ML algorithms

November 1, 2020

[ ]:
```
#Linear Regression
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
%matplotlib inline
```

[3]:
```
df = pd.read_csv('FuelConsumptionCo2.csv')
```

[4]:
```
df.head()
```

[4]:
```
   MODELYEAR   MAKE       MODEL VEHICLECLASS  ENGINESIZE  CYLINDERS  \
0       2014  ACURA         ILX      COMPACT         2.0          4
1       2014  ACURA         ILX      COMPACT         2.4          4
2       2014  ACURA  ILX HYBRID      COMPACT         1.5          4
3       2014  ACURA     MDX 4WD  SUV - SMALL         3.5          6
4       2014  ACURA     RDX AWD  SUV - SMALL         3.5          6

  TRANSMISSION FUELTYPE  FUELCONSUMPTION_CITY  FUELCONSUMPTION_HWY  \
0          AS5        Z                   9.9                  6.7
1           M6        Z                  11.2                  7.7
2          AV7        Z                   6.0                  5.8
3          AS6        Z                  12.7                  9.1
4          AS6        Z                  12.1                  8.7

   FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  CO2EMISSIONS
0                   8.5                        33           196
1                   9.6                        29           221
2                   5.9                        48           136
3                  11.1                        25           255
4                  10.6                        27           244
```

[5]:
```
df.describe()
```

[5]:
```
       MODELYEAR   ENGINESIZE    CYLINDERS  FUELCONSUMPTION_CITY  \
count     1067.0  1067.000000  1067.000000           1067.000000
mean      2014.0     3.346298     5.794752             13.296532
```

```
std         0.0     1.415895     1.797447               4.101253
min      2014.0     1.000000     3.000000               4.600000
25%      2014.0     2.000000     4.000000              10.250000
50%      2014.0     3.400000     6.000000              12.600000
75%      2014.0     4.300000     8.000000              15.550000
max      2014.0     8.400000    12.000000              30.200000

       FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  \
count          1067.000000           1067.000000               1067.000000
mean              9.474602             11.580881                 26.441425
std               2.794510              3.485595                  7.468702
min               4.900000              4.700000                 11.000000
25%               7.500000              9.000000                 21.000000
50%               8.800000             10.900000                 26.000000
75%              10.850000             13.350000                 31.000000
max              20.500000             25.800000                 60.000000

       CO2EMISSIONS
count   1067.000000
mean     256.228679
std       63.372304
min      108.000000
25%      207.000000
50%      251.000000
75%      294.000000
max      488.000000
```

[6]:

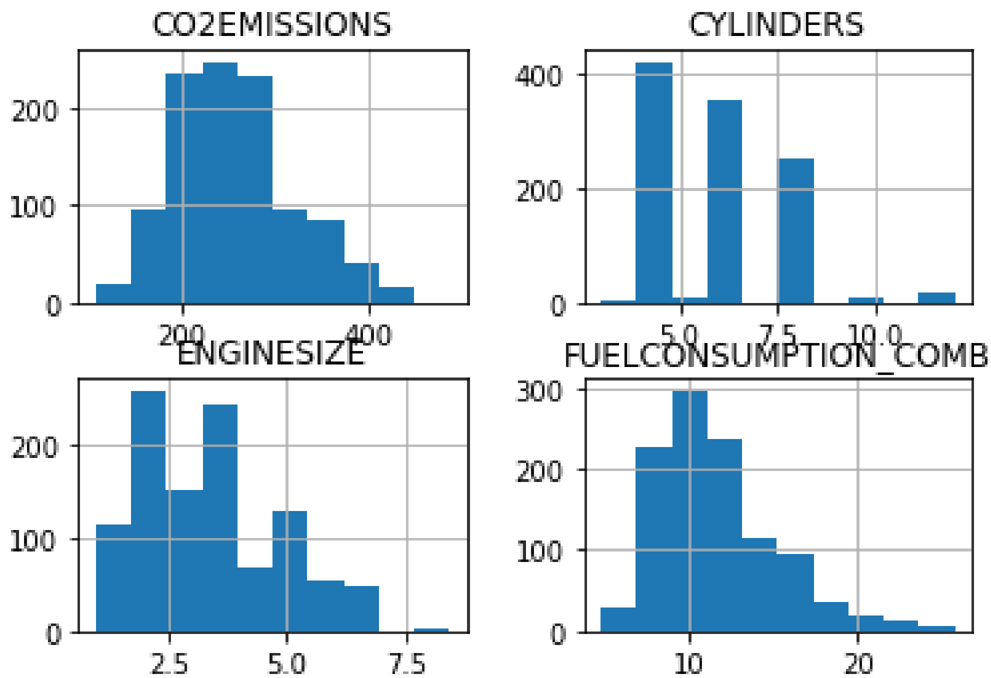
```
#simple linear regression
```

[7]:

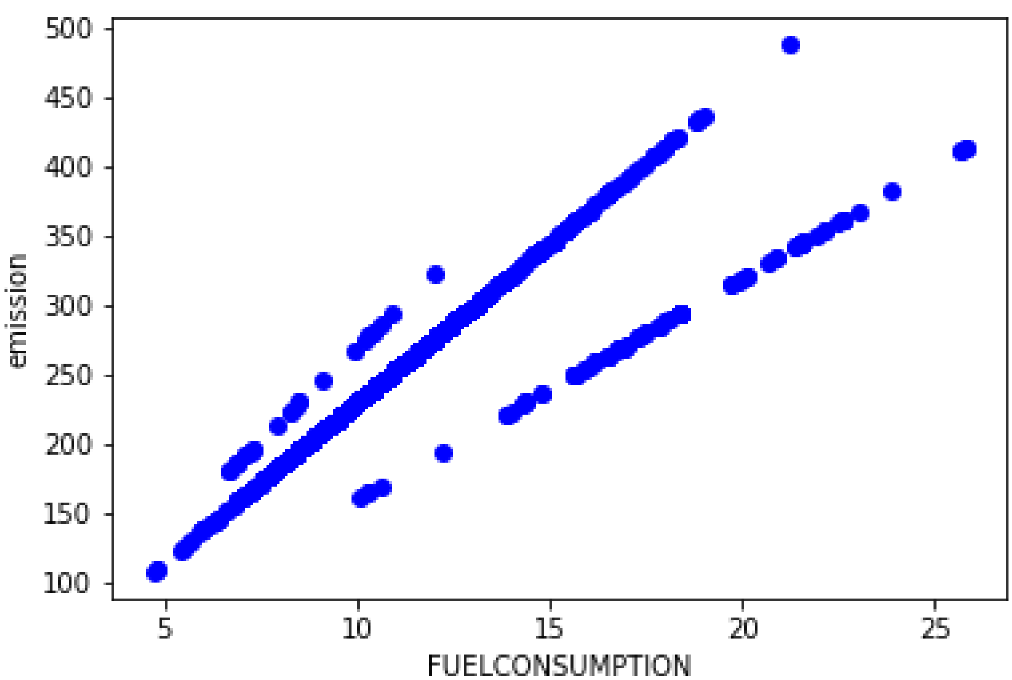
```
cdf = df[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB', 'CO2EMISSIONS']]
```

[8]:

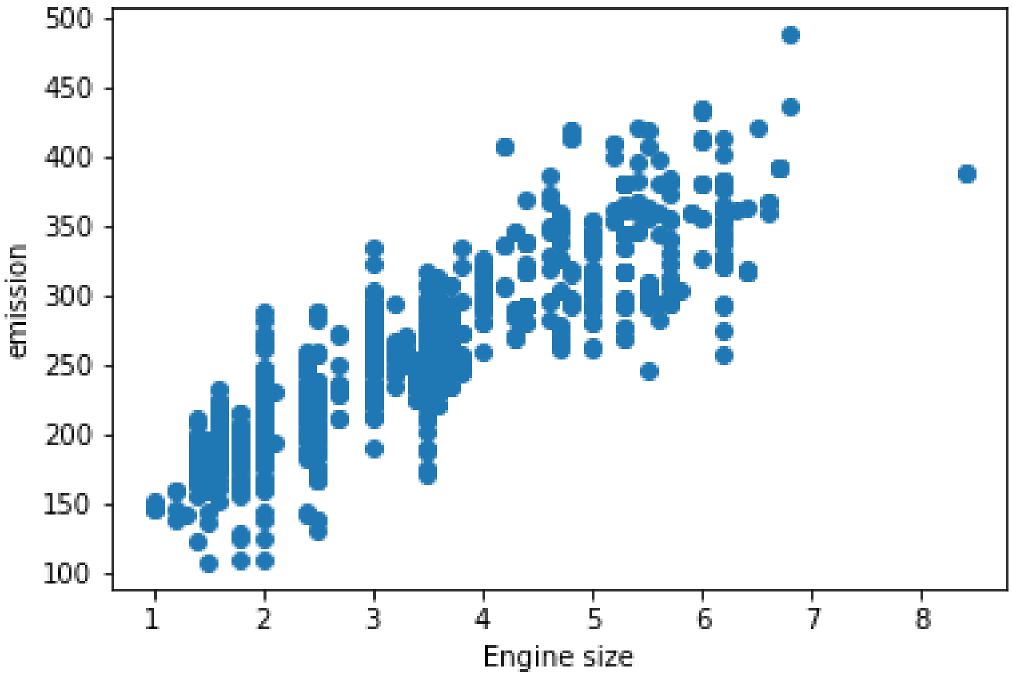
```
viz = cdf
viz.hist()
plt.show()
```



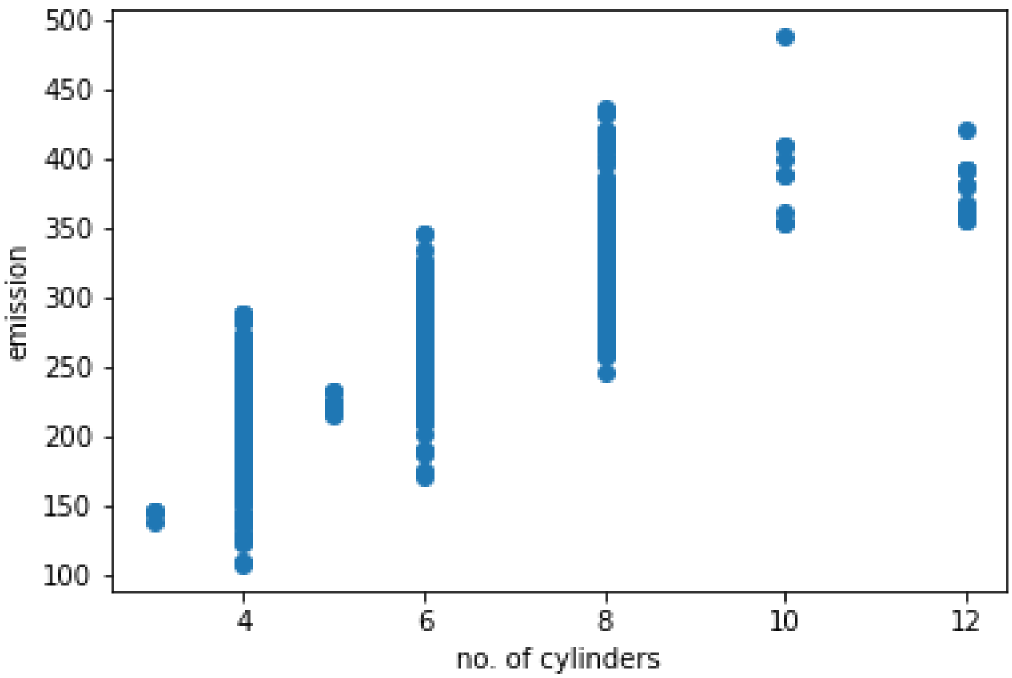
[9]:
```
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS, color='blue')
plt.xlabel('FUELCONSUMPTION')
plt.ylabel('emission')
plt.show()
```



[10]:
```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS)
plt.xlabel('Engine size')
plt.ylabel('emission')
plt.show()
```

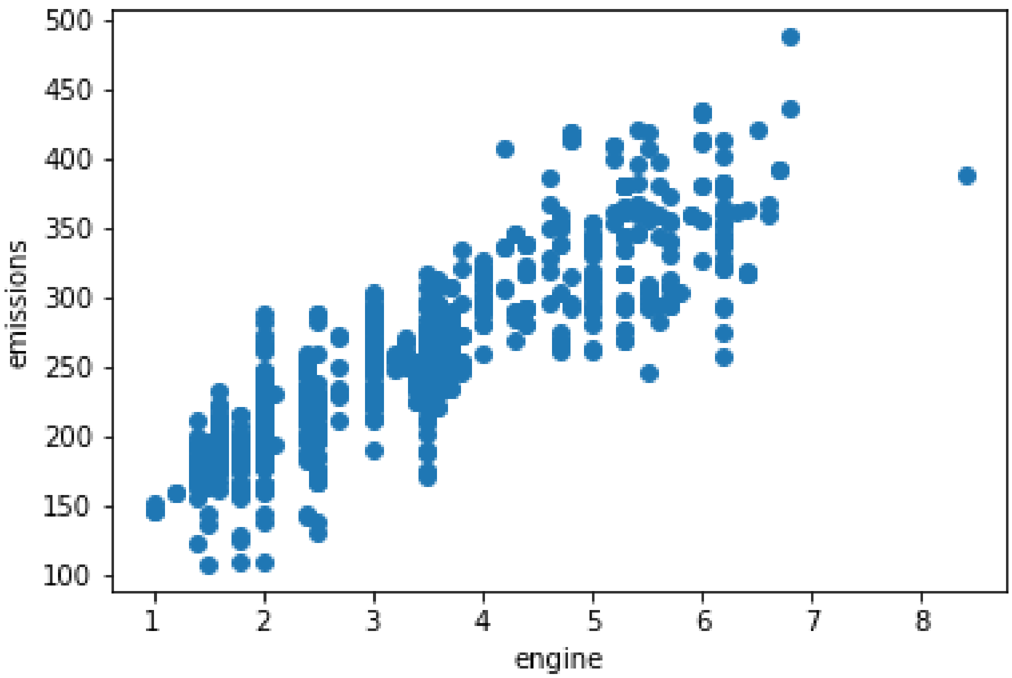


[11]:
```
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS)
plt.xlabel('no. of cylinders')
plt.ylabel('emission')
plt.show()
```



[12]:
```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

[13]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS)
plt.xlabel('engine')
plt.ylabel('emissions')
plt.show()
```




```
[14]: from sklearn import linear_model
      Regression = linear_model.LinearRegression()
      train_x = np.asanyarray(train[['ENGINESIZE']])
      train_y = np.asanyarray(train[['CO2EMISSIONS']])
      Regression.fit(train_x, train_y)
```

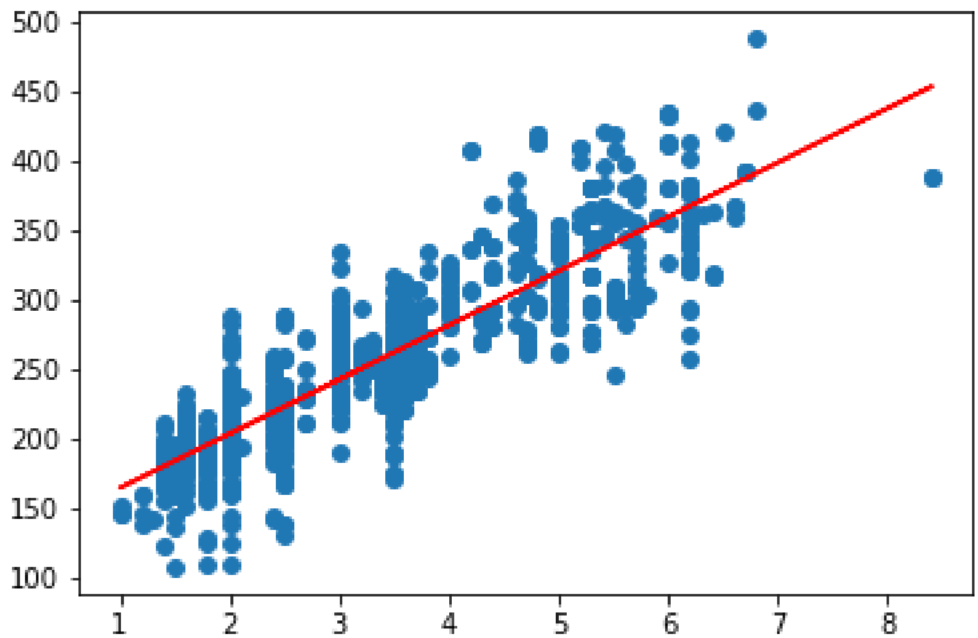
```
[14]: LinearRegression()
```

```
[15]: print('coefficient', Regression.coef_)
      print('intersept', Regression.intercept_)
```

```
coefficient [[39.02456543]]
intersept [125.76253682]
```

```
[16]: plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS)
      plt.plot(train_x, train_x * Regression.coef_ + Regression.intercept_, '-r')
```

```
[16]: [<matplotlib.lines.Line2D at 0x17c59c7f880>]
```



[17]:
```
from sklearn.metrics import r2_score
```

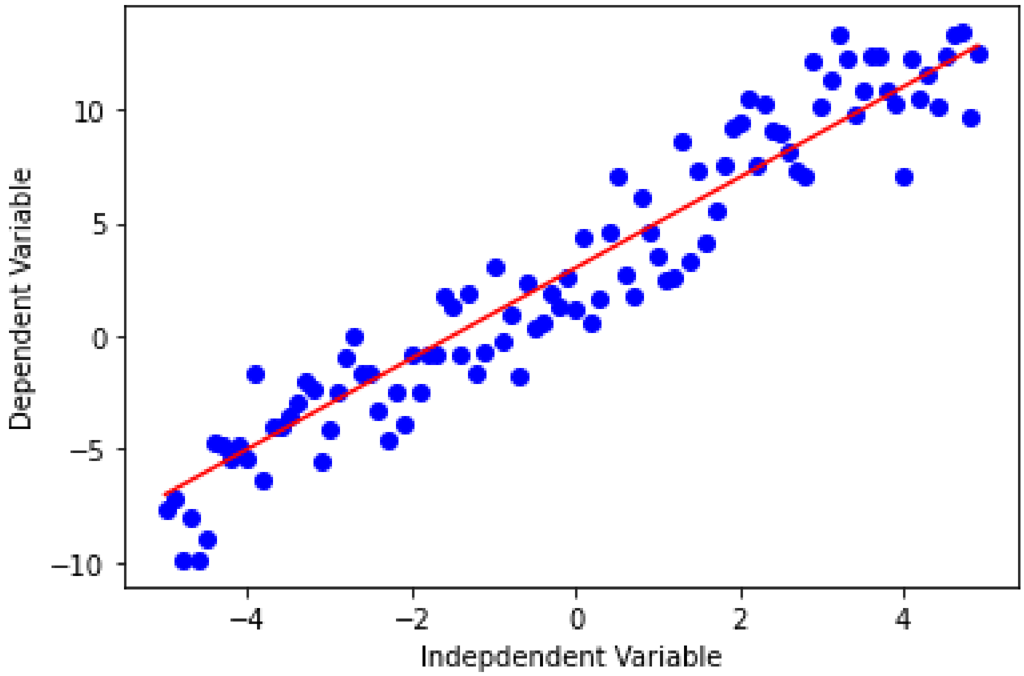
[18]:
```
test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])
test_y_hat = Regression.predict(test_x)

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

```
Mean absolute error: 22.09
Residual sum of squares (MSE): 811.08
R2-score: 0.74
```

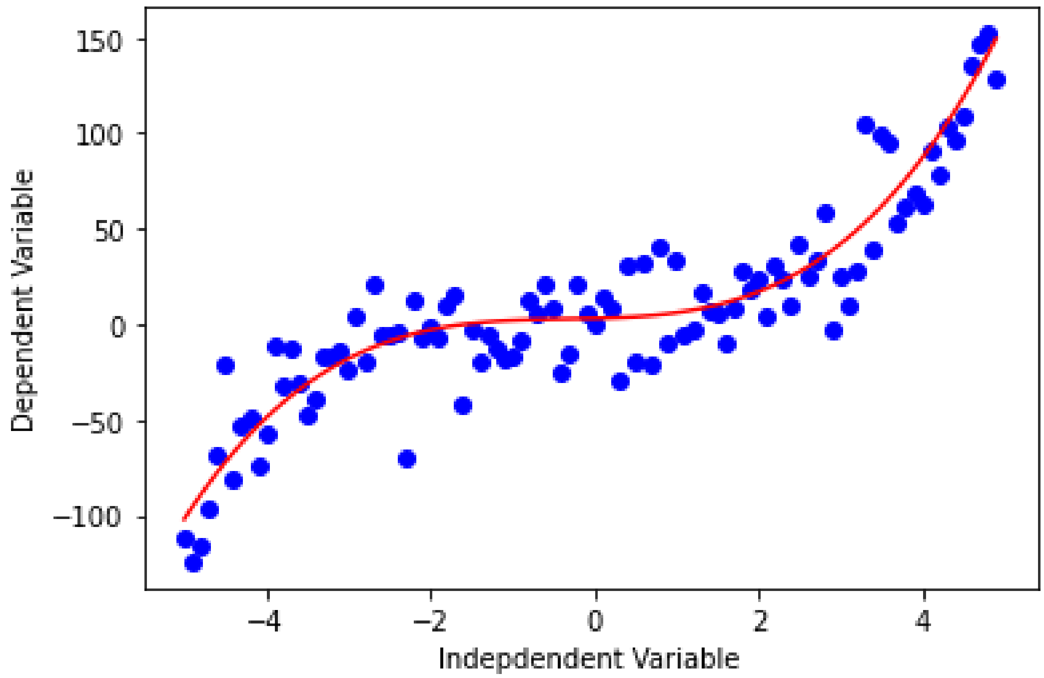
[19]:
```
#diff between linear and non-linear regression
```

[20]:
```
x = np.arange(-5.0, 5.0, 0.1)

y = 2*(x) + 3
y_noise = 2 * np.random.normal(size=x.size)
ydata = y + y_noise
#plt.figure(figsize=(8,6))
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
```

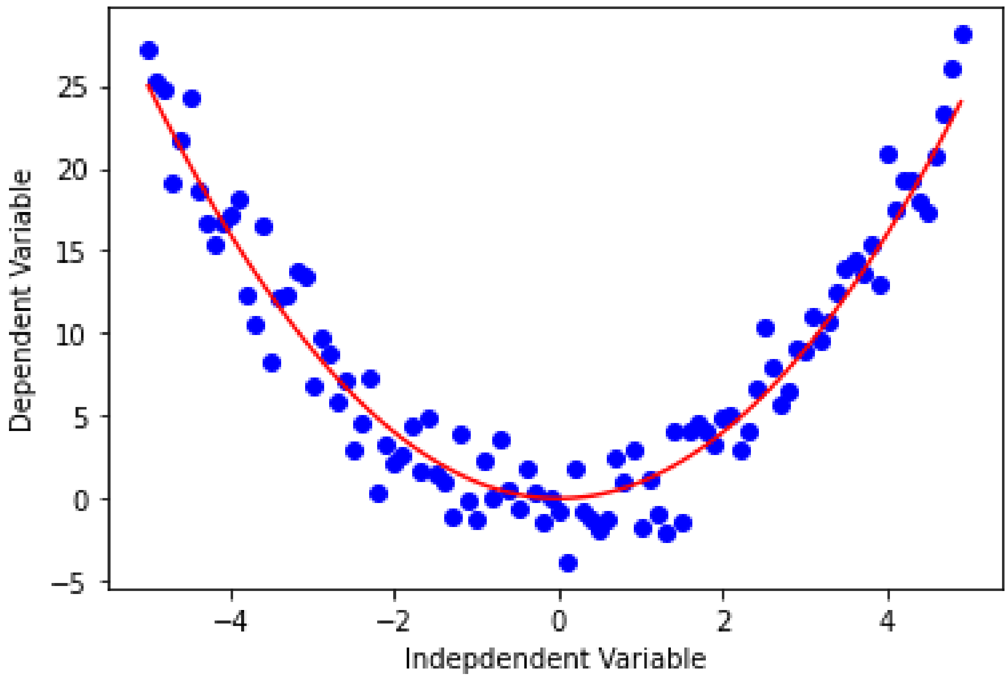
```
plt.xlabel('Indepdendent Variable')
plt.show()
```



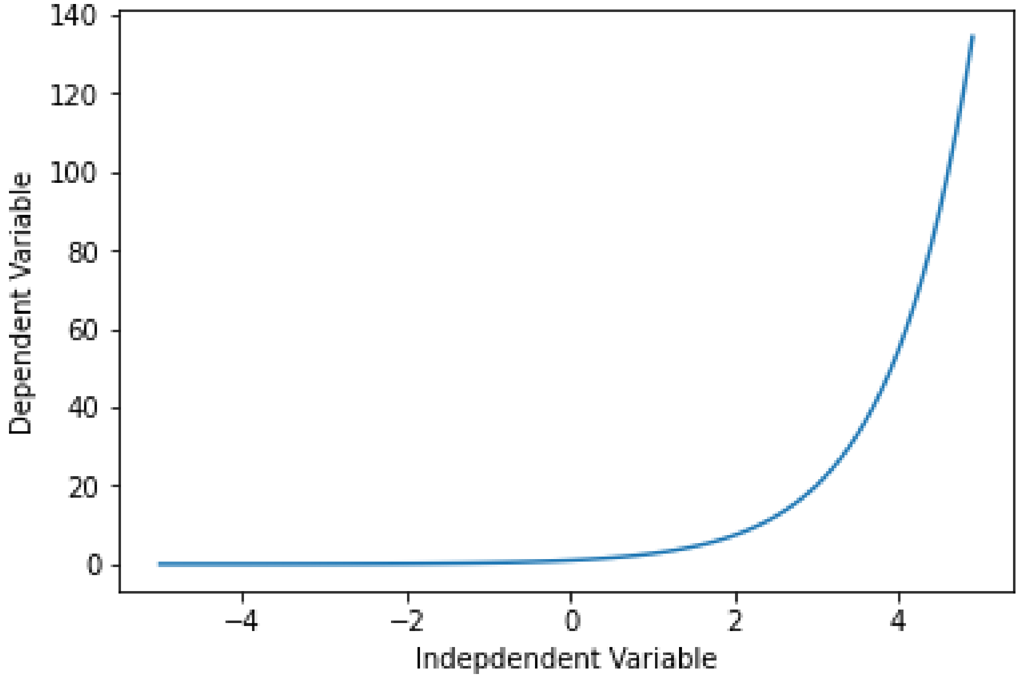
[21]:
```
#cubic
x = np.arange(-5.0, 5.0, 0.1)

y = 1*(x**3) + 1*(x**2) + 1*x + 3
y_noise = 20 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



[22]:
```
#quadradic
x = np.arange(-5.0, 5.0, 0.1)


y = np.power(x,2)
y_noise = 2 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```

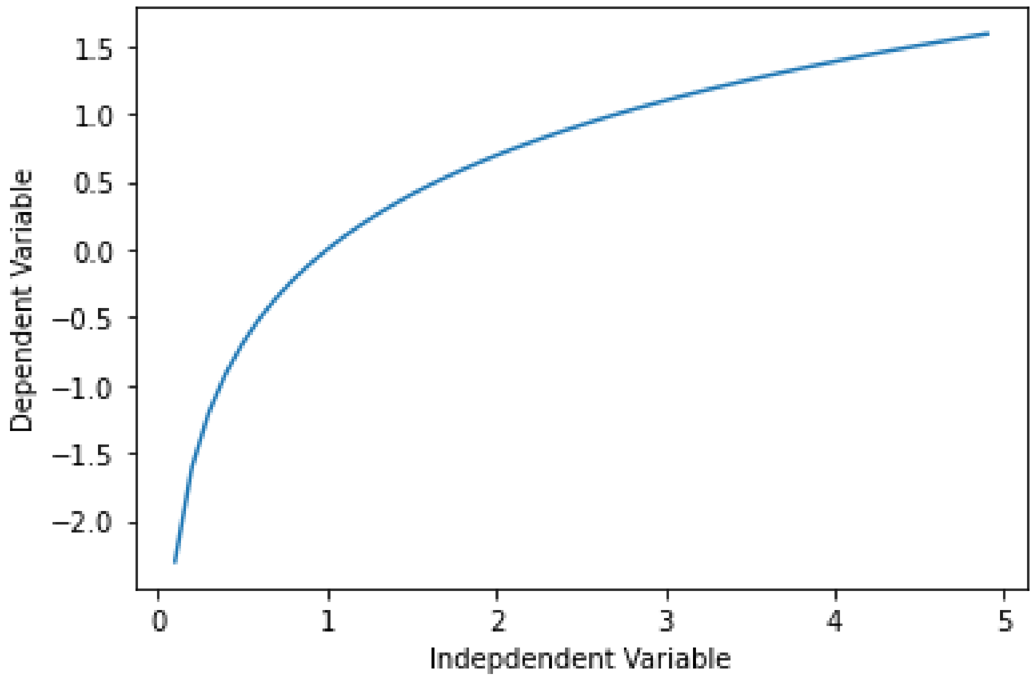


[23]:
```
#exponential
X = np.arange(-5.0, 5.0, 0.1)

Y= np.exp(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```

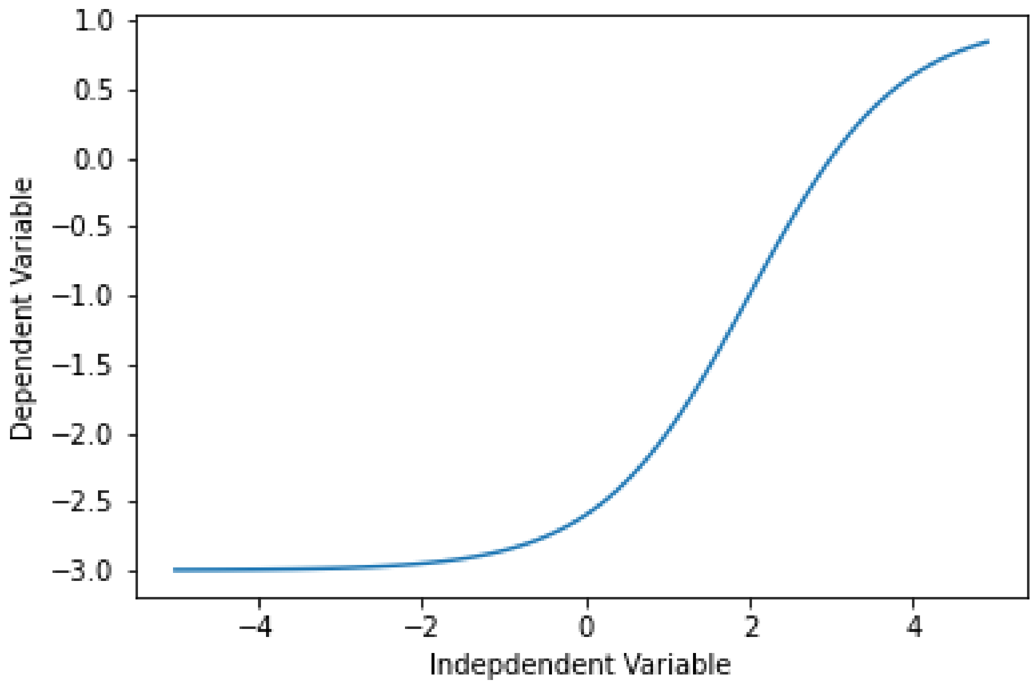


[24]:
```
#log
X = np.arange(-5.0, 5.0, 0.1)

Y = np.log(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```

```
<ipython-input-24-cdc1416ded40>:4: RuntimeWarning: invalid value encountered in
log
  Y = np.log(X)
```



[25]:
```
#sigmoidal/logistic
X = np.arange(-5.0, 5.0, 0.1)


Y = 1-4/(1+np.power(3, X-2))

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



[26]:
```
#non-linear regression
```

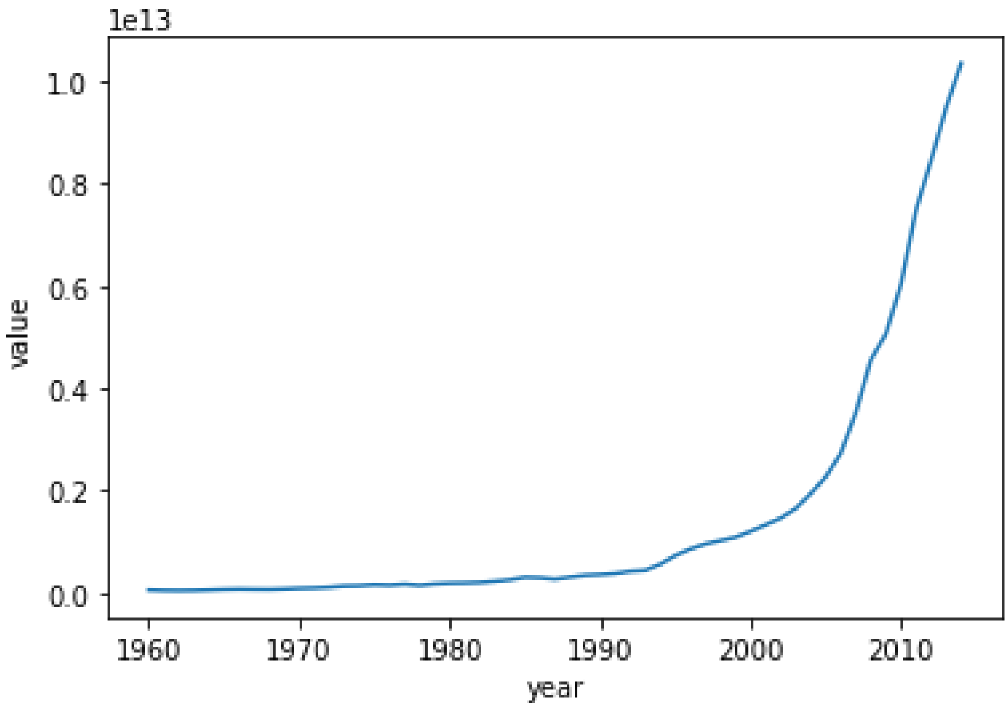
[27]:
```
df1 = pd.read_csv('China_gdp.csv')
```

[28]:
```
df1.head()
```

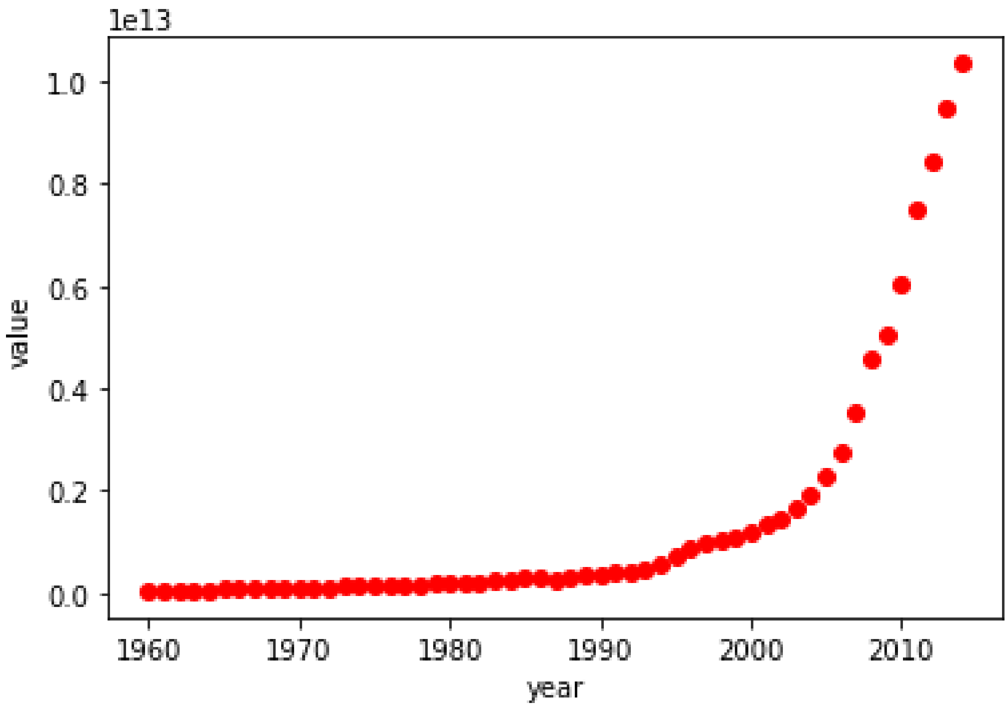
[28]:

	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

[29]:
```
plt.plot(df1.Year, df1.Value)
plt.xlabel('year')
plt.ylabel('value')
plt.show()
```



[30]:
```
plt.plot(df1['Year'].values, df1['Value'].values, 'ro')
plt.xlabel('year')
plt.ylabel('value')
plt.show()
```

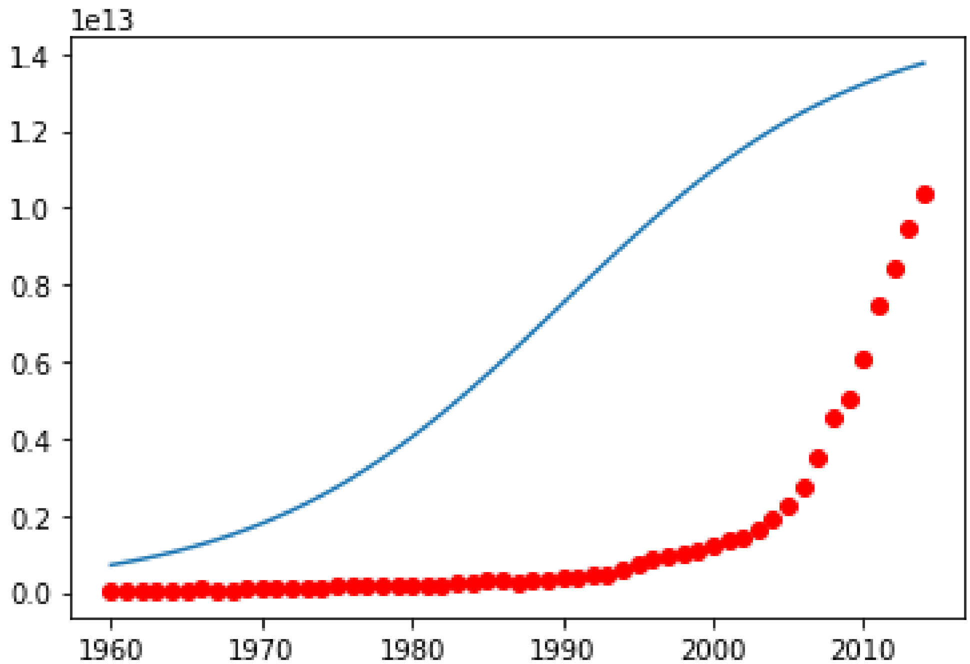


[31]:
```
def sigmoid(x, Beta_1, Beta_2):
     y = 1 / (1 + np.exp(-Beta_1*(x-Beta_2)))
     return y
```

[32]:
```
x_data = df1['Year']
y_data = df1['Value']
```

[33]:
```
beta_1 = 0.10
beta_2 = 1990.0

#logistic function
Y_pred = sigmoid(x_data, beta_1 , beta_2)

#plot initial prediction against datapoints
plt.plot(x_data, Y_pred*15000000000000.)
plt.plot(x_data, y_data, 'ro')
```

[33]: [<matplotlib.lines.Line2D at 0x17c59edbbb0>]



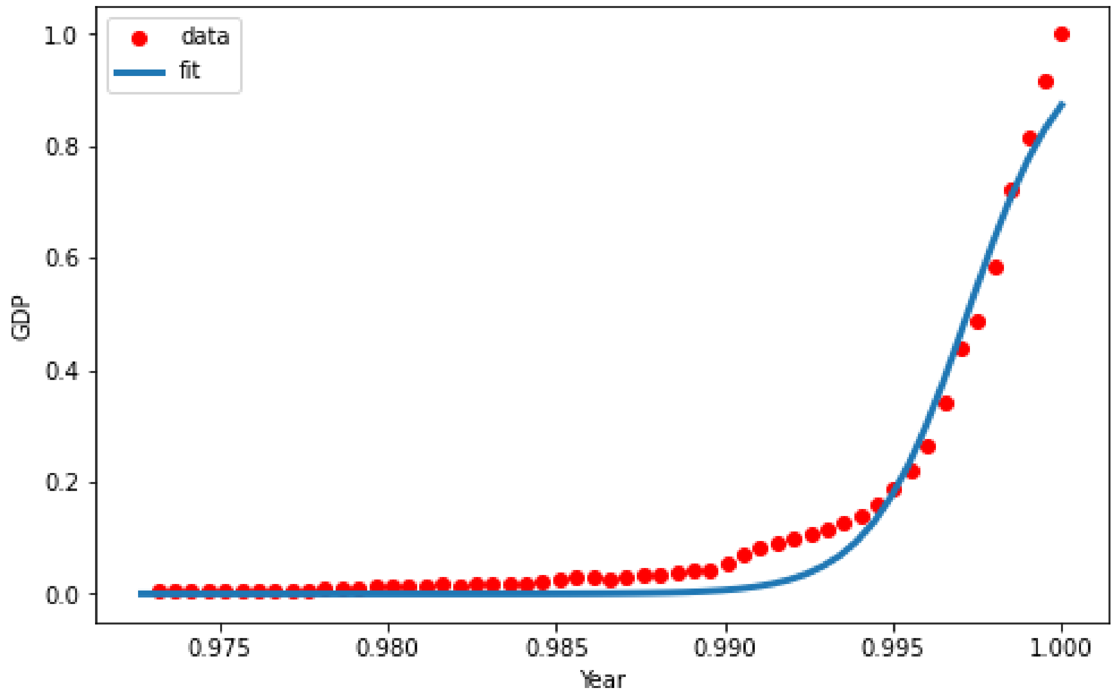
[34]:
```
# Lets normalize our data
xdata =x_data/max(x_data)
ydata =y_data/max(y_data)
```

[35]:
```
#clearly parameter doesnt fit our model.So, we'll use curve_fit
```

[36]:
```
from scipy.optimize import curve_fit
popt, pcov = curve_fit(sigmoid, xdata, ydata)
#print the final parameters
print(" beta_1 = %f, beta_2 = %f" % (popt[0], popt[1]))
```

```
beta_1 = 690.451711, beta_2 = 0.997207
```

[37]:
```
x = np.linspace(1960, 2015, 55)
x = x/max(x)
plt.figure(figsize=(8,5))
y = sigmoid(x, *popt)
plt.plot(xdata, ydata, 'ro', label='data')
plt.plot(x,y, linewidth=3.0, label='fit')
plt.legend(loc='best')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```



[38]:

```
#finding accuracy

# split data into train/test
msk = np.random.rand(len(df1)) < 0.8
train_x = xdata[msk]
test_x = xdata[~msk]
train_y = ydata[msk]
test_y = ydata[~msk]

# build the model using train set
popt, pcov = curve_fit(sigmoid, train_x, train_y)

# predict using test set
y_hat = sigmoid(test_x, *popt)

# evaluation
print("Mean absolute error: %.2f" % np.mean(np.absolute(y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((y_hat - test_y) ** 2))
from sklearn.metrics import r2_score
print("R2-score: %.2f" % r2_score(y_hat , test_y) )
```

```
Mean absolute error: 0.04
Residual sum of squares (MSE): 0.00
R2-score: 0.96
```

[39]:
```
#Classification
```

[40]:
```
#KNN
```

[41]:
```
import itertools
from matplotlib.ticker import NullFormatter
import matplotlib.ticker as ticker
from sklearn import preprocessing
```

[42]:
```
df2 = pd.read_csv('teleCust1000t.csv')
df2.head()
```

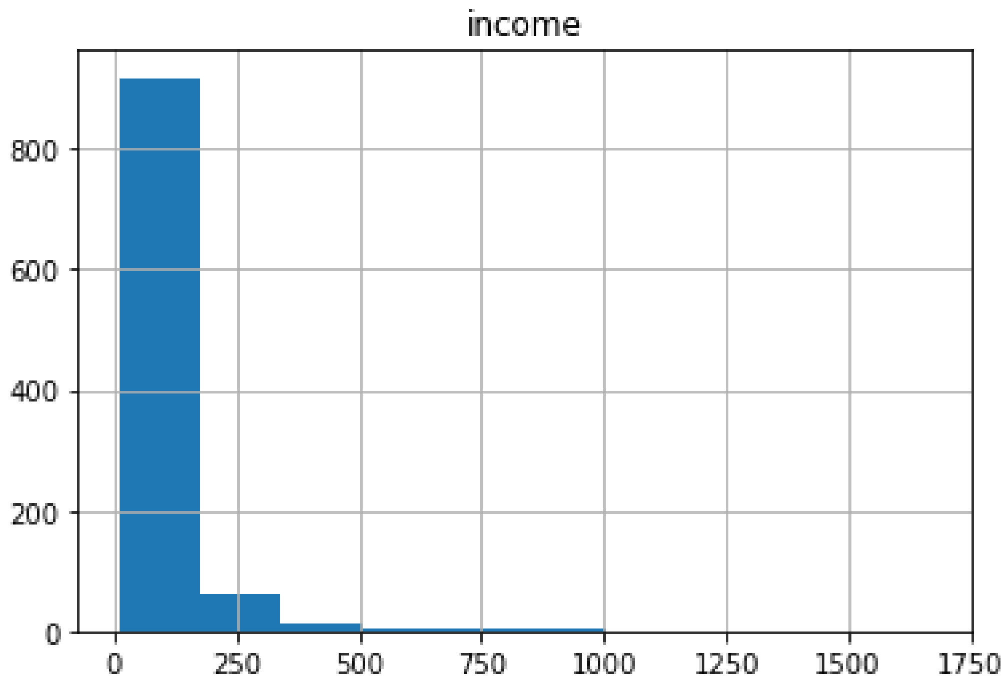
[42]:
```
   region  tenure  age  marital  address  income  ed  employ  retire  gender  \
0       2      13   44        1        9    64.0   4       5     0.0       0
1       3      11   33        1        7   136.0   5       5     0.0       0
2       3      68   52        1       24   116.0   1      29     0.0       1
3       2      33   33        0       12    33.0   2       0     0.0       1
4       2      23   30        1        9    30.0   1       2     0.0       0

   reside  custcat
0       2        1
1       6        4
2       2        3
3       1        1
4       4        3
```

[43]:
```
df2['custcat'].value_counts()
```

[43]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

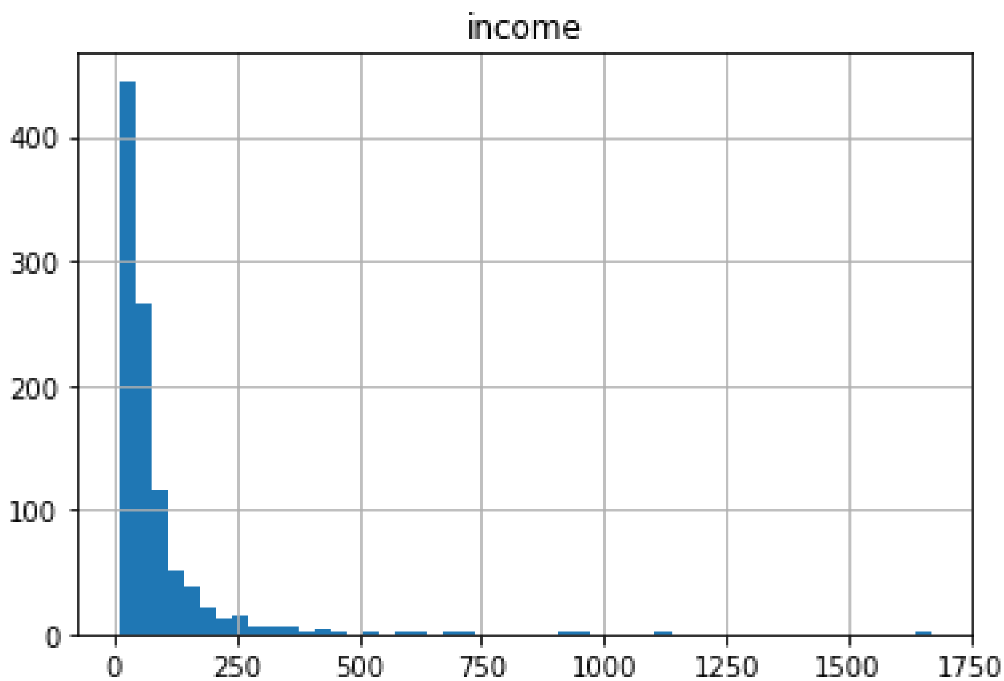
[44]:
```
df2.hist(column='income')
```

[44]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C5A38C640>]],
      dtype=object)
```



[45]:
```
df2.hist(column='income', bins=50)
```

[45]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C5A412310>]],
      dtype=object)
```



[46]:
```
df2.columns
```

[46]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

[47]:
```
#converting pandas data frame into numpys array
X = df2[['region', 'tenure','age', 'marital', 'address', 'income', 'ed',␣
 ↪'employ','retire', 'gender', 'reside']] .values  #.astype(float)
X[0:5]
```

[47]:
```
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.]])
```

[48]:
```
y = df2['custcat'].values
y[0:5]
```

[48]:
```
array([1, 4, 3, 1, 3], dtype=int64)
```

[49]:
```
#Data Standardization give data zero mean and unit variance, it is good␣
 ↪practice,
#especially for algorithms such as KNN which is based on distance of cases:
```

[50]:
```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

[50]:
```
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]])
```

[51]:
```
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=4)
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,)
```

[52]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[53]:
```
neigh = KNeighborsClassifier(n_neighbors=4)
```

[54]:
```
neigh.fit(X_train, y_train)
```

[54]: KNeighborsClassifier(n_neighbors=4)

[55]:
```
yhat = neigh.predict(X_test)
```

[56]:
```
from sklearn.metrics import accuracy_score
```

[57]:
```
accuracy_score(yhat, y_test)
```

[57]: 0.32

[58]:
```
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))
```

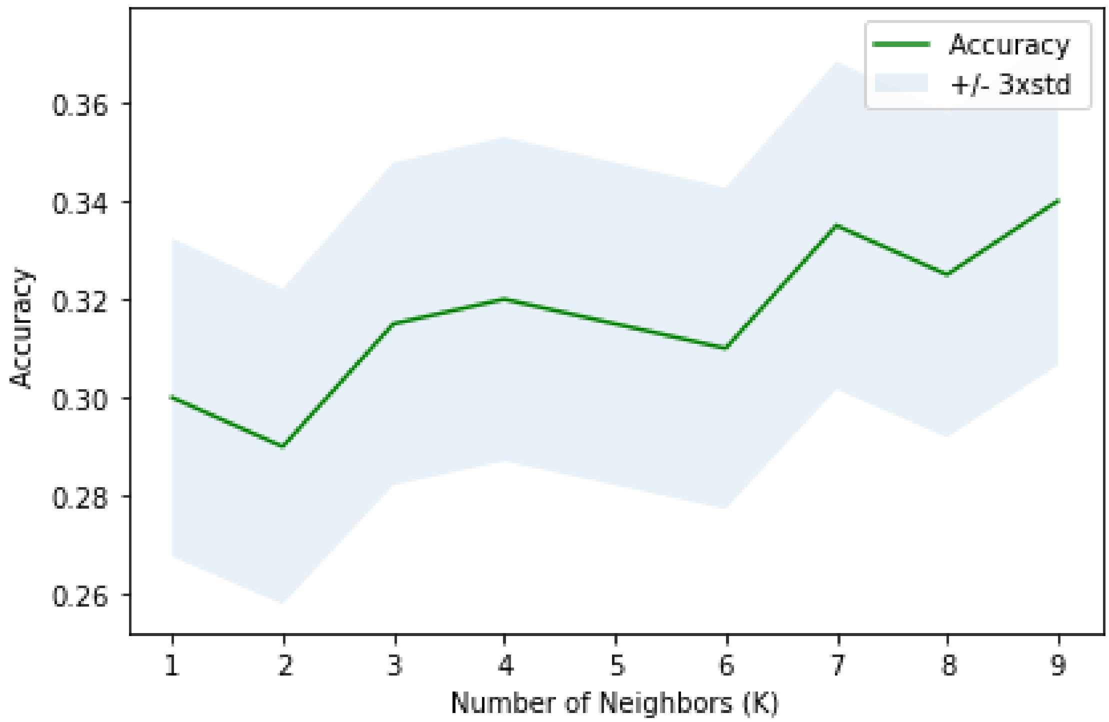
```
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

[59]:
```
#to see which k is the best
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
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
```

[59]: array([0.3 , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34])

[60]:
```
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.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



[61]:
```
#The best accuracy was with k=9
```

[62]:
```
#DECISION TREE
```

[99]:
```
from sklearn.tree import DecisionTreeClassifier
```

[65]:
```
df3 = pd.read_csv('drug200.csv')
df3.head()
```

[65]:
```
   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
```

[82]:
```python
x = df3[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
```

[83]:
```python
#cleaning categorical data
```

[84]:
```python
from sklearn import preprocessing

le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
x[:, 1] = le_sex.transform(x[:, 1])
```

[86]:
```python
le_bp = preprocessing.LabelEncoder()
le_bp.fit(['HIGH', 'LOW', 'NORMAL'])
x[:,2] = le_bp.transform(x[:,2])
```

[87]:
```python
le_cholestrol = preprocessing.LabelEncoder()
le_cholestrol.fit(['NORMAL', 'HIGH'])
x[:,3] = le_cholestrol.transform(x[:,3])
```

[92]:
```python
x[0:5]
```

[92]:
```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.113999999999999],
       [28, 0, 2, 0, 7.797999999999999],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

[93]:
```python
y = df3['Drug']
```

[94]:
```python

```

[94]:
```
0      drugY
1      drugC
2      drugC
3      drugX
4      drugY
       …
195    drugC
196    drugC
197    drugX
198    drugX
199    drugX
```

```
Name: Drug, Length: 200, dtype: object
```

[97]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=3)
```

[117]:
```
tree1 = DecisionTreeClassifier(criterion='entropy', max_depth=4)
tree1
```

[117]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

[118]:
```
tree1.fit(x_train, y_train)
```

[118]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

[119]:
```
prediction = tree1.predict(x_test)
```

[120]:
```
Accuracy = accuracy_score(prediction, y_test)
Accuracy
```

[120]:
```
1.0
```

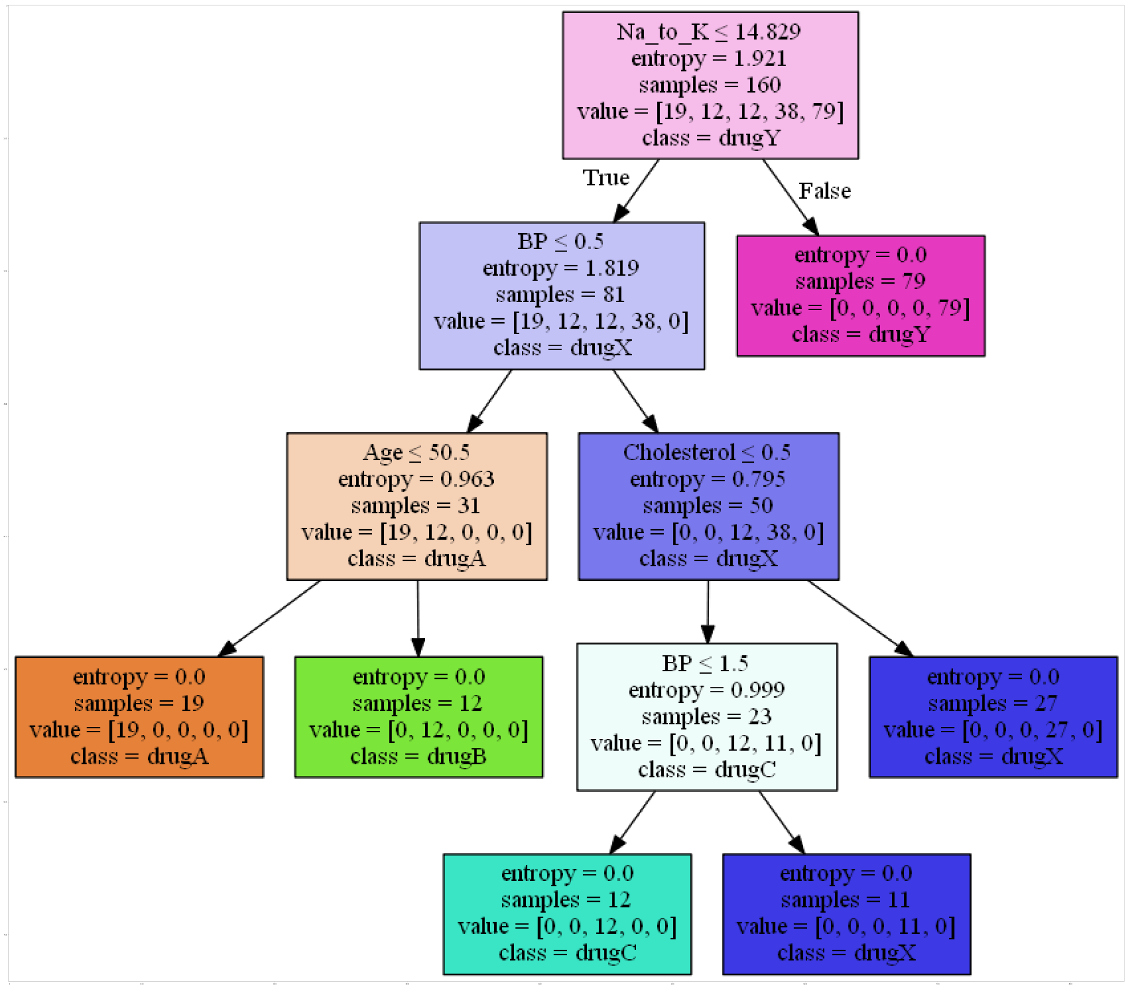
[121]:
```
import io
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

[ ]:
```
estimator = model.estimators_[5]
export_graphviz(estimator_limited,
                out_file='tree.dot',
                feature_names = iris.feature_names,
                class_names = iris.target_names,
                rounded = True, proportion = False,
                precision = 2, filled = True)
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])
from IPython.display import Image
Image(filename = 'tree.png')
```

[123]:
```
dot_data = io.StringIO()
filename = "drugtree.png"
featureNames = df3.columns[0:5]
targetNames = df3["Drug"].unique().tolist()
out=tree.export_graphviz(tree1,feature_names=featureNames, out_file=dot_data,␣
↪class_names= np.unique(y_train), filled=True, ␣
↪special_characters=True,rotate=False)
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')
```

[123]: <matplotlib.image.AxesImage at 0x17c5a8d9550>



[124]:
```
#LOGISTIC REGRESSION
```

[125]:
```
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
```

[126]:
```
df4 = pd.read_csv('ChurnData.csv')
```

[127]:
```
df4.head()
```

[127]:
```
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
0    11.0  33.0      7.0   136.0  5.0     5.0    0.0       1.0       1.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0
2    23.0  30.0      9.0    30.0  1.0     2.0    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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0

   longmon  …  pager  internet  callwait  confer  ebill  loglong  logtoll  \
0     4.40  …    1.0       0.0       1.0     1.0    0.0    1.482    3.033
1     9.45  …    0.0       0.0       0.0     0.0    0.0    2.246    3.240
2     6.30  …    0.0       0.0       0.0     1.0    0.0    1.841    3.240
3     6.05  …    1.0       1.0       1.0     1.0    1.0    1.800    3.807
4     7.10  …    0.0       0.0       1.0     1.0    0.0    1.960    3.091

   lninc  custcat  churn
0  4.913      4.0    1.0
1  3.497      1.0    1.0
2  3.401      3.0    0.0
3  4.331      4.0    0.0
4  4.382      3.0    0.0

[5 rows x 28 columns]
```

[131]:
```
df4 = df4[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',  ␣
 ↪'callcard', 'wireless','churn']]
df4['churn'] = df4['churn'].astype('int')
df4.head()
```

[131]:
```
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
0    11.0  33.0      7.0   136.0  5.0     5.0    0.0       1.0       1.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0
2    23.0  30.0      9.0    30.0  1.0     2.0    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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0

   churn
0      1
1      1
2      0
3      0
4      0
```

[134]:
```
X = df4[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']].values
X[0:5]
```

[134]:
```
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.]])
```

[136]:
```
X = np.asarray(df4[['tenure', 'age', 'address', 'income', 'ed', 'employ',␣
↪'equip']])
X[0:5]
```

[136]:
```
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.]])
```

[139]:
```
y = np.asarray(df4['churn'])
y[0:5]
```

[139]:
```
array([1, 1, 0, 0, 0])
```

[141]:
```
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

[141]:
```
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]])
```

[142]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=4)
```

[143]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

[144]:
```
LR = LogisticRegression()
```

[145]:
```
LR.fit(X_train, y_train)
```

[145]:
```
LogisticRegression()
```

[147]:
```
Yhat = LR.predict(X_test)
```

[163]:
```
y_test.shape
```

[163]:
```
(40,)
```

[148]:
```
#predict_proba returns estimates for all classes, ordered by the label of␣
↪classes.
# So, the first column is the probability of class 1, P(Y=1|X), and second␣
↪column is probability of class 0, P(Y=0|X)
```

[152]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

[152]:
```
array([[0.74658429, 0.25341571],
       [0.92677899, 0.07322101],
       [0.83445726, 0.16554274],
       [0.94596742, 0.05403258],
       [0.84351139, 0.15648861],
       [0.71452329, 0.28547671],
       [0.77085785, 0.22914215],
       [0.90956492, 0.09043508],
       [0.26142925, 0.73857075],
       [0.94907369, 0.05092631],
       [0.84772942, 0.15227058],
       [0.89315103, 0.10684897],
       [0.57506834, 0.42493166],
       [0.32555873, 0.67444127],
       [0.91995311, 0.08004689],
       [0.633071  , 0.366929  ],
       [0.6297197 , 0.3702803 ],
       [0.71293143, 0.28706857],
       [0.64068923, 0.35931077],
       [0.7794542 , 0.2205458 ],
       [0.91593448, 0.08406552],
       [0.64123809, 0.35876191],
       [0.96435248, 0.03564752],
       [0.55216187, 0.44783813],
       [0.62291087, 0.37708913],
       [0.97603043, 0.02396957],
       [0.6014112 , 0.3985888 ],
       [0.68062074, 0.31937926],
       [0.71779212, 0.28220788],
       [0.9820836 , 0.0179164 ],
       [0.96445529, 0.03554471],
       [0.765139  , 0.234861  ],
       [0.29422866, 0.70577134],
       [0.96537173, 0.03462827],
       [0.93653126, 0.06346874],
```

```
        [0.88756299, 0.11243701],
        [0.22268043, 0.77731957],
        [0.70018568, 0.29981432],
        [0.93940692, 0.06059308],
        [0.72116371, 0.27883629]])
```

[1]:
```
#SVM
```

[2]:
```
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
import matplotlib.pyplot as plt
%matplotlib inline
```

[5]:
```
df = pd.read_csv('cell_samples.csv')
df.tail()
```

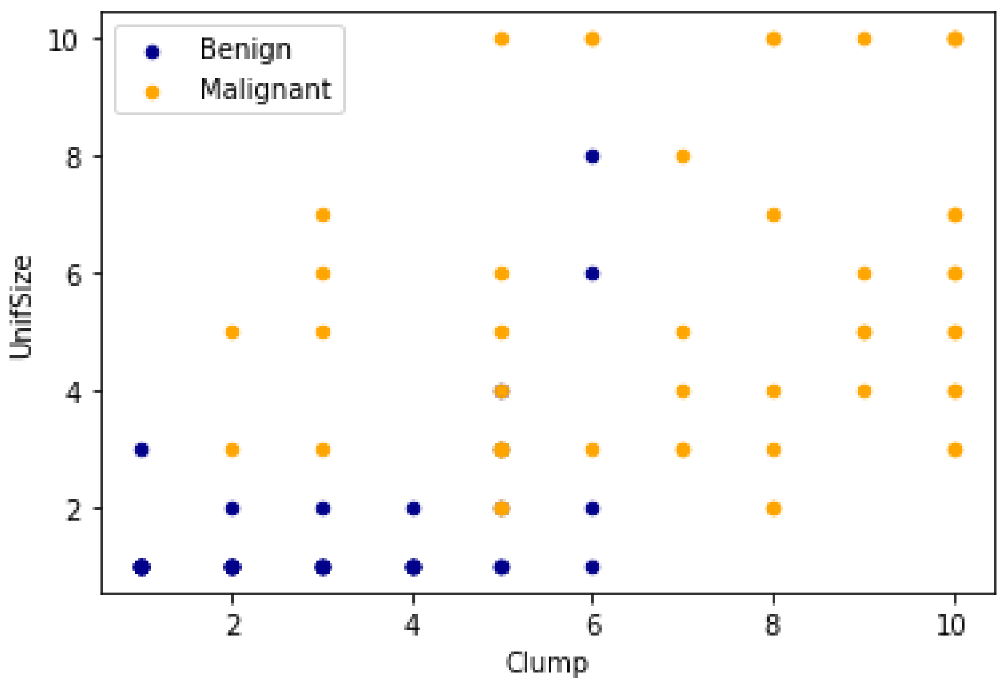
[5]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc
694	776715	3	1	1	1	3	2
695	841769	2	1	1	1	2	1
696	888820	5	10	10	3	7	3
697	897471	4	8	6	4	3	4
698	897471	4	8	8	5	4	5

	BlandChrom	NormNucl	Mit	Class
694	1	1	1	2
695	1	1	1	2
696	8	10	2	4
697	10	6	1	4
698	10	4	1	4

[6]:
```
#The attribute class has 2 and 4 where 2 means benign and 4 means malignant␣
 ↪type of cancer
```

[16]:
```
ax = df[df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize',␣
 ↪color='DarkBlue', label='Benign');
df[df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='UnifSize',␣
 ↪color='Orange', label='Malignant', ax=ax);
plt.show()
```



[17]:
```
df.dtypes
```

[17]:
```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc        object
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

[20]:
```
df = df[pd.to_numeric(df['BareNuc'], errors='coerce').notnull()]
df['BareNuc'] = df['BareNuc'].astype('int')
df.dtypes
```

[20]:
```
ID             int64
Clump          int64
UnifSize       int64
UnifShape      int64
MargAdh        int64
SingEpiSize    int64
```

```
BareNuc        int32
BlandChrom     int64
NormNucl       int64
Mit            int64
Class          int64
dtype: object
```

[29]:
```python
feature_df = df.drop(columns='Class')
X = np.asarray(feature_df)
```

[31]:
```python
X
```

[31]:
```
array([[1000025,       5,       1, …,       3,       1,       1],
       [1002945,       5,       4, …,       3,       2,       1],
       [1015425,       3,       1, …,       3,       1,       1],
       …,
       [ 888820,       5,      10, …,       8,      10,       2],
       [ 897471,       4,       8, …,      10,       6,       1],
       [ 897471,       4,       8, …,      10,       4,       1]],
      dtype=int64)
```

[32]:
```python
y = df['Class'].values
```

[34]:
```python
y
```

[34]:
```
array([2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 4, 4, 2, 2, 4, 2, 4, 4,
       2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 4, 4, 4, 4, 4, 2,
       4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 2, 4, 2, 4,
       4, 2, 2, 4, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 2, 4, 2, 4,
       4, 4, 2, 2, 2, 4, 2, 2, 2, 2, 4, 4, 4, 2, 4, 2, 4, 2, 2, 2, 4, 2,
       2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 4, 4, 2, 2, 4, 2, 2,
       4, 4, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 4, 4, 4, 2, 4, 2, 4, 2, 2,
       2, 4, 4, 2, 4, 4, 4, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2,
       2, 4, 4, 2, 2, 2, 4, 4, 2, 4, 4, 4, 2, 2, 4, 2, 2, 4, 4, 4, 4, 2,
       4, 4, 2, 4, 4, 4, 2, 4, 2, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 4, 4, 2,
       2, 4, 2, 4, 4, 4, 2, 2, 2, 2, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 2,
       4, 4, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4, 4, 4, 2, 4, 4, 2, 2, 4, 4, 2,
       2, 4, 4, 2, 4, 2, 4, 4, 2, 2, 4, 2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 4,
       2, 4, 2, 2, 4, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 2, 2, 4, 4, 2, 2, 2,
       4, 2, 2, 2, 4, 4, 2, 2, 2, 4, 2, 2, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2,
       4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2,
       2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 4, 2, 4, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2,
       2, 2, 4, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2,
       4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 4, 4, 4, 2, 4, 2, 4, 2, 2, 2, 2, 2,
```

```
       2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 4, 2,
       2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       4, 2, 2, 4, 4, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 4,
       2, 4, 2, 4, 4, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 4,
       2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2,
       2, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 4,
       4], dtype=int64)
```

[35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=4)
```

[36]:
```
from sklearn import svm
```

[38]:
```
model = svm.SVC(kernel='rbf')
```

[39]:
```
model.fit(X_train, y_train)
```

[39]:
```
SVC()
```

[40]:
```
yhat = model.predict(X_test)
```

[53]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, yhat)
score
```

[53]:
```
0.656934306569343
```

[50]:
```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

[50]:
```
0.5209170712884659
```

[54]:
```
model = svm.SVC(kernel='linear')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
```

[55]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, yhat)
score
```

[55]:
```
0.6861313868613139
```

[1]:
```
#K-means using random generated datasets
```

[6]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

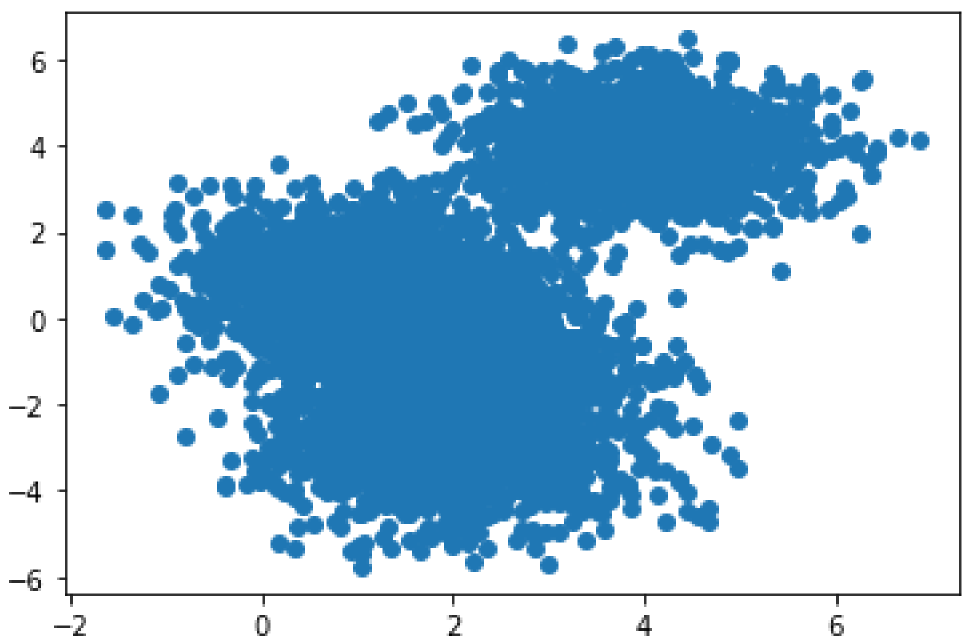
```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143:
FutureWarning: The sklearn.datasets.samples_generator module is  deprecated in
version 0.22 and will be removed in version 0.24. The corresponding classes /
functions should instead be imported from sklearn.datasets. Anything that cannot
be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[9]:
```
np.random.seed(0)
```

[12]:
```
X,y =make_blobs(n_samples = 5000, centers = [[4,4], [2,-1], [2,-3], [1,1]],␣
↪cluster_std=0.9)
```

[20]:
```
plt.scatter(X[:,0], X[:,1], marker='o')
```

[20]: <matplotlib.collections.PathCollection at 0x2bec1c87250>



[23]:
```
k_means = KMeans(init='k-means++', n_clusters=4, n_init=12)
```

[24]:
```
k_means.fit(X)
```

[24]: KMeans(n_clusters=4, n_init=12)

[25]:
```
k_means_labels = k_means.labels_
k_means_labels
```

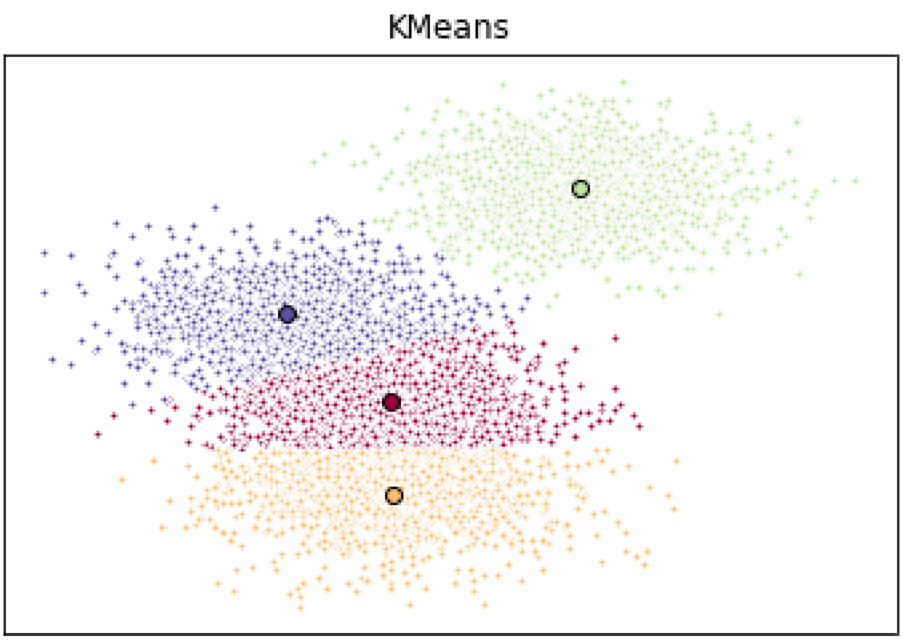
[25]: array([0, 1, 1, …, 2, 0, 3])

[26]:
```
k_means_cluster_center = k_means.cluster_centers_
k_means_cluster_center
```

[26]:
```
array([[ 1.99386362, -0.98210637],
       [ 2.00767516, -3.16352369],
       [ 3.97423467,  3.98310553],
       [ 0.90429725,  1.07253251]])
```

[28]:
```
# 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 poitns 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_center[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, ␣
 ↪markeredgecolor='k', markersize=6)
```

```python
# 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()
```



[29]:
```python
#k-means using dataset
```

[30]:
```python
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

[30]:
```
   Customer Id  Age  Edu  Years Employed  Income  Card Debt  Other Debt  \
0            1   41    2               6      19      0.124       1.073
1            2   47    1              26     100      4.582       8.218
2            3   33    2              10      57      6.111       5.802
3            4   29    2               4      19      0.681       0.516
4            5   47    1              31     253      9.308       8.908
```

```
   Defaulted Address  DebtIncomeRatio
0        0.0  NBA001              6.3
1        0.0  NBA021             12.8
2        1.0  NBA013             20.9
3        0.0  NBA009              6.3
4        0.0  NBA008              7.2
```

[32]:
```python
cust_df.drop('Address', axis=1, inplace=True)
```

[33]:
```python
cust_df
```

[33]:
```
     Customer Id  Age  Edu  Years Employed  Income  Card Debt  Other Debt  \
0              1   41    2               6      19      0.124       1.073
1              2   47    1              26     100      4.582       8.218
2              3   33    2              10      57      6.111       5.802
3              4   29    2               4      19      0.681       0.516
4              5   47    1              31     253      9.308       8.908
..           …   …    …               …       …         …           …
845          846   27    1               5      26      0.548       1.220
846          847   28    2               7      34      0.359       2.021
847          848   25    4               0      18      2.802       3.210
848          849   32    1              12      28      0.116       0.696
849          850   52    1              16      64      1.866       3.638

     Defaulted  DebtIncomeRatio
0          0.0              6.3
1          0.0             12.8
2          1.0             20.9
3          0.0              6.3
4          0.0              7.2
..         …              …
845        NaN              6.8
846        0.0              7.0
847        1.0             33.4
848        0.0              2.9
849        0.0              8.6

[850 rows x 9 columns]
```

[35]:
```python
from sklearn.preprocessing import StandardScaler
X = cust_df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

[35]:
```
array([[ 0.74291541,  0.31212243, -0.37878978, …, -0.59048916,
        -0.52379654, -0.57652509],
```

```
[ 1.48949049, -0.76634938,  2.5737211 , …,  1.51296181,
 -0.52379654,  0.39138677],
[-0.25251804,  0.31212243,  0.2117124 , …,  0.80170393,
  1.90913822,  1.59755385],
…,
[-1.24795149,  2.46906604, -1.26454304, …,  0.03863257,
  1.90913822,  3.45892281],
[-0.37694723, -0.76634938,  0.50696349, …, -0.70147601,
 -0.52379654, -1.08281745],
[ 2.1116364 , -0.76634938,  1.09746566, …,  0.16463355,
 -0.52379654, -0.2340332 ]])
```

[36]:

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

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

[38]:

```python
cust_df["Clus_km"] = labels
cust_df.head(5)
```

[38]:

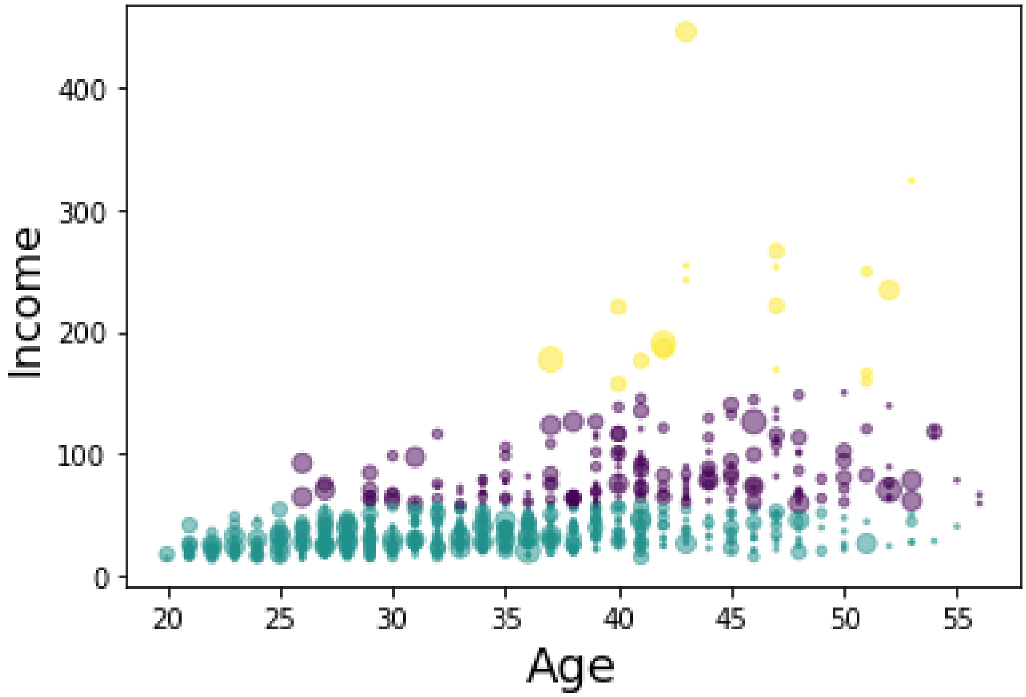
	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	\
0	1	41	2	6	19	0.124	1.073	
1	2	47	1	26	100	4.582	8.218	

```
2              3   33    2              10     57      6.111       5.802
3              4   29    2               4     19      0.681       0.516
4              5   47    1              31    253      9.308       8.908

   Defaulted  DebtIncomeRatio  Clus_km
0        0.0              6.3        1
1        0.0             12.8        0
2        1.0             20.9        1
3        0.0              6.3        1
4        0.0              7.2        2
```

[40]:
```python
cust_df.groupby('Clus_km').mean()
```

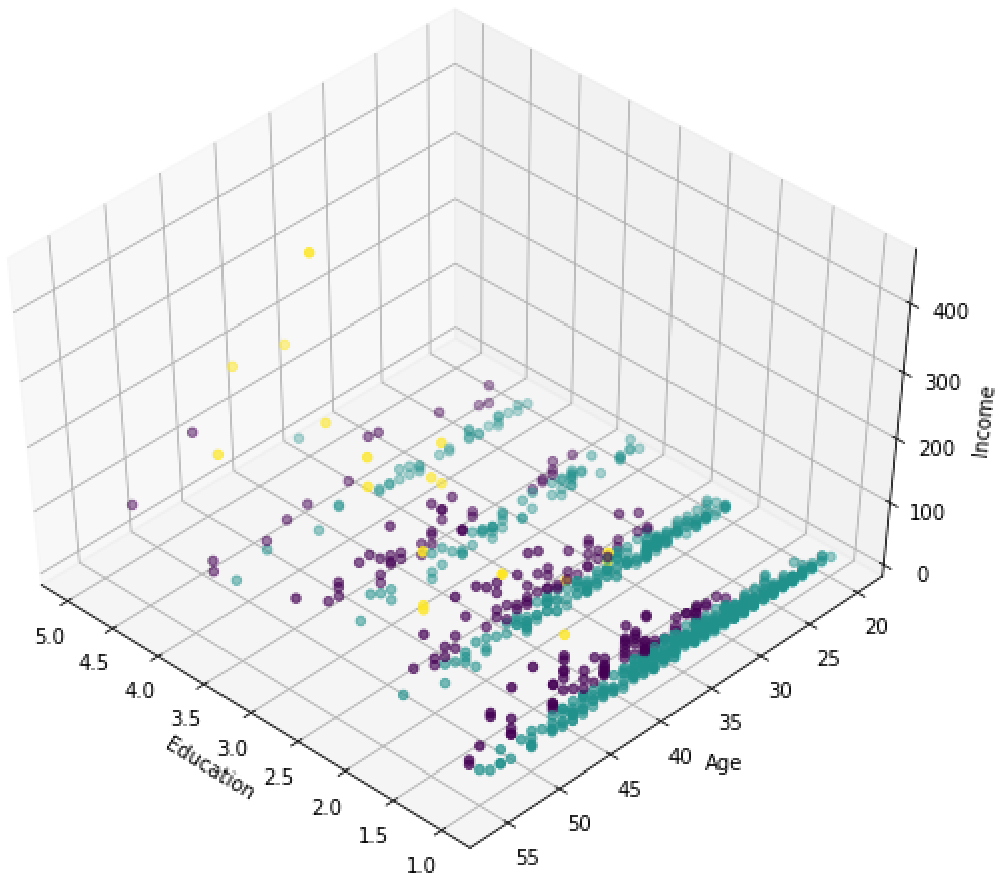
[40]:
```
         Customer Id        Age       Edu  Years Employed      Income  \
Clus_km
0         403.780220  41.368132  1.961538       15.252747   84.076923
1         432.006154  32.967692  1.613846        6.389231   31.204615
2         410.166667  45.388889  2.666667       19.555556  227.166667

         Card Debt  Other Debt  Defaulted  DebtIncomeRatio
Clus_km
0         3.114412    5.770352   0.172414        10.725824
1         1.032711    2.108345   0.284658        10.095385
2         5.678444   10.907167   0.285714         7.322222
```

[41]:
```python
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()
```



[42]:
```
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))
```

[42]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x2bec34bc250>
```



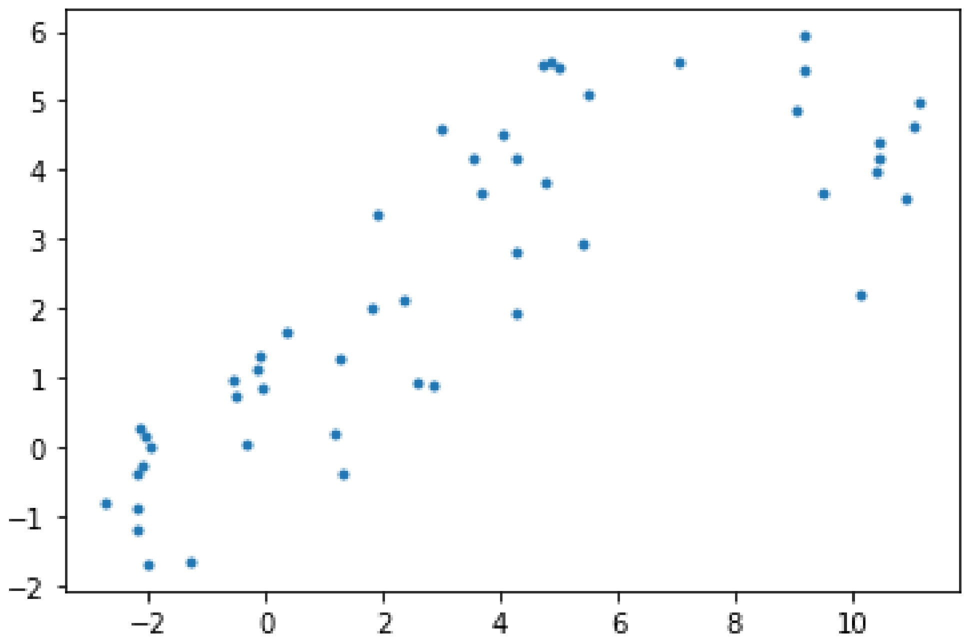
[1]:
```
#hierarchial clustering using random data
```

[4]:
```
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.samples_generator import make_blobs
%matplotlib inline
```

[5]:
```
X1,y1 = make_blobs(n_samples=50, centers=[[4,4],[-2,-1],[1,1],[10,4]],␣
 ↪cluster_std=0.9)
```

[6]:
```
plt.scatter(X1[:,0], X1[:,1], marker='.')
```

[6]: <matplotlib.collections.PathCollection at 0x1feeca44fd0>



[7]:
```
agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
```

[8]:
```
agglom.fit(X1,y1)
```

[8]:
```
AgglomerativeClustering(linkage='average', n_clusters=4)
```

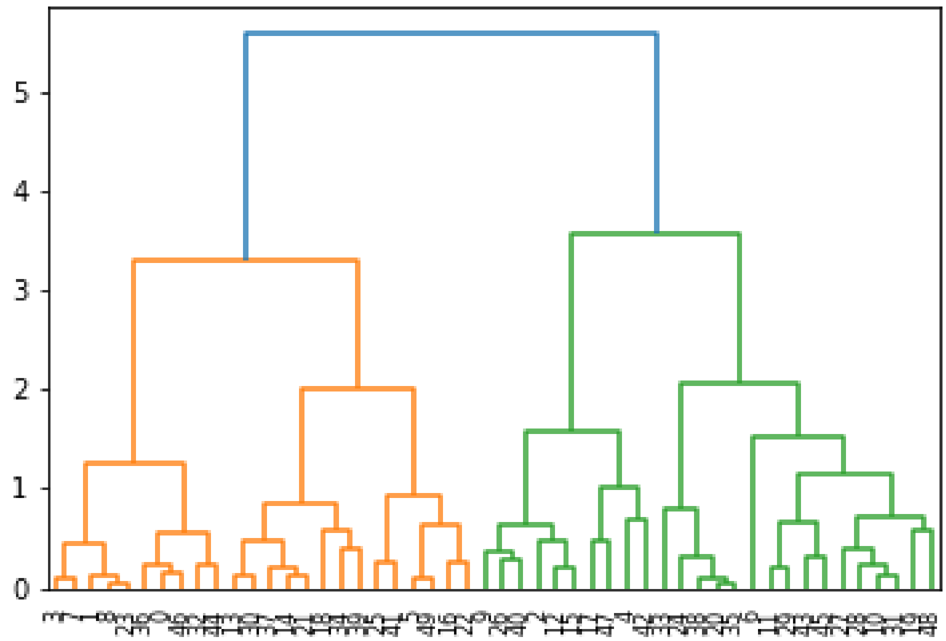
[10]:
```
dist_matrix = distance_matrix(X1,X1)
print(dist_matrix)
```

```
[[0.         0.12247938 0.74991261 … 0.76668181 1.06019761 0.46335653]
 [0.12247938 0.         0.62989098 … 0.65519682 0.9547588  0.36737556]
 [0.74991261 0.62989098 0.         … 0.15898383 0.42069497 0.36995815]
 …
 [0.76668181 0.65519682 0.15898383 … 0.         0.3101169  0.32386818]
 [1.06019761 0.9547588  0.42069497 … 0.3101169  0.         0.59944437]
 [0.46335653 0.36737556 0.36995815 … 0.32386818 0.59944437 0.        ]]
```

[11]:
```
Z = hierarchy.linkage(dist_matrix, 'complete')
```

```
<ipython-input-11-3814b774a052>:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  Z = hierarchy.linkage(dist_matrix, 'complete')
```

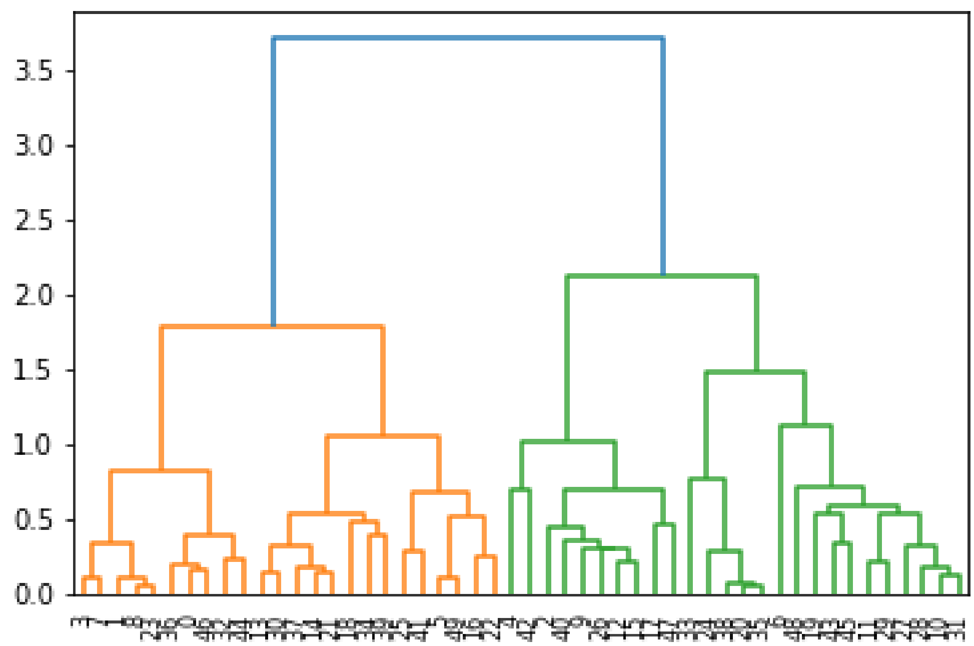
[14]:
```
dendro = hierarchy.dendrogram(Z)
```



[17]:
```
Z = hierarchy.linkage(dist_matrix, 'centroid')
```

```
<ipython-input-17-a08c4ecedfa6>:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  Z = hierarchy.linkage(dist_matrix, 'centroid')
```

[18]:
```
dendro = hierarchy.dendrogram(Z)
```



[19]:
```
#hierarchial clustering on real dataset
```

[27]:
```
df= pd.read_csv('cars_clus.csv')
df.head()
```

[27]:
```
  manufact    model   sales  resale   type   price engine_s horsepow wheelbas  \
0    Acura  Integra  16.919  16.360  0.000  21.500    1.800  140.000  101.200
1    Acura       TL  39.384  19.875  0.000  28.400    3.200  225.000  108.100
2    Acura       CL  14.114  18.225  0.000  $null$    3.200  225.000  106.900
3    Acura       RL   8.588  29.725  0.000  42.000    3.500  210.000  114.600
4     Audi       A4  20.397  22.255  0.000  23.990    1.800  150.000  102.600

    width   length curb_wgt fuel_cap     mpg lnsales  partition
0  67.300  172.400    2.639   13.200  28.000   2.828        0.0
1  70.300  192.900    3.517   17.200  25.000   3.673        0.0
2  70.600  192.000    3.470   17.200  26.000   2.647        0.0
3  71.400  196.600    3.850   18.000  22.000   2.150        0.0
4  68.200  178.000    2.998   16.400  27.000   3.015        0.0
```

[29]:
```
print ("Shape of dataset before cleaning: ", df.size)
df[[ 'sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']] = df[['sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']].apply(pd.to_numeric, errors='coerce')
df = df.dropna()
```

```
df = df.reset_index(drop=True)
print ("Shape of dataset after cleaning: ", df.size)
df.head(5)
```

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

[29]:

```
  manufact    model   sales  resale  type  price  engine_s  horsepow  \
0    Acura  Integra  16.919  16.360   0.0  21.50       1.8     140.0
1    Acura       TL  39.384  19.875   0.0  28.40       3.2     225.0
2    Acura       RL   8.588  29.725   0.0  42.00       3.5     210.0
3     Audi       A4  20.397  22.255   0.0  23.99       1.8     150.0
4     Audi       A6  18.780  23.555   0.0  33.95       2.8     200.0

   wheelbas  width  length  curb_wgt  fuel_cap   mpg  lnsales  partition
0     101.2   67.3   172.4     2.639      13.2  28.0    2.828        0.0
1     108.1   70.3   192.9     3.517      17.2  25.0    3.673        0.0
2     114.6   71.4   196.6     3.850      18.0  22.0    2.150        0.0
3     102.6   68.2   178.0     2.998      16.4  27.0    3.015        0.0
4     108.7   76.1   192.0     3.561      18.5  22.0    2.933        0.0
```

[30]:

```
featureset = df[['engine_s',  'horsepow', 'wheelbas', 'width', 'length',␣
↪'curb_wgt', 'fuel_cap', 'mpg']]
```

[31]:

```
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]
```

[31]:

```
array([[0.11428571, 0.21518987, 0.18655098, 0.28143713, 0.30625832,
        0.2310559 , 0.13364055, 0.43333333],
       [0.31428571, 0.43037975, 0.3362256 , 0.46107784, 0.5792277 ,
        0.50372671, 0.31797235, 0.33333333],
       [0.35714286, 0.39240506, 0.47722343, 0.52694611, 0.62849534,
        0.60714286, 0.35483871, 0.23333333],
       [0.11428571, 0.24050633, 0.21691974, 0.33532934, 0.38082557,
        0.34254658, 0.28110599, 0.4       ],
       [0.25714286, 0.36708861, 0.34924078, 0.80838323, 0.56724368,
        0.5173913 , 0.37788018, 0.23333333]])
```

[33]:

```
import scipy
leng = feature_mtx.shape[0]
D = np.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])
```

[34]:
```
Z = hierarchy.linkage(D, 'complete')
```

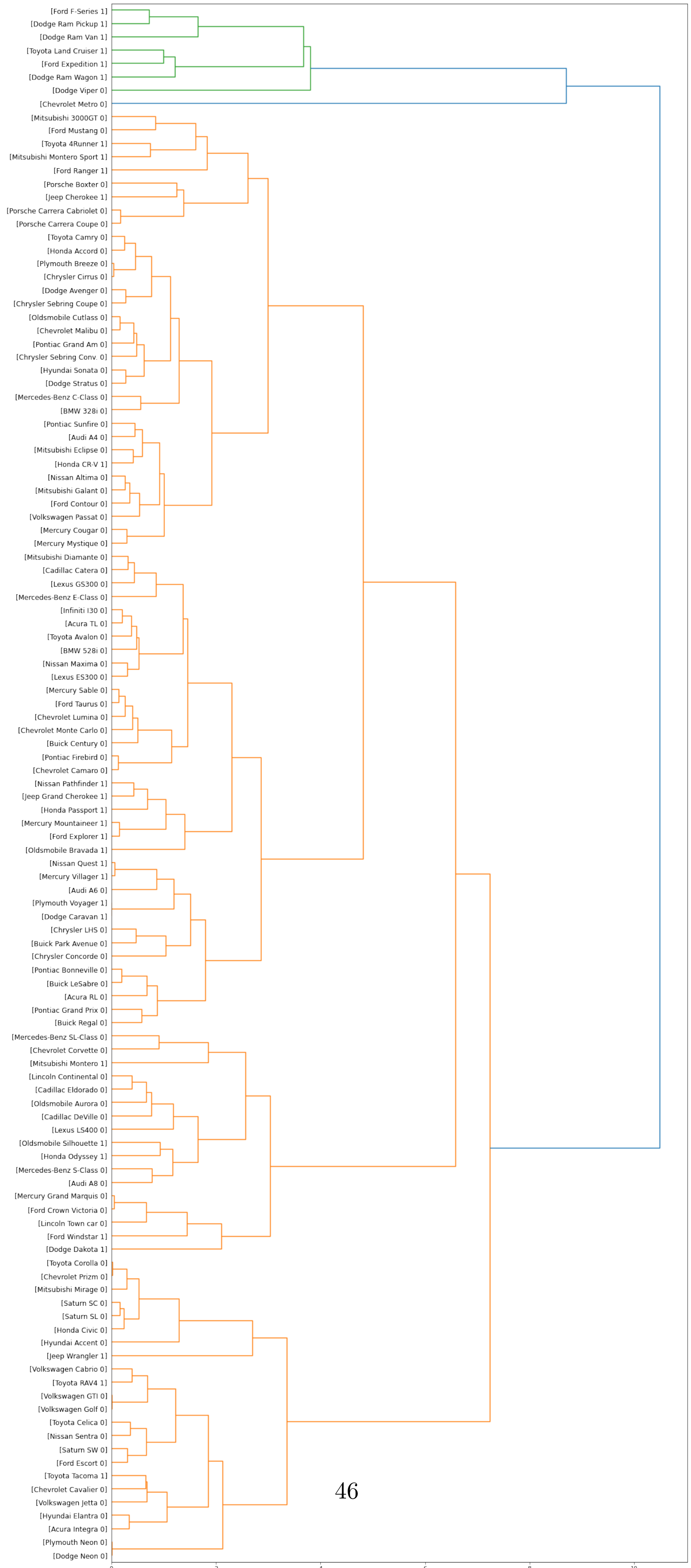
```
<ipython-input-34-f7fd5c287128>:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  Z = hierarchy.linkage(D, 'complete')
```

[35]:
```
#Hierarchical clustering does not require a pre-specified number of clusters.
#However, in some applications we want a partition of disjoint clusters just as␣
↪in flat clustering.
#So you can use a cutting line:
```

[36]:
```
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

[36]:
```
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)
```

[45]:
```
import pylab
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (df['manufact'][id], df['model'][id],␣
↪int(float(df['type'][id])) )

dendro = hierarchy.dendrogram(Z,  leaf_label_func=llf, leaf_rotation=0,␣
↪leaf_font_size =12, orientation = 'right')
```


[Ford F-Series 1]
[Dodge Ram Pickup 1]
[Dodge Ram Van 1]
[Toyota Land Cruiser 1]
[Ford Expedition 1]
[Dodge Ram Wagon 1]
[Dodge Viper 0]
[Chevrolet Metro 0]
[Mitsubishi 3000GT 0]
[Ford Mustang 0]
[Toyota 4Runner 1]
[Mitsubishi Montero Sport 1]
[Ford Ranger 1]
[Porsche Boxter 0]
[Jeep Cherokee 1]
[Porsche Carrera Cabriolet 0]
[Porsche Carrera Coupe 0]
[Toyota Camry 0]
[Honda Accord 0]
[Plymouth Breeze 0]
[Chrysler Cirrus 0]
[Dodge Avenger 0]
[Chrysler Sebring Coupe 0]
[Oldsmobile Cutlass 0]
[Chevrolet Malibu 0]
[Pontiac Grand Am 0]
[Chrysler Sebring Conv. 0]
[Hyundai Sonata 0]
[Dodge Stratus 0]
[Mercedes-Benz C-Class 0]
[BMW 328i 0]
[Pontiac Sunfire 0]
[Audi A4 0]
[Mitsubishi Eclipse 0]
[Honda CR-V 1]
[Nissan Altima 0]
[Mitsubishi Galant 0]
[Ford Contour 0]
[Volkswagen Passat 0]
[Mercury Cougar 0]
[Mercury Mystique 0]
[Mitsubishi Diamante 0]
[Cadillac Catera 0]
[Lexus GS300 0]
[Mercedes-Benz E-Class 0]
[Infiniti I30 0]
[Acura TL 0]
[Toyota Avalon 0]
[BMW 528i 0]
[Nissan Maxima 0]
[Lexus ES300 0]
[Mercury Sable 0]
[Ford Taurus 0]
[Chevrolet Lumina 0]
[Chevrolet Monte Carlo 0]
[Buick Century 0]
[Pontiac Firebird 0]
[Chevrolet Camaro 0]
[Nissan Pathfinder 1]
[Jeep Grand Cherokee 1]
[Honda Passport 1]
[Mercury Mountaineer 1]
[Ford Explorer 1]
[Oldsmobile Bravada 1]
[Nissan Quest 1]
[Mercury Villager 1]
[Audi A6 0]
[Plymouth Voyager 1]
[Dodge Caravan 1]
[Chrysler LHS 0]
[Buick Park Avenue 0]
[Chrysler Concorde 0]
[Pontiac Bonneville 0]
[Buick LeSabre 0]
[Acura RL 0]
[Pontiac Grand Prix 0]
[Buick Regal 0]
[Mercedes-Benz SL-Class 0]
[Chevrolet Corvette 0]
[Mitsubishi Montero 1]
[Lincoln Continental 0]
[Cadillac Eldorado 0]
[Oldsmobile Aurora 0]
[Cadillac DeVille 0]
[Lexus LS400 0]
[Oldsmobile Silhouette 1]
[Honda Odyssey 1]
[Mercedes-Benz S-Class 0]
[Audi A8 0]
[Mercury Grand Marquis 0]
[Ford Crown Victoria 0]
[Lincoln Town car 0]
[Ford Windstar 1]
[Dodge Dakota 1]
[Toyota Corolla 0]
[Chevrolet Prizm 0]
[Mitsubishi Mirage 0]
[Saturn SC 0]
[Saturn SL 0]
[Honda Civic 0]
[Hyundai Accent 0]
[Jeep Wrangler 1]
[Volkswagen Cabrio 0]
[Toyota RAV4 1]
[Volkswagen GTI 0]
[Volkswagen Golf 0]
[Toyota Celica 0]
[Nissan Sentra 0]
[Saturn SW 0]
[Ford Escort 0]
[Toyota Tacoma 1]
[Chevrolet Cavalier 0]
[Volkswagen Jetta 0]
[Hyundai Elantra 0]
[Acura Integra 0]
[Plymouth Neon 0]
[Dodge Neon 0]
0
2
4
6
8
10

[1]:
```
#DBSCAN
```

[3]:
```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.datasets.samples_generator import make_blobs
import matplotlib.pyplot as plt
%matplotlib inline
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143:
FutureWarning: The sklearn.datasets.samples_generator module is  deprecated in
version 0.22 and will be removed in version 0.24. The corresponding classes /
functions should instead be imported from sklearn.datasets. Anything that cannot
be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[4]:
```
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
```

[5]:
```
X, y = createDataPoints([[4,3], [2,-1], [-1,4]] , 1500, 0.5)
```

[6]:
```
db = DBSCAN(eps=0.3, min_samples=7)
```

[7]:
```
db.fit(X)
```

[7]:
```
DBSCAN(eps=0.3, min_samples=7)
```

[8]:
```
labels=db.labels_
labels
```

[8]:
```
array([0, 1, 2, …, 2, 0, 1], dtype=int64)
```

[10]:
```
#distinguish outliers
core_sample_mask = np.zeros_like(labels, dtype=bool)
core_sample_mask[db.core_sample_indices_]=True
core_sample_mask
```

[10]:
```
array([ True,  True,  True, …,  True,  True,  True])
```

[11]:
```python
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_
```

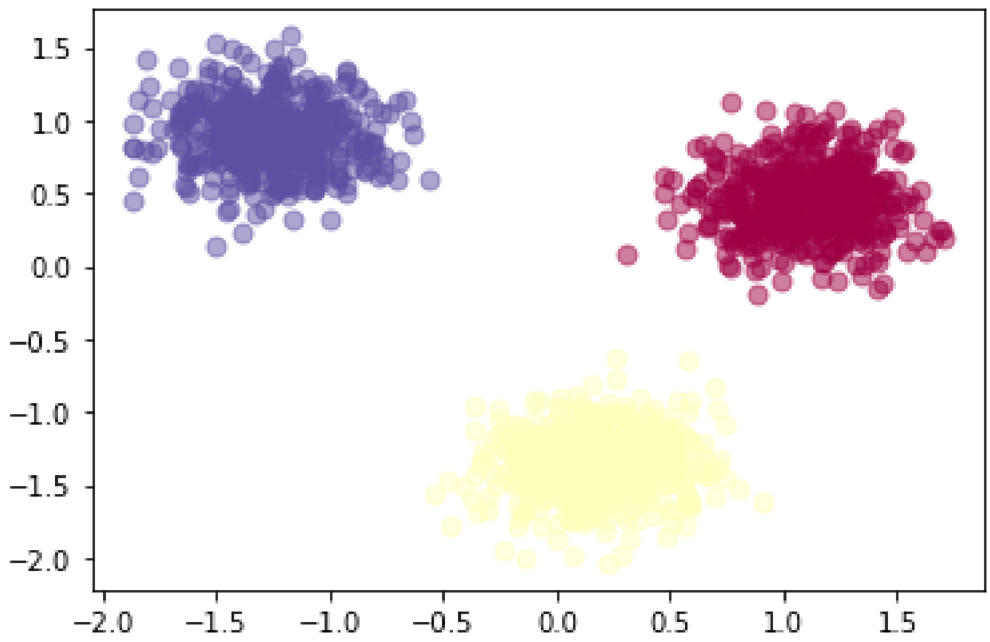
[11]: 3

[12]:
```python
unique_label = set(labels)
unique_label
```

[12]: {0, 1, 2}

[14]:
```python
# Create colors for the clusters.
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_label)))
```

[18]:
```python
# Plot the points with colors
for k, col in zip(unique_label, 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_sample_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_sample_mask]
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)
```



[19]:
```
#using dataset
```

[20]:
```
import pandas as pd
import numpy as np
```

[23]:
```
pdf = pd.read_csv('weather-stations20140101-20141231.csv')
```

[24]:
```
pdf.head()
```

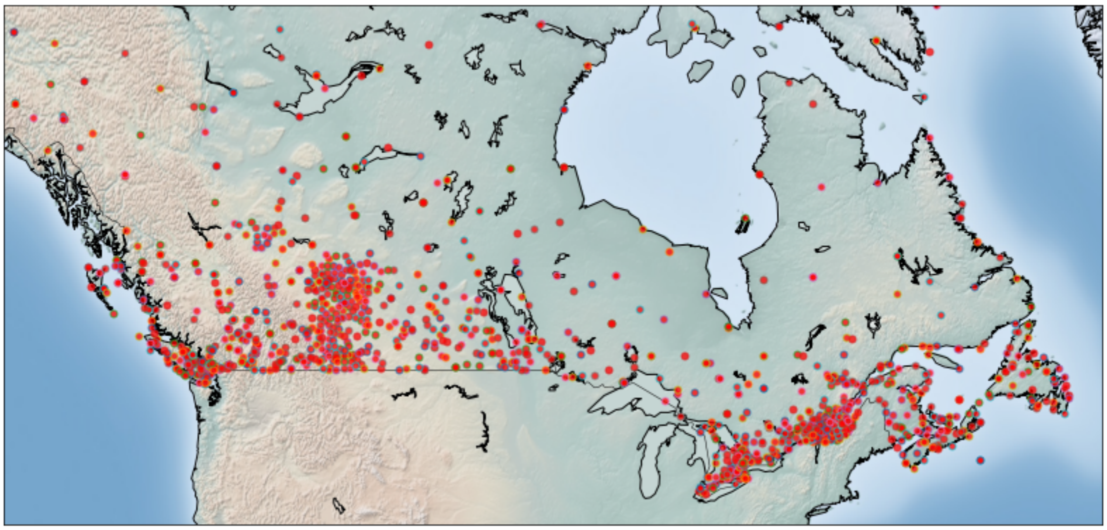
[24]:
```
                Stn_Name     Lat     Long Prov   Tm  DwTm    D    Tx  DwTx  \
0              CHEMAINUS  48.935 -123.742   BC  8.2   0.0  NaN  13.5   0.0
1  COWICHAN LAKE FORESTRY  48.824 -124.133   BC  7.0   0.0  3.0  15.0   0.0
2          LAKE COWICHAN  48.829 -124.052   BC  6.8  13.0  2.8  16.0   9.0
3       DISCOVERY ISLAND  48.425 -123.226   BC  NaN   NaN  NaN  12.5   0.0
4     DUNCAN KELVIN CREEK  48.735 -123.728   BC  7.7   2.0  3.4  14.5   2.0

    Tn  …  DwP    P%N  S_G    Pd  BS  DwBS  BS%    HDD  CDD   Stn_No
0  1.0  …  0.0    NaN  0.0  12.0 NaN   NaN  NaN  273.3  0.0  1011500
1 -3.0  …  0.0  104.0  0.0  12.0 NaN   NaN  NaN  307.0  0.0  1012040
2 -2.5  …  9.0    NaN  NaN  11.0 NaN   NaN  NaN  168.1  0.0  1012055
3  NaN  …  NaN    NaN  NaN   NaN NaN   NaN  NaN    NaN  NaN  1012475
4 -1.0  …  2.0    NaN  NaN  11.0 NaN   NaN  NaN  267.7  0.0  1012573

[5 rows x 25 columns]
```

[25]:
```
pdf = pdf[pd.notnull(pdf["Tm"])]
pdf = pdf.reset_index(drop=True)
pdf.head(5)
```

[25]:
```
                 Stn_Name     Lat     Long Prov   Tm  DwTm    D    Tx  DwTx  \
0               CHEMAINUS  48.935 -123.742   BC  8.2   0.0  NaN  13.5   0.0
1  COWICHAN LAKE FORESTRY  48.824 -124.133   BC  7.0   0.0  3.0  15.0   0.0
2           LAKE COWICHAN  48.829 -124.052   BC  6.8  13.0  2.8  16.0   9.0
3      DUNCAN KELVIN CREEK  48.735 -123.728   BC  7.7   2.0  3.4  14.5   2.0
4       ESQUIMALT HARBOUR  48.432 -123.439   BC  8.8   0.0  NaN  13.1   0.0

    Tn  …  DwP    P%N  S_G    Pd  BS  DwBS  BS%    HDD  CDD   Stn_No
0  1.0  …  0.0    NaN  0.0  12.0 NaN   NaN  NaN  273.3  0.0  1011500
1 -3.0  …  0.0  104.0  0.0  12.0 NaN   NaN  NaN  307.0  0.0  1012040
2 -2.5  …  9.0    NaN  NaN  11.0 NaN   NaN  NaN  168.1  0.0  1012055
3 -1.0  …  2.0    NaN  NaN  11.0 NaN   NaN  NaN  267.7  0.0  1012573
4  1.9  …  8.0    NaN  NaN  12.0 NaN   NaN  NaN  258.6  0.0  1012710

[5 rows x 25 columns]
```

[26]:
```
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'] < ulat)]

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

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, alpha = 0.75)
#plt.text(x,y,stn)
plt.show()
```



[27]:
```
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)
```

[27]:

```
                 Stn_Name    Tx   Tm  Clus_Db
0               CHEMAINUS  13.5  8.2        0
1  COWICHAN LAKE FORESTRY  15.0  7.0        0
2           LAKE COWICHAN  16.0  6.8        0
3     DUNCAN KELVIN CREEK  14.5  7.7        0
4       ESQUIMALT HARBOUR  13.1  8.8        0
```

[28]:

```
set(labels)
```

[28]: {-1, 0, 1, 2, 3, 4}

[29]:

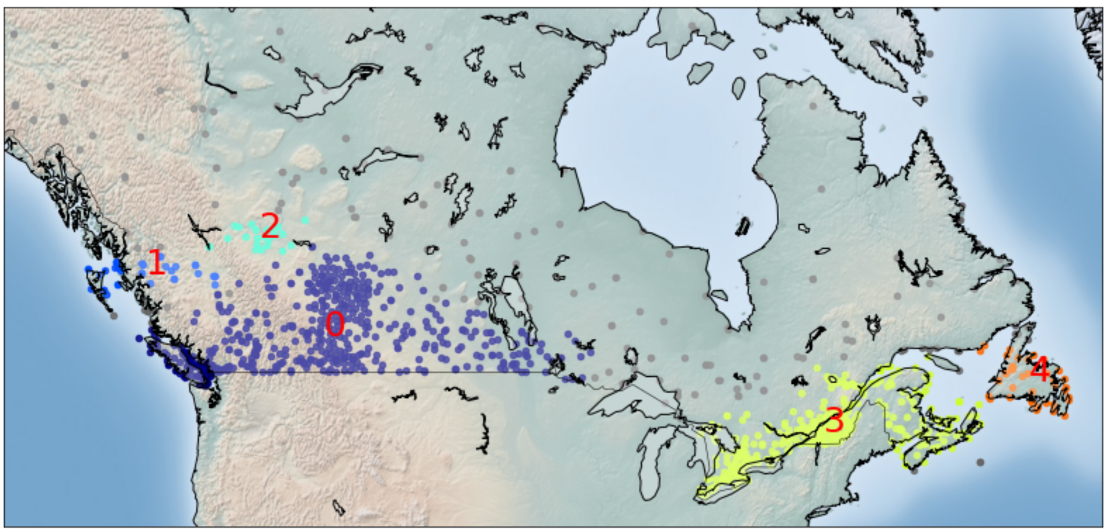
```
#visualization of clusters based on location
```

[30]:

```
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 (llcrnrlat)
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and␣
 ↪latitude (urcrnrlat)

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 = 0.85)
    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)))
```

```
Cluster 0, Avg Temp: -5.538747553816051
Cluster 1, Avg Temp: 1.9526315789473685
Cluster 2, Avg Temp: -9.195652173913045
Cluster 3, Avg Temp: -15.300833333333333
Cluster 4, Avg Temp: -7.769047619047619
```



[ ]:
```
#Clustering of stations based on their location, mean, max, and min Temperature
```

[31]:
```
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)
```

[31]:

	Stn_Name	Tx	Tm	Clus_Db
0	CHEMAINUS	13.5	8.2	0
1	COWICHAN LAKE FORESTRY	15.0	7.0	0
2	LAKE COWICHAN	16.0	6.8	0
3	DUNCAN KELVIN CREEK	14.5	7.7	0
4	ESQUIMALT HARBOUR	13.1	8.8	0

[32]:

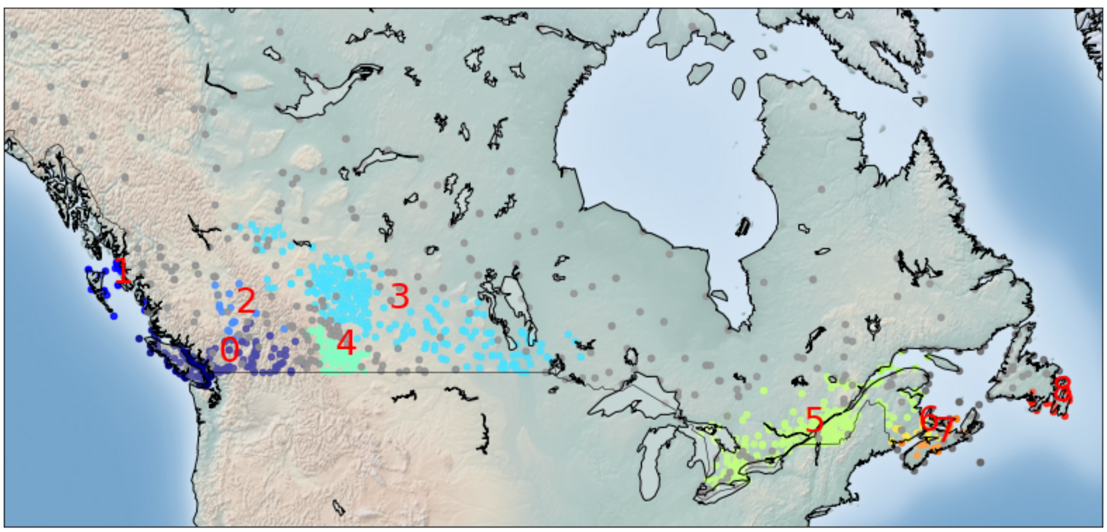
```
#visualization based on loxation and temperature
```

[33]:

```
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 (llcrnrlat)
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and␣
 ↪latitude (urcrnrlat)

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 = 0.85)
    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)))
```

```
Cluster 0, Avg Temp: 6.2211920529801334
Cluster 1, Avg Temp: 6.790000000000001
Cluster 2, Avg Temp: -0.49411764705882355
Cluster 3, Avg Temp: -13.877209302325586
Cluster 4, Avg Temp: -4.186274509803922
Cluster 5, Avg Temp: -16.301503759398482
Cluster 6, Avg Temp: -13.599999999999998
Cluster 7, Avg Temp: -9.753333333333334
Cluster 8, Avg Temp: -4.258333333333334
```



[ ]:

```

```