.78 ... 0.36 0.15 0.  ]]
```

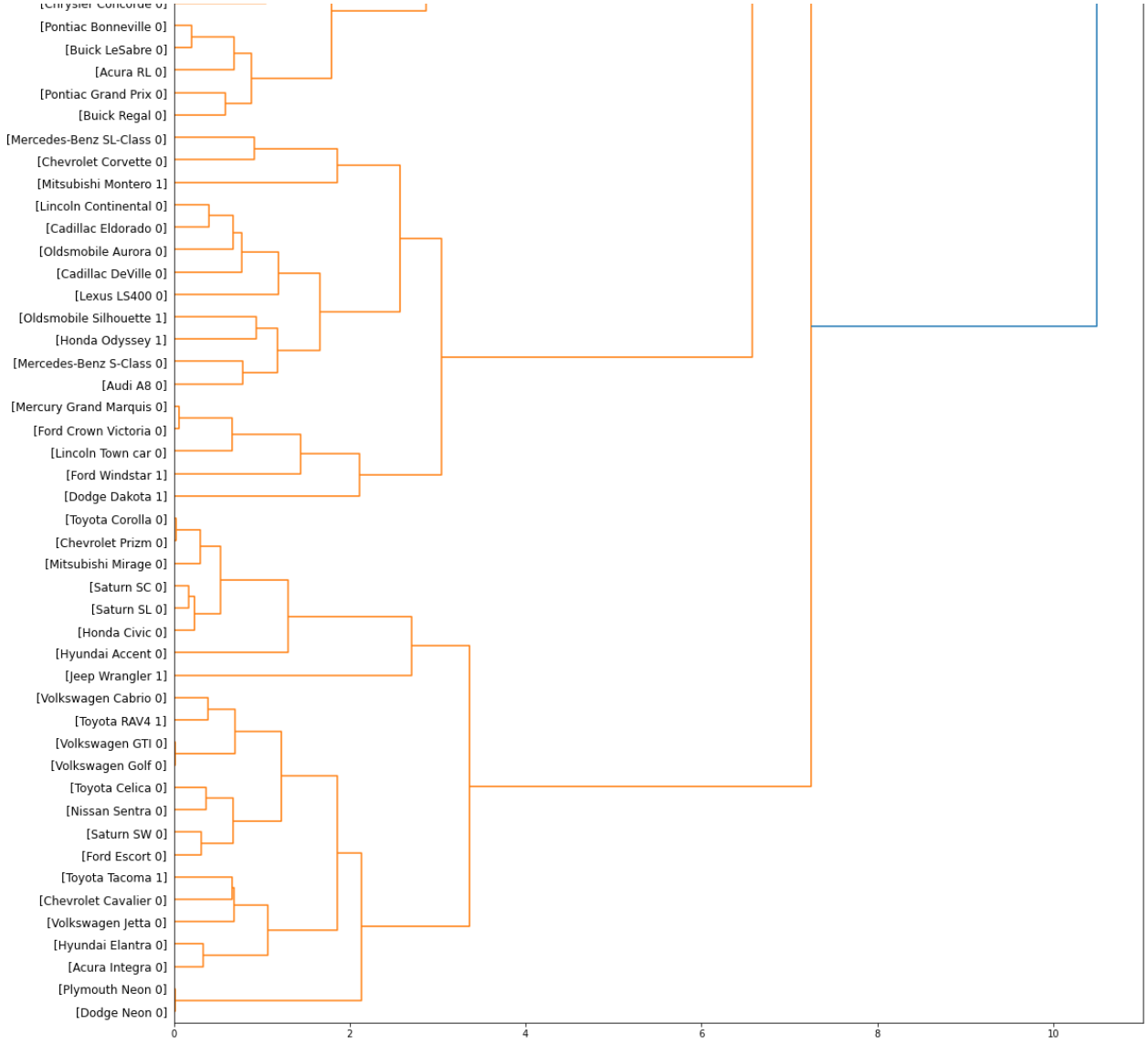
In [136…

```
Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

In [137…

```
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (pdf['manufact'][id], pdf['model'][id], int(float(pdf['type']

dendro = hierarchy.dendrogram(Z_using_dist_matrix,  leaf_label_func=llf, leaf_rotation=
```





In [138…

```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(dist_matrix)

agglom.labels_
```

Out[138…

```
array([1, 2, 2, 3, 2, 4, 3, 2, 2, 2, 2, 2, 4, 4, 2, 1, 3, 2, 2, 2, 4, 1,
       5, 3, 3, 2, 3, 2, 1, 3, 3, 0, 0, 0, 0, 4, 2, 1, 3, 3, 2, 4, 2, 4,
       0, 3, 0, 1, 3, 3, 2, 4, 1, 1, 3, 2, 1, 3, 2, 2, 2, 4, 4, 4, 1, 3,
       3, 2, 3, 4, 3, 3, 3, 2, 4, 2, 2, 3, 2, 4, 4, 1, 3, 2, 2, 2, 3, 4,
       2, 4, 1, 3, 2, 3, 3, 2, 2, 2, 3, 3, 3, 1, 1, 1, 1, 3, 2, 1, 1, 1,
       3, 0, 1, 1, 3, 1, 1], dtype=int64)
```

In [139…

```
# add new field to show the cluster of each row
pdf['cluster_'] = agglom.labels_
pdf.head()
```

Out[139…

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition	cluster_
0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0	2.828	0.0	1
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0	3.673	0.0	2
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0	2.150	0.0	2

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition	cluster_
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0	3.015	0.0	3
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561	18.5	22.0	2.933	0.0	2

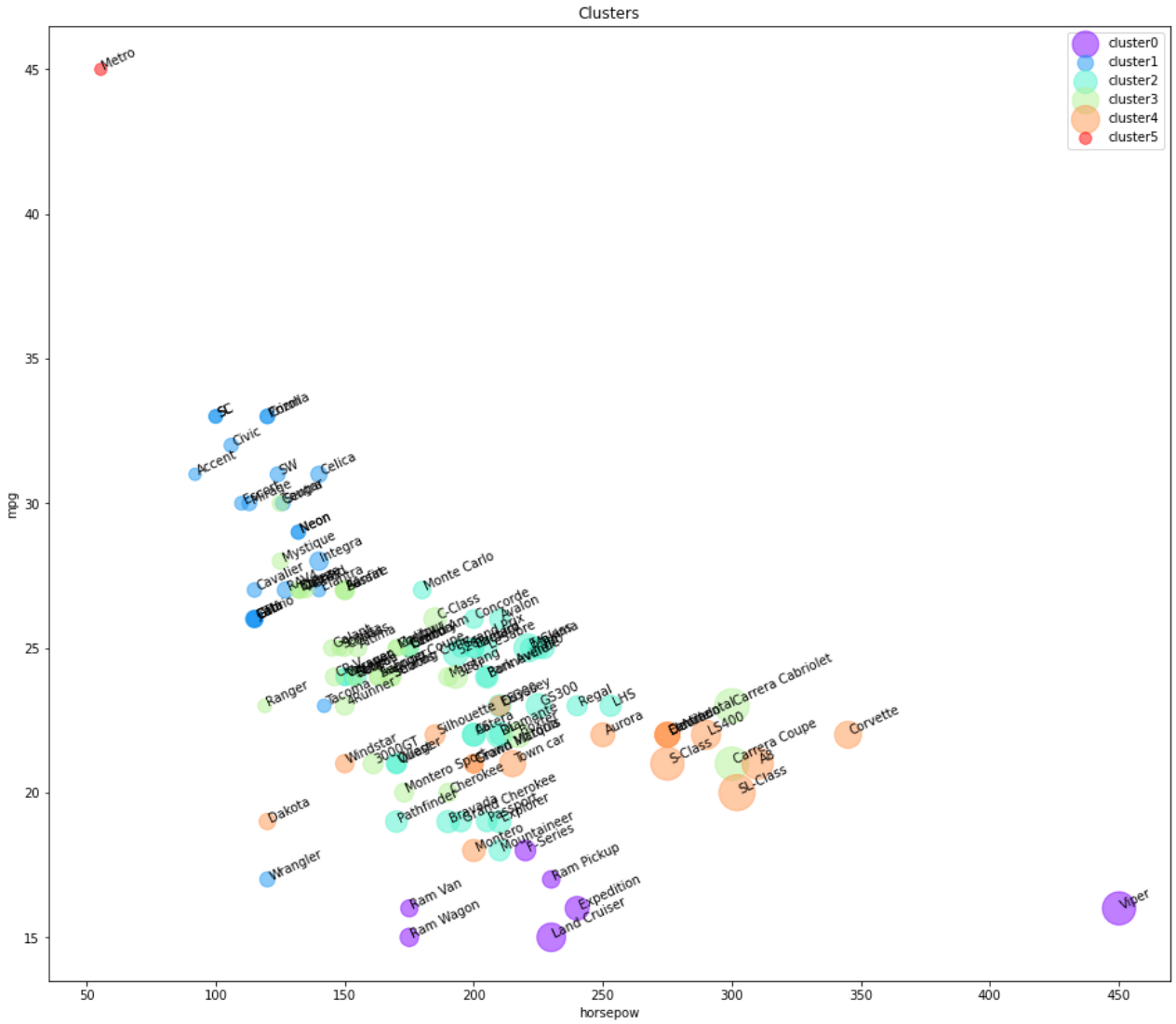
In [144…

```
# plot
import matplotlib.cm as cm
n_clusters = max(agglom.labels_)+1
colors = cm.rainbow(np.linspace(0, 1, n_clusters))
cluster_labels = list(range(0, n_clusters))

# Create a figure of size 6 inches by 4 inches.
plt.figure(figsize=(16,14))

for color, label in zip(colors, cluster_labels):
    subset = pdf[pdf.cluster_ == label]
    for i in subset.index:
            plt.text(subset.horsepow[i], subset.mpg[i],str(subset['model'][i]), rotatio
    plt.scatter(subset.horsepow, subset.mpg, s= subset.price*10, c=color, label='cluste
#    plt.scatter(subset.horsepow, subset.mpg)
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('mpg')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
```

Out[144…

```
Text(0, 0.5, 'mpg')
```



In [141…

```
# count number of cases in each group
pdf.groupby(['cluster_','type'])['cluster_'].count()
```

Out[141…

```
cluster_  type
0         0.0      1
          1.0      6
1         0.0     20
          1.0      3
2         0.0     26
          1.0     10
3         0.0     28
          1.0      5
4         0.0     12
          1.0      5
5         0.0      1
Name: cluster_, dtype: int64
```

In [142…

```
# check characteristics of each cluster
agg_cars = pdf.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean()
agg_cars
```

Out[142…

		horsepow	engine_s	mpg	price
cluster_	**type**				

cluster_	type	horsepow	engine_s	mpg	price
0	0.0	450.000000	8.000000	16.000000	69.725000
	1.0	211.666667	4.483333	16.166667	29.024667
1	0.0	118.500000	1.890000	29.550000	14.226100
	1.0	129.666667	2.300000	22.333333	14.292000
2	0.0	203.615385	3.284615	24.223077	27.988692
	1.0	182.000000	3.420000	20.300000	26.120600
3	0.0	168.107143	2.557143	25.107143	24.693786
	1.0	155.600000	2.840000	22.000000	19.807000
4	0.0	267.666667	4.566667	21.416667	46.417417
	1.0	173.000000	3.180000	20.600000	24.308400
5	0.0	55.000000	1.000000	45.000000	9.235000

It is obvious that we have 3 main clusters with the majority of vehicles in those.

Cars:

- Cluster 1: with almost high mpg, and low in horsepower.
- Cluster 2: with good mpg and horsepower, but higher price than average.
- Cluster 3: with low mpg, high horsepower, highest price.

Trucks:

- Cluster 1: with almost highest mpg among trucks, and lowest in horsepower and price.
- Cluster 2: with almost low mpg and medium horsepower, but higher price than average.
- Cluster 3: with good mpg and horsepower, low price.

Please notice that we did not use **type** and **price** of cars in the clustering process, but Hierarchical clustering could forge the clusters and discriminate them with quite a high accuracy.

In [143…

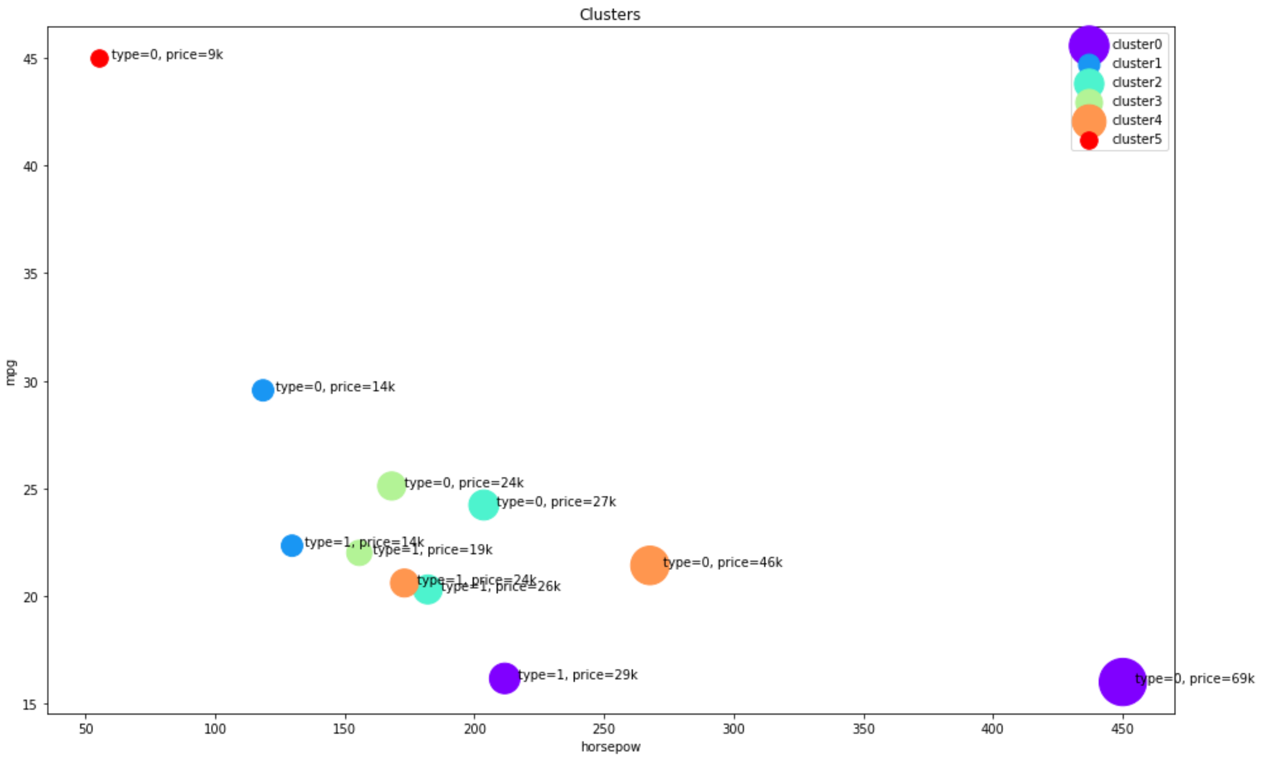
```
# plot
plt.figure(figsize=(16,10))
for color, label in zip(colors, cluster_labels):
    subset = agg_cars.loc[(label,),]
    for i in subset.index:
        plt.text(subset.loc[i][0]+5, subset.loc[i][2], 'type='+str(int(i)) + ', price='
    plt.scatter(subset.horsepow, subset.mpg, s=subset.price*20, c=color, label='cluster
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('mpg')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
```

```
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
```

Out[143…

```
Text(0, 0.5, 'mpg')
```



DBSCAN Clustering

Density-based Clustering:

- can have spherical shaped clusters or arbitrary shaped clusters
- k-means partition-based clustering assigns all points to a cluster, even if one really shouldn't be long
- density-based clustering locates regions of high density and separates outliers

DBSCAN:

- Density based spatial clustering of applications with noise

- one of the most common clustering algorithms
- works based on density of objects
- R: radius of neighborhood. If R includes enough points within it, it is a 'dense area'
- M: Min number of neighbors. the min number of datapoints we want in a neighborhood to define a cluster
- These clusters have:
  - core points: those which have the proper number of 'border points' around them to meet the R and M parameters
  - border points: those which do not meet the R and M parameters, but do for some other core point
  - outliers: aren't within a different neighborhood and don't have their own neighborhood
- connect core points that are neighbors and put them in one cluster

Lab: DBSCAN Clustering

Most of the traditional clustering techniques, such as k-means, hierarchical and fuzzy clustering, can be used to group data without supervision.

However, when applied to tasks with arbitrary shape clusters, or clusters within cluster, the traditional techniques might be unable to achieve good results. That is, elements in the same cluster might not share enough similarity or the performance may be poor. Additionally, Density-based clustering locates regions of high density that are separated from one another by regions of low density. Density, in this context, is defined as the number of points within a specified radius.

In this section, the main focus will be manipulating the data and properties of DBSCAN and observing the resulting clustering.

In [148…

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

Data Generation

The function below will generate the data points and requires these inputs:

- **centroidLocation**: Coordinates of the centroids that will generate the random data.
  - Example: input: [[4,3], [2,-1], [-1,4]]
- **numSamples**: The number of data points we want generated, split over the number of centroids (# of centroids defined in centroidLocation)
  - Example: 1500
- **clusterDeviation**: The standard deviation of the clusters. The larger the number, the further the spacing of the data points within the clusters.
  - Example: 0.5

In [149…

```
def createDataPoints(centroidLocation, numSamples, clusterDeviation):
```

```
    # Create random data and store in feature matrix X and response vector y.
    X, y = make_blobs(n_samples=numSamples, centers=centroidLocation,
                                cluster_std=clusterDeviation)

    # Standardize features by removing the mean and scaling to unit variance
    X = StandardScaler().fit_transform(X)
    return X, y
```

In [150…

```
# use the function and store the output into x and y
X, y = createDataPoints([[4,3], [2,-1], [-1,4]] , 1500, 0.5)
```

Modeling

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This technique is one of the most common clustering algorithms which works based on density of object. The whole idea is that if a particular point belongs to a cluster, it should be near to lots of other points in that cluster.

It works based on two parameters: Epsilon and Minimum Points\ **Epsilon** determine a specified radius that if includes enough number of points within, we call it dense area\ **minimumSamples** determine the minimum number of data points we want in a neighborhood to define a cluster.

In [151…

```
epsilon = 0.3
minimumSamples = 7
db = DBSCAN(eps=epsilon, min_samples=minimumSamples).fit(X)
labels = db.labels_
labels
```

Out[151… `array([0, 1, 1, ..., 0, 1, 1], dtype=int64)`

In [152…

```
# replace all elements with 'true' in core_samples_mask that are in the cluster, false
# Firts, create an array of booleans using the labels from db.
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
core_samples_mask
```

Out[152… `array([ True,  True,  True, ...,  True,  True,  True])`

In [153…

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_
```

Out[153… `3`

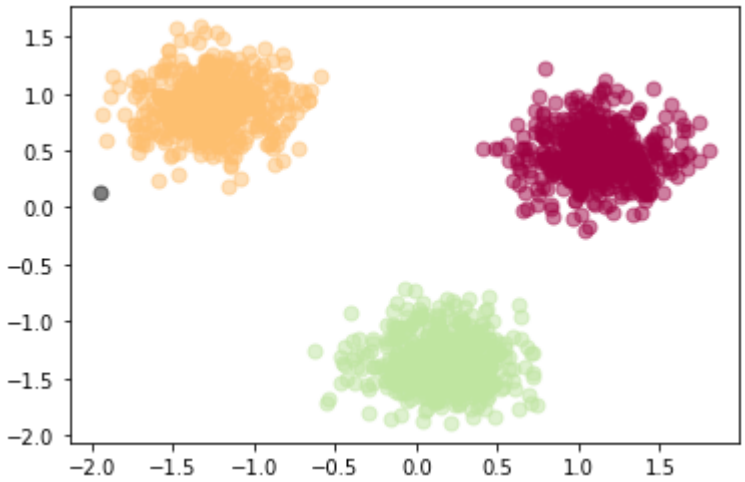
In [154…

```
# Remove repetition in labels by turning it into a set.
unique_labels = set(labels)
unique_labels
```

Out[154… `{-1, 0, 1, 2}`

In [155…
```
# Create colors for the clusters.
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
```

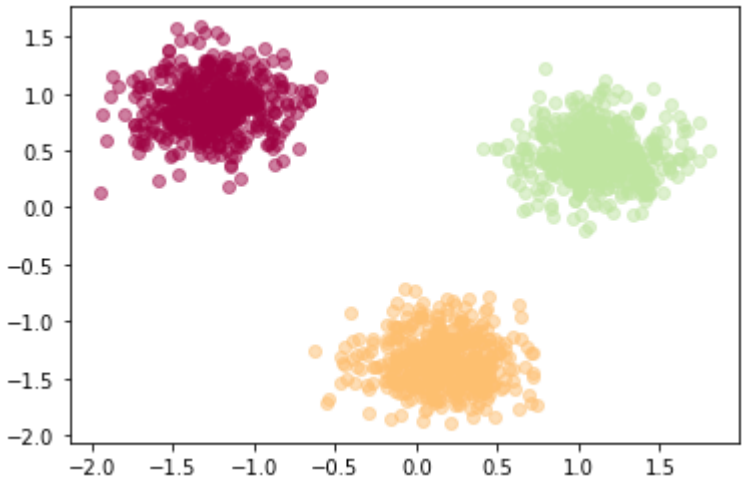
In [156…
```
# Plot the points with colors
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'

    class_member_mask = (labels == k)

    # Plot the datapoints that are clustered
    xy = X[class_member_mask & core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)

    # Plot the outliers
    xy = X[class_member_mask & ~core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)
```



In [157…
```
# try to cluster the above dataset into 3 clusters using k-means
from sklearn.cluster import KMeans
k = 3
k_means3 = KMeans(init = "k-means++", n_clusters = k, n_init = 12)
k_means3.fit(X)
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(k), colors):
    my_members = (k_means3.labels_ == k)
    plt.scatter(X[my_members, 0], X[my_members, 1],  c=col, marker=u'o', alpha=0.5)
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
```

```
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
```



Weather Station Clustering

DBSCAN is especially very good for tasks like class identification in a spatial context. The wonderful attribute of DBSCAN algorithm is that it can find out any arbitrary shape cluster without getting affected by noise. For example, this following example cluster the location of weather stations in Canada. \<Click 1> DBSCAN can be used here, for instance, to find the group of stations which show the same weather condition. As you can see, it not only finds different arbitrary shaped clusters, can find the denser part of data-centered samples by ignoring less-dense areas or noises.

Let's start playing with the data. We will be working according to the following workflow:

1. Loading data

- Overview data
- Data cleaning
- Data selection
- Clustering

Environment Canada
Monthly Values for July - 2015

Name in the table	Meaning
Stn_Name	**Station Name**
Lat	**Latitude (North+, degrees)**
Long	**Longitude (West - , degrees)**
Prov	Province
Tm	Mean Temperature (°C)
DwTm	Days without Valid Mean Temperature
D	Mean Temperature difference from Normal (1981-2010) (°C)
Tx	Highest Monthly Maximum Temperature (°C)
DwTx	Days without Valid Maximum Temperature

Tn	Lowest Monthly Minimum Temperature (°C)
DwTn	Days without Valid Minimum Temperature
S	Snowfall (cm)
DwS	Days without Valid Snowfall
S%N	Percent of Normal (1981-2010) Snowfall
P	**Total Precipitation (mm)**
DwP	Days without Valid Precipitation
P%N	Percent of Normal (1981-2010) Precipitation
S_G	Snow on the ground at the end of the month (cm)
Pd	Number of days with Precipitation 1.0 mm or more
BS	Bright Sunshine (hours)
DwBS	Days without Valid Bright Sunshine
BS%	Percent of Normal (1981-2010) Bright Sunshine
HDD	Degree Days below 18 °C
CDD	Degree Days above 18 °C
Stn_No	Climate station identifier (first 3 digits indicate drainage basin, last 4 characters are for sorting alphabetically).
NA	Not Available

In [158…

```
# download data
!wget -O weather-stations20140101-20141231.csv https://cf-courses-data.s3.us.cloud-obje
```

```
--2021-08-12 18:35:47--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/weather-statio
ns20140101-20141231.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 169.45.118.108
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|169.45.118.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 129821 (127K) [text/csv]
Saving to: 'weather-stations20140101-20141231.csv'

     0K .......... .......... .......... .......... .......... 39%  589K 0s
    50K .......... .......... .......... .......... .......... 78%  781K 0s
   100K .......... .......... ......                          100%  630K=0.2s

2021-08-12 18:35:48 (662 KB/s) - 'weather-stations20140101-20141231.csv' saved [129821/1
29821]
```

In [159…

```
# Load data
import csv
import pandas as pd
import numpy as np

filename='weather-stations20140101-20141231.csv'

#Read csv
```

```
pdf = pd.read_csv(filename)
pdf.head(5)
```

Out[159…

	Stn_Name	Lat	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%N	S_G	Pd	BS	DwBS	S%S	HDD	CDD	Stn_No
0	CHEMAINUS	48.935	123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500
1	COWICHAN LAKE FORESTRY	48.824	124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040
2	LAKE COWICHAN	48.829	124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055
3	DISCOVERY ISLAND	48.425	123.226	BC	NaN	NaN	NaN	12.5	0.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1012475
4	DUNCAN KELVIN CREEK	48.735	123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573

5 rows × 25 columns

In [160…

```
# remove rows that don't have value in Tm field
pdf = pdf[pd.notnull(pdf["Tm"])]
pdf = pdf.reset_index(drop=True)
pdf.head(5)
```

Out[160…

	Stn_Name	Lat	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%N	S_G	Pd	BS	DwBS	S%S	HDD	CDD	Stn_No
0	CHEMAINUS	48.935	123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500
1	COWICHAN LAKE FORESTRY	48.824	124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040
2	LAKE COWICHAN	48.829	124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055
3	DUNCAN KELVIN CREEK	48.735	123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573
4	ESQUIMALT HARBOUR	48.432	123.439	BC	8.8	0.0	NaN	13.1	0.0	1.9	...	8.0	NaN	NaN	12.0	NaN	NaN	NaN	258.6	0.0	1012710

5 rows × 25 columns

In [ ]:

```
# create visualization of stations
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

llon=-140
ulon=-50
llat=40
ulat=65
```

```
pdf = pdf[(pdf['Long'] > llon) & (pdf['Long'] < ulon) & (pdf['Lat'] > llat) &(pdf['Lat'

my_map = Basemap(projection='merc',
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (ll
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (ur

my_map.drawcoastlines()
my_map.drawcountries()
# my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To collect data based on stations

xs,ys = my_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))
pdf['xm']= xs.tolist()
pdf['ym'] =ys.tolist()

#Visualization1
for index,row in pdf.iterrows():
#   x,y = my_map(row.Long, row.Lat)
   my_map.plot(row.xm, row.ym,markerfacecolor =([1,0,0]),  marker='o', markersize= 5, a
#plt.text(x,y,stn)
plt.show()
```

In [ ]:

```
#

from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.15, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))


# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)
```

In [ ]:

```
set(labels)
```

In [ ]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
```

```
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (ll
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (ur

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))



#Visualization1
for clust_number in set(labels):
    c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color =c,  marker='o', s= 20, alpha = 0.
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red',)
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm)))
```

In [ ]:

```
from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym','Tx','Tm','Tn']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))


# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)
```

In [ ]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
```

```
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (ll
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (ur

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))



#Visualization1
for clust_number in set(labels):
    c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color =c,  marker='o', s= 20, alpha = 0.
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red',)
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm)))
```

Week 5: Recommender Systems

Introduction to Recommender Systems

Recommender Systems:

- these systems capture the pattern of peoples' behavior and use it to predict what else they may want or like.

Two types of systems:

- content based: 'show me more of the same of what I have liked before'
- collaberative filtering: 'tell me what's popular among my neighbors, I may also like it'

Implementing systems:

- memory-based:
  - uses the entire user-item dataset to generate a recommendation
  - uses statistical techniques to approximate users or items, i.e. pearson correlation, cosine similarity, euclidean distance, etc
- model-based:
  - develops a model of users in an attempt to learn their preferences
  - models can be created using ML techniques like regression, clustering, classification, etc

Content-based Recommender Systems

Content-based system:

1. create a vector to show the user's current ratings for movies ('input user ratings')
2. encode the movies with one hot encoding
3. genres can be used as a feature set matrix
4. multiply the vector and the matrix to get another matrix ('weighted genre matrix'), which represents the interest of the user for each genre based on movies they have seen
5. given this matrix, we can shape the profile of the active user. i.e., aggregate the weighted genres and normalize them to find the user profile

Lab: Content-based Recommendation Systems

In [4]:
```
# download dataset
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
print('unziping ...')
!unzip -o -j moviedataset.zip
```

```
unziping ...
```

In [9]:
```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

In [ ]:
```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in their t
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that may ha
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

In [ ]:
```
#split values in genre column
#Every genre is separated by a | so we simply have to call the split function on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

In [ ]:
```
#Copying the movie dataframe into a new one since we won't need to use the genre inform
moviesWithGenres_df = movies_df.copy()
```

```
#For every row in the dataframe, iterate through the list of genres and place a 1 into
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that column's genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

In [ ]:
```
ratings_df.head()
```

In [ ]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

In [ ]:
```
# create user input dictionary
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

In [ ]:
```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

In [ ]:
```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movie
userMovies
```

In [ ]:
```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop(
userGenreTable
```

In [ ]:
```
inputMovies['rating']
```

In [ ]:
```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
```

```
#The user profile
userProfile
```

In [ ]:
```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('yea
genreTable.head()
```

In [ ]:
```
genreTable.shape
```

In [ ]:
```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

In [ ]:
```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

In [ ]:
```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Advantages and Disadvantages of Content-Based Filtering

Advantages

- Learns user's preferences
- Highly personalized for the user

Disadvantages

- Doesn't take into account what others think of the item, so low quality item recommendations might happen
- Extracting data is not always intuitive
- Determining what characteristics of the item the user dislikes or likes is not always obvious

Collaborative Filtering

Collaberative Filtering:

- user-based:
  - based on users' neighborhood
- item-based:
  - based on items' similarity

Lab: Collaborative Filtering

In [ ]:
```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
```

In [ ]:
```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in their t
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that may ha
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
```

In [ ]:
```
movies_df.head()
```

In [ ]:
```
#Dropping the genres column
movies_df = movies_df.drop('genres', 1)
```

In [ ]:
```
movies_df.head()
```

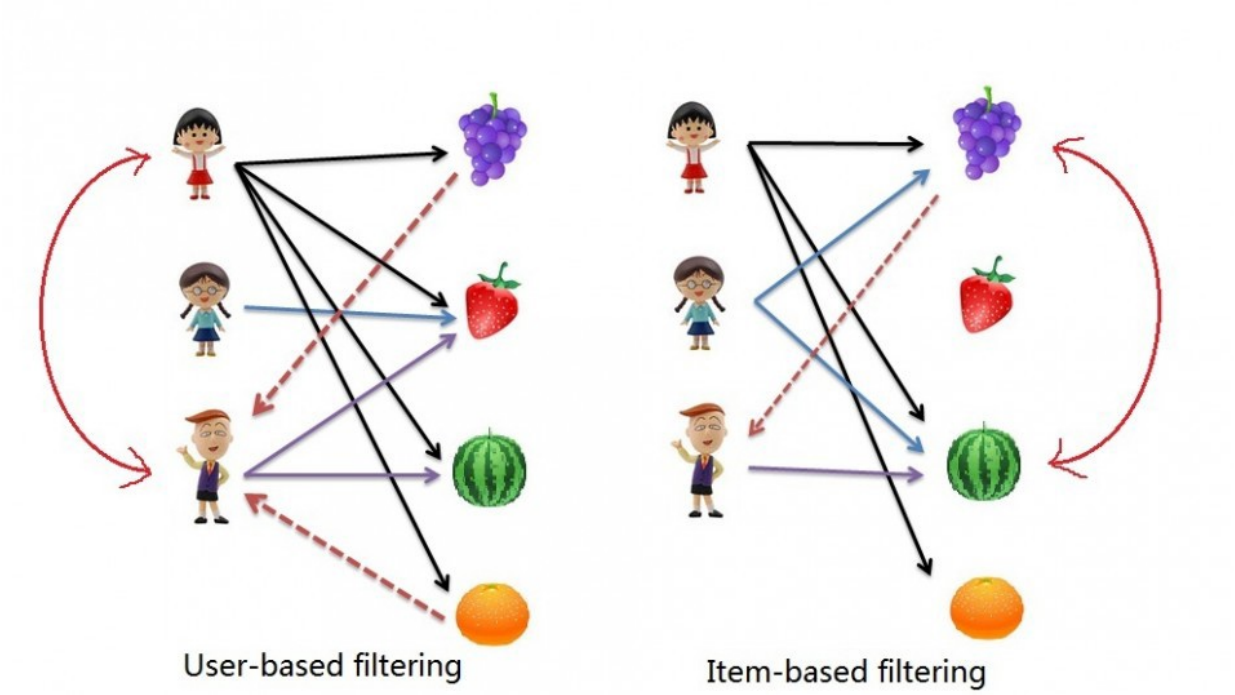
In [ ]:
```
ratings_df.head()
```

In [ ]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
```

In [ ]:
```
ratings_df.head()
```

Now it's time to start our work on recommendation systems.

The first technique we're going to take a look at is called **Collaborative Filtering**, which is also known as **User-User Filtering**. As hinted by its alternate name, this technique uses other users to recommend items to the input user. It attempts to find users that have similar preferences and opinions as the input and then recommends items that they have liked to the input. There are several methods of finding similar users (Even some making use of Machine Learning), and the one we will be using here is going to be based on the **Pearson Correlation Function**.



The process for creating a User Based recommendation system is as follows:

- Select a user with the movies the user has watched
- Based on his rating to movies, find the top X neighbours
- Get the watched movie record of the user for each neighbour.
- Calculate a similarity score using some formula
- Recommend the items with the highest score

Let's begin by creating an input user to recommend movies to:

Notice: To add more movies, simply increase the amount of elements in the userInput. Feel free to add more in! Just be sure to write it in with capital letters and if a movie starts with a "The", like "The Matrix" then write it in like this: 'Matrix, The' .

In [ ]:
```
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

In [ ]:
```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
```

```
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

In [ ]:
```
#Filtering out users that have watched movies that the input has watched and storing it
userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
userSubset.head()
```

In [ ]:
```
#Groupby creates several sub dataframes where they all have the same value in the colum
userSubsetGroup = userSubset.groupby(['userId'])
```

In [ ]:
```
userSubsetGroup.get_group(1130)
```

In [ ]:
```
#Sorting it so users with movie most in common with the input will have priority
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

In [ ]:
```
userSubsetGroup[0:3]
```

Similiarity of users to input user

Next, we are going to compare all users (not really all !!!) to our specified user and find the one that is most similar.\ we're going to find out how similar each user is to the input through the **Pearson Correlation Coefficient**. It is used to measure the strength of a linear association between two variables. The formula for finding this coefficient between sets X and Y with N values can be seen in the image below.

Why Pearson Correlation?

Pearson correlation is invariant to scaling, i.e. multiplying all elements by a nonzero constant or adding any constant to all elements. For example, if you have two vectors X and Y,then, pearson(X, Y) == pearson(X, 2 * Y + 3). This is a pretty important property in recommendation systems because for example two users might rate two series of items totally different in terms of absolute rates, but they would be similar users (i.e. with similar ideas) with similar rates in various scales .

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The values given by the formula vary from r = -1 to r = 1, where 1 forms a direct correlation between the two entities (it means a perfect positive correlation) and -1 forms a perfect negative correlation.

In our case, a 1 means that the two users have similar tastes while a -1 means the opposite.

In [ ]:
```
userSubsetGroup = userSubsetGroup[0:100]
```

In [ ]:
```
#Store the Pearson Correlation in a dictionary, where the key is the user Id and the va
pearsonCorrelationDict = {}

#For every user group in our subset
for name, group in userSubsetGroup:
    #Let's start by sorting the input and current user group so the values aren't mixed
    group = group.sort_values(by='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]
    #And then store them in a temporary buffer variable in a list format to facilitate 
    tempRatingList = temp_df['rating'].tolist()
    #Let's also put the current user group reviews in a list format
    tempGroupList = group['rating'].tolist()
    #Now let's calculate the pearson correlation between two users, so called, x and y
    Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float(nRatin
    Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(nRatings
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingList

    #If the denominator is different than zero, then divide, else, 0 correlation.
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
    else:
        pearsonCorrelationDict[name] = 0
```

In [ ]:
```
pearsonCorrelationDict.items()
```

In [ ]:
```
pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['userId'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
pearsonDF.head()
```

In [ ]:
```
topUsers=pearsonDF.sort_values(by='similarityIndex', ascending=False)[0:50]
topUsers.head()
```

In [ ]:
```
topUsersRating=topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inn
topUsersRating.head()
```

In [ ]:
```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['ra
topUsersRating.head()
```

In [ ]:
```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weight
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

In [ ]:
```
#Creates an empty dataframe
```

```
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_we
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

In [ ]:

```
recommendation_df = recommendation_df.sort_values(by='weighted average recommendation s
recommendation_df.head(10)
```

In [ ]:

```
movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['movieId'].tolist())
```

Advantages and Disadvantages of Collaborative Filtering

Advantages

- Takes other user's ratings into consideration
- Doesn't need to study or extract information from the recommended item
- Adapts to the user's interests which might change over time

Disadvantages

- Approximation function can be slow
- There might be a low of amount of users to approximate
- Privacy issues when trying to learn the user's preferences

In [ ]: