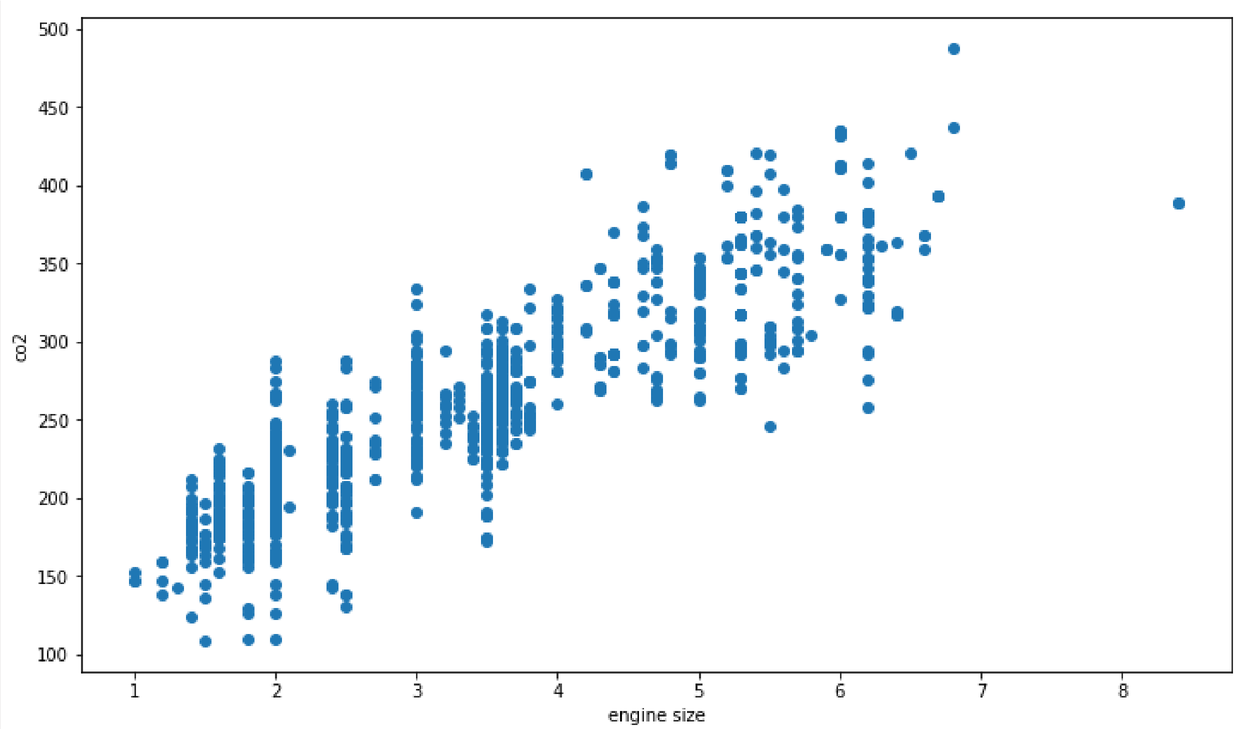
Linear regression

Simple linear regression

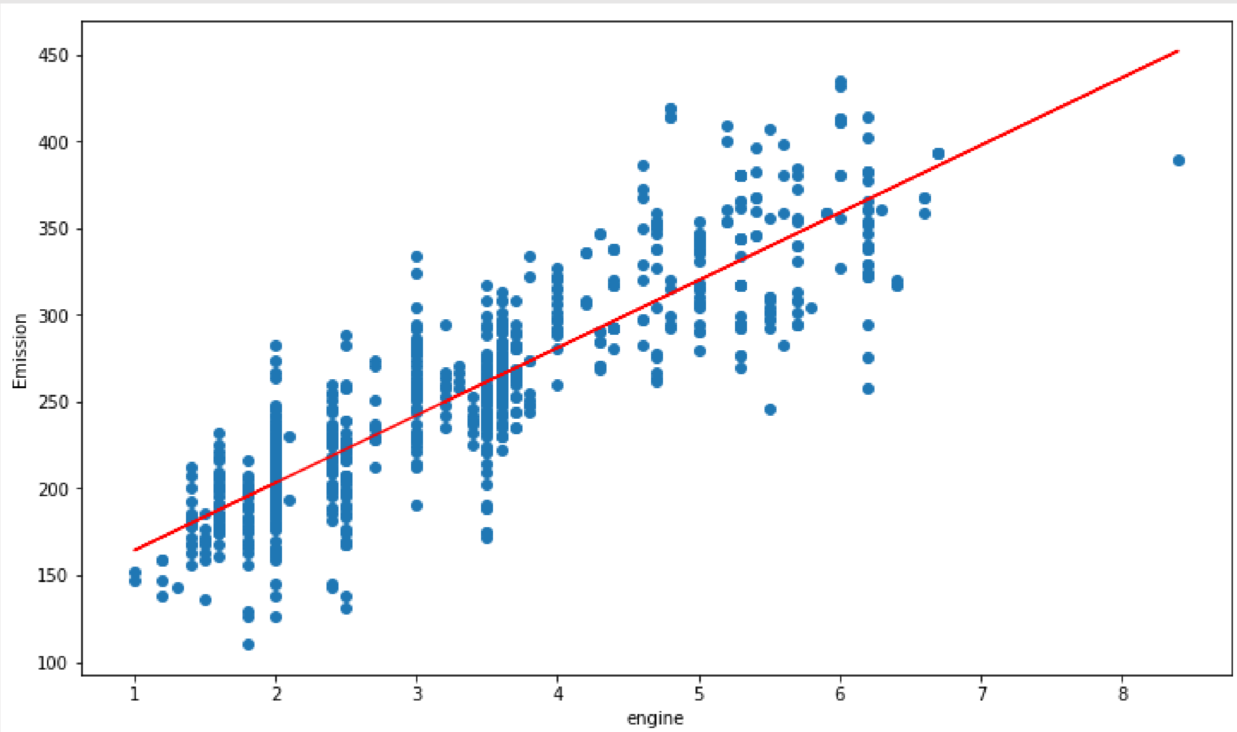
Code:

```
import matplotlib.pyplot as plt

import pandas as pd

import pylab as pl

import numpy as np

from sklearn import linear_model

from sklearn.metrics import r2_score


df=pd.read_csv('Fuel.csv')

df.head()


df.describe()# count, mean,min,25%,50%,75%

values = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]


# plot the relationship between engine size and co2 emission

plt.figure(figsize=(12,7))

plt.scatter(values['ENGINESIZE'],values['CO2EMISSIONS'])

plt.xlabel('engine size')

plt.ylabel('co2')
```




```
#plot the relatiohsip between cylinder and co2 emission
plt.figure(figsize=(12,7))
plt.scatter(values['CYLINDERS'],values['CO2EMISSIONS'])
plt.xlabel('cylinder size')
plt.ylabel('co2')


# get train and test data, train: 80%, test: 20%
msk=np.random.rand(len(values))<0.8
train=values[msk]
test=values[~msk]# ~ works for array not list


# linear regression model
linear=linear_model.LinearRegression()
train_x=np.array(train['ENGINESIZE']).reshape(-1,1)
train_y=np.array(train['CO2EMISSIONS']).reshape(-1,1)
linear.fit(train_x,train_y) # fit(x,y)
```

```
print('the coef is {0}, the intercept is {1}'.format(linear.coef_,linear.intercept_))


# plot predicted data

plt.figure(figsize=(12,7))

plt.scatter(train['ENGINESIZE'],train['CO2EMISSIONS'])

plt.plot(train_x,linear.coef_[0][0]*train_x + linear.intercept_[0],color='r')

plt.xlabel('engine')

plt.ylabel('Emission')
```



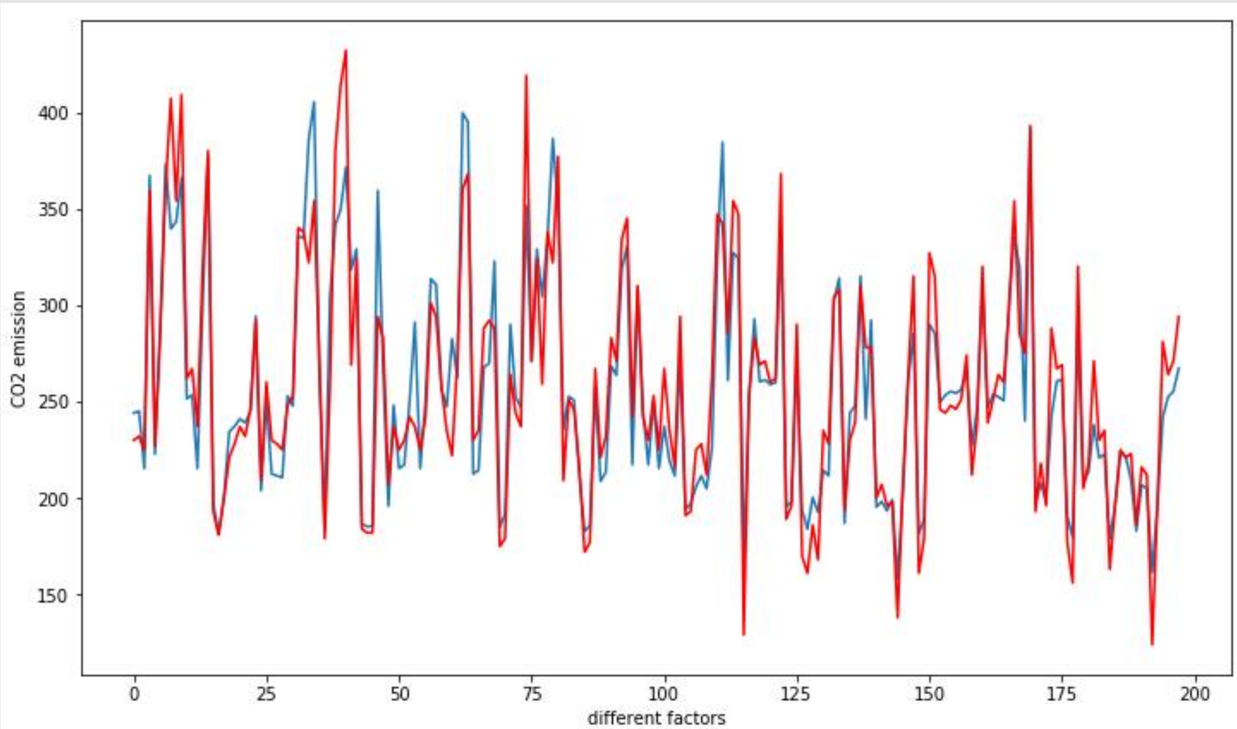

```
# test the value


test_x=np.array(test['ENGINESIZE']).reshape(-1,1)

test_y=np.array(test['CO2EMISSIONS']).reshape(-1,1)

predict_y=linear.predict(test_x)


print('The mean of square error is {}'.format(np.mean((predict_y-test_y)**2)))

print('R2 is {}'.format(r2_score(test_y,predict_y))) # R2 the bigger the better
```

Multiple linear regression

Code:

```
# import modules

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn import linear_model


data=pd.read_csv('Fuel.csv')

data.head()


cdata=data[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCO
NSUMPTION_COMB','CO2EMISSIONS']]


# generate training and testing data: 80% for train

msk=np.random.rand(len(data))<0.8

train=cdata[msk]

test=cdata[~msk]


# use linear regression model

regr=linear_model.LinearRegression()

x=np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])

y=np.asanyarray(train[['CO2EMISSIONS']])

regr.fit(x,y)

print('the coefficient is: {}'.format(regr.coef_))
```

```
plt.figure(figsize=(12,7))
y_pred=regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
test_x=np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
test_y=np.asanyarray(test[['CO2EMISSIONS']])
plt.plot(y_pred)
plt.plot(test_y,color='r')
plt.xlabel('different factors')
plt.ylabel('CO2 emission')
```



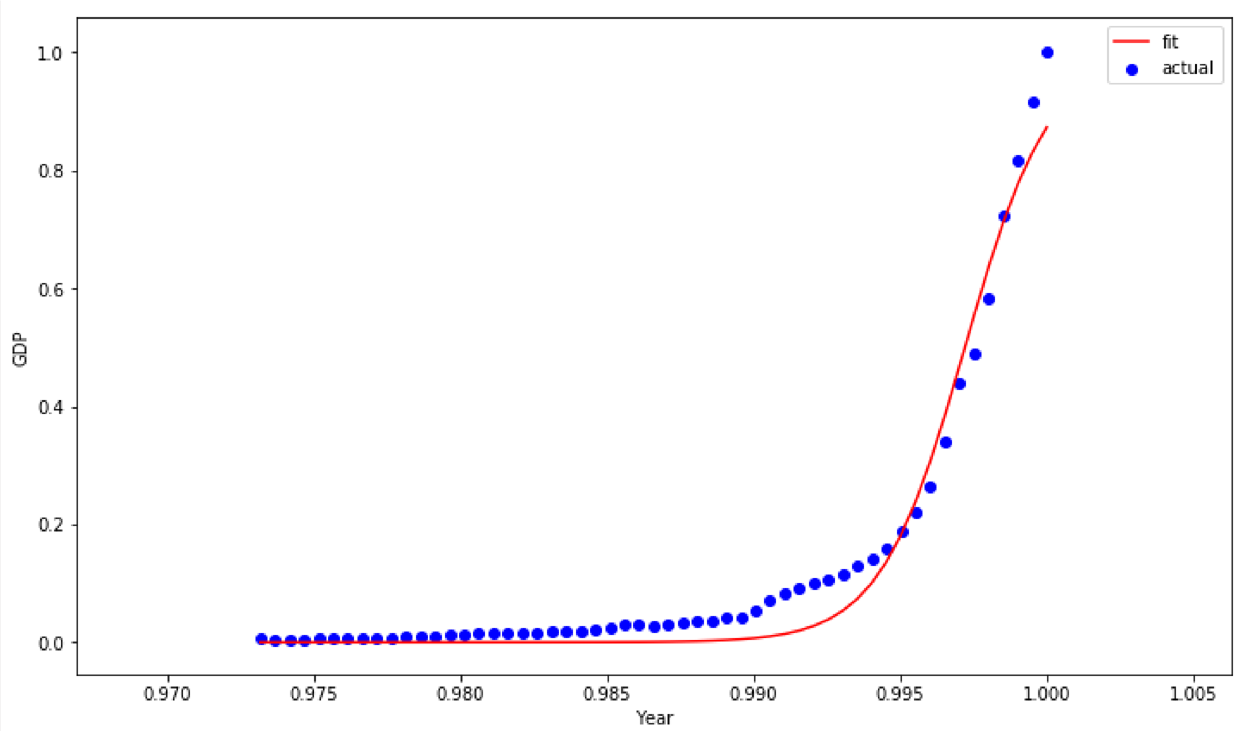

```
# evaluate the model
print('mean of error square is {}'.format(np.mean((y_pred-test_y)**2)))
print('score is {}'.format(regr.score(test_x,test_y)))
```

Nonlinear regression

code:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd

df=pd.read_csv('china_gdp.csv')
plt.plot(df['Year'],df['Value'])
plt.xlabel('Years')
plt.ylabel('GDP')

x=df['Year'].values
y=df['Value'].values

# choose a function and get the parameters
# in this case, logistic funcion is a good choice, but nomalization is required
def f(x,b1,b2):
    y=1/(1+np.exp(-b1*(x-b2)))
    return y
x_norm=x/max(x)
y_norm=y/max(y)
popt,pcov=curve_fit(f,x_norm,y_norm)
print('the parameters are: ',popt)
# plot the result
plt.figure(figsize=(12,7))
plt.scatter(x_norm,y_norm,label='actual',color='b')
y_p=f(x_norm,*popt)
plt.plot(x_norm,y_p,label='fit',color='r')
plt.legend(loc='best')
plt.xlabel('Year')
plt.ylabel('GDP')
```



Classification

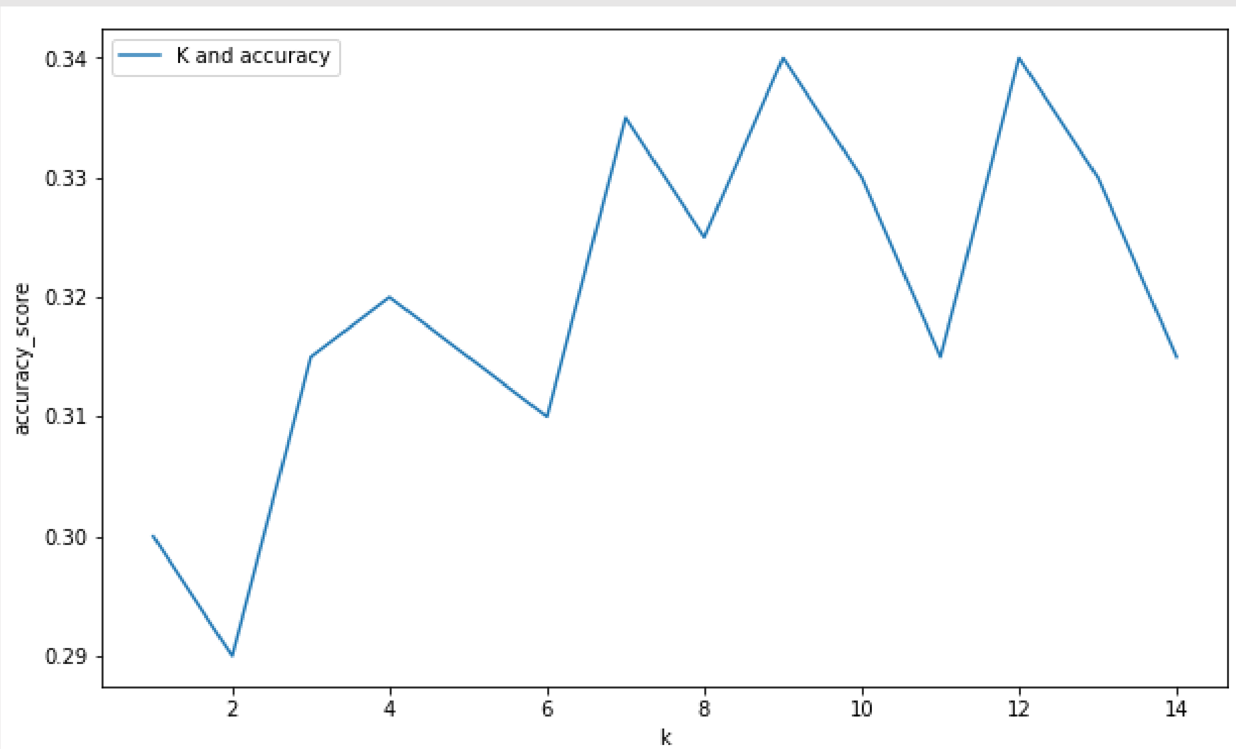
Knn

Code:

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 22:24:12 2018

@author: hejia
"""
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

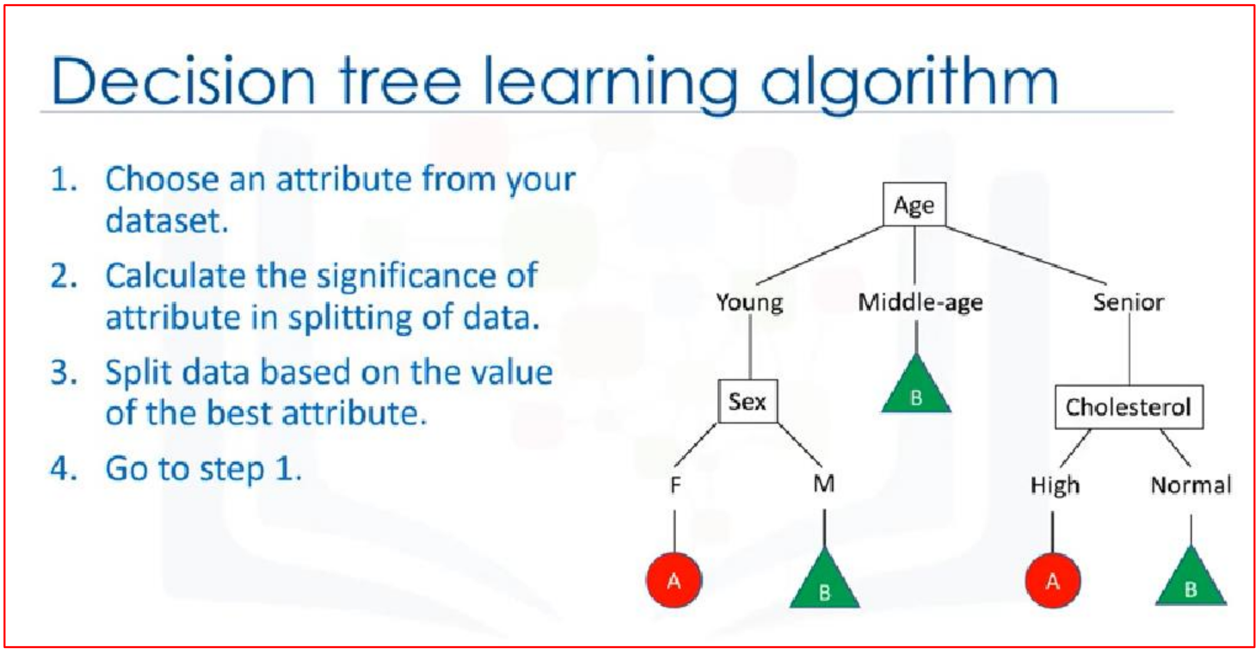
```
from sklearn import metrics
# import data
df=pd.read_csv('C:\\Users\\hejia\\Documents\\python\\machine learning\\teleCust1000t.csv')
all_columns=df.columns
# series method value_counts()
df['gender'].value_counts()
# use hist to see distribution
plt.hist(df['income'])
for column in all_columns:
    plt.figure(figsize=(10,6))
    plt.hist(df[column],label=column)
    plt.legend(loc='best')

# convert Pandas data frame to Numpy array
X=df[all_columns[0:-1]].values
y=df[all_columns[-1]].values

# Normalize data
from sklearn import preprocessing
X=preprocessing.StandardScaler().fit(X).transform(X.astype(float))

# train and test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=4)

#train and predict
k=4
knn=KNeighborsClassifier(k)
neigh=knn.fit(X_train,y_train)
y_pred=neigh.predict(X_test)
```

```
# evaluation
accuracy=metrics.accuracy_score(y_test,y_pred)

# evaluate the relationship between k and accuracy
accuracy=[]
for k in range(1,15):
    knn=KNeighborsClassifier(k)
    y_pred=knn.fit(X_train,y_train).predict(X_test)
    accuracy.append(metrics.accuracy_score(y_test,y_pred))
plt.figure(figsize=(10,6))
plt.plot(range(1,15),accuracy,label='K and accuracy')
plt.xlabel('k')
plt.ylabel('accuracy_score')
plt.legend(loc='best')
```



Decision trees

Logic:



Code:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 16 20:04:58 2018

@author: hejia
"""


import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
# import data
df=pd.read_csv('drug200.csv')
```

```
columns=df.columns
X=df[columns[:-1]].values
y=df[columns[-1]]

# transfer text values to numerical
sex_code=preprocessing.LabelEncoder()
sex_code.fit(['F','M'])
X[:,1]=sex_code.transform(X[:,1])

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

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

X_trainset, X_testset, y_trainset, y_testset=train_test_split(X,y,test_size=0.3,random_state=3)



# model with decision tree
drugTree=DecisionTreeClassifier(criterion='entropy',max_depth=4)
drugTree.fit(X_trainset,y_trainset)
predTree=drugTree.predict(X_testset)

# evaluation
print('decision tree accuracy: ',metrics.accuracy_score(y_testset,predTree))
```

Logistic regression

Code:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 17 05:09:50 2018

@author: hejia
"""


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

# import data
df=pd.read_csv('ChurnData.csv')

columns=['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',   'callcard', 'wireless','churn']
df=df[columns]
df['churn']=df['churn'].astype('int') # convert the data type
X=df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']].values
y=df['churn'].values

#preprocess the data
X=preprocessing.StandardScaler().fit(X).transform(X)
X_train,X_test, y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=4)
```

```
logreg=LogisticRegression().fit(X_train,y_train)
y_pred=logreg.predict(X_test)
```

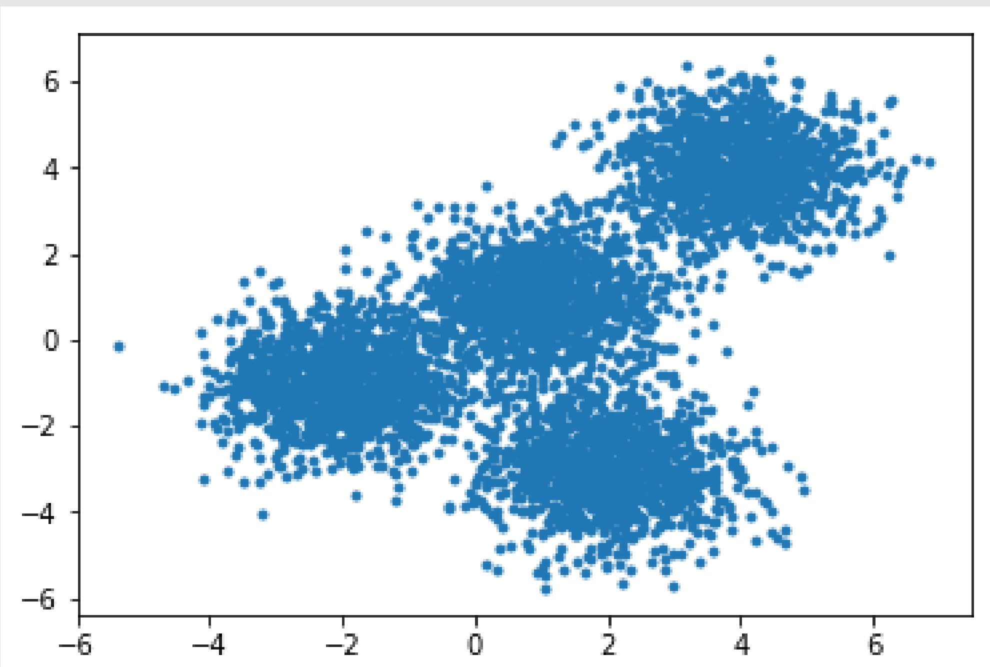
Cluster

K means

Code:

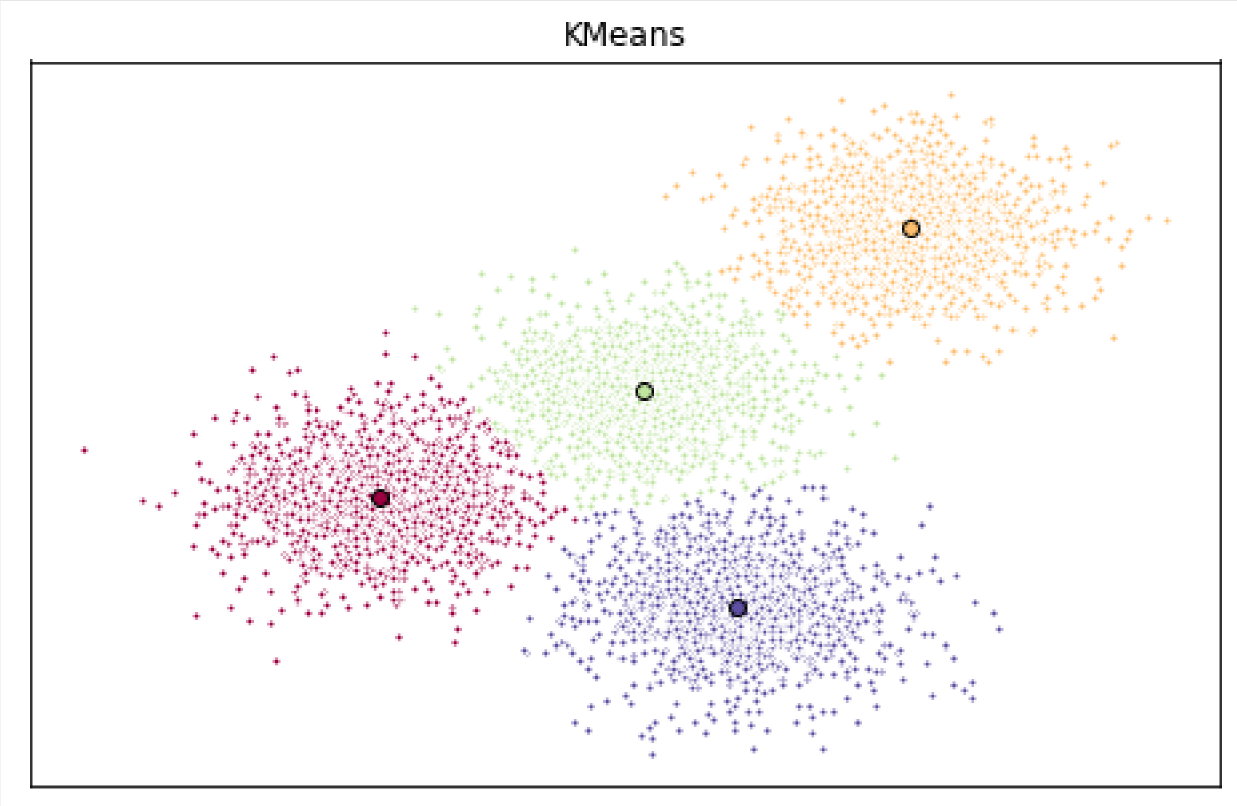
```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 17 20:23:03 2018

@author: hejia
"""

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs

# generate source data
np.random.seed(0)
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_std=0.9)
plt.scatter(X[:,0],X[:,1],marker='.')

# set up K means
k_means=KMeans(n_clusters=4,n_init=12)
k_means.fit(X)
k_means_labels=k_means.labels_
k_means_labels
```

```
k_means_cluster_centers=k_means.cluster_centers_


# plot the data

fig=plt.figure(figsize=(8,5))

colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels))))

ax=fig.add_subplot(1,1,1)


for k,col in zip(range(0,4),colors):

    my_members=(k_means_labels==k)

    cluster_center=k_means_cluster_centers[k]

    # select for the points belonging to cluster 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,  markeredgecolor='k',
markersize=6)

# Title of the plot

ax.set_title('KMeans')
```



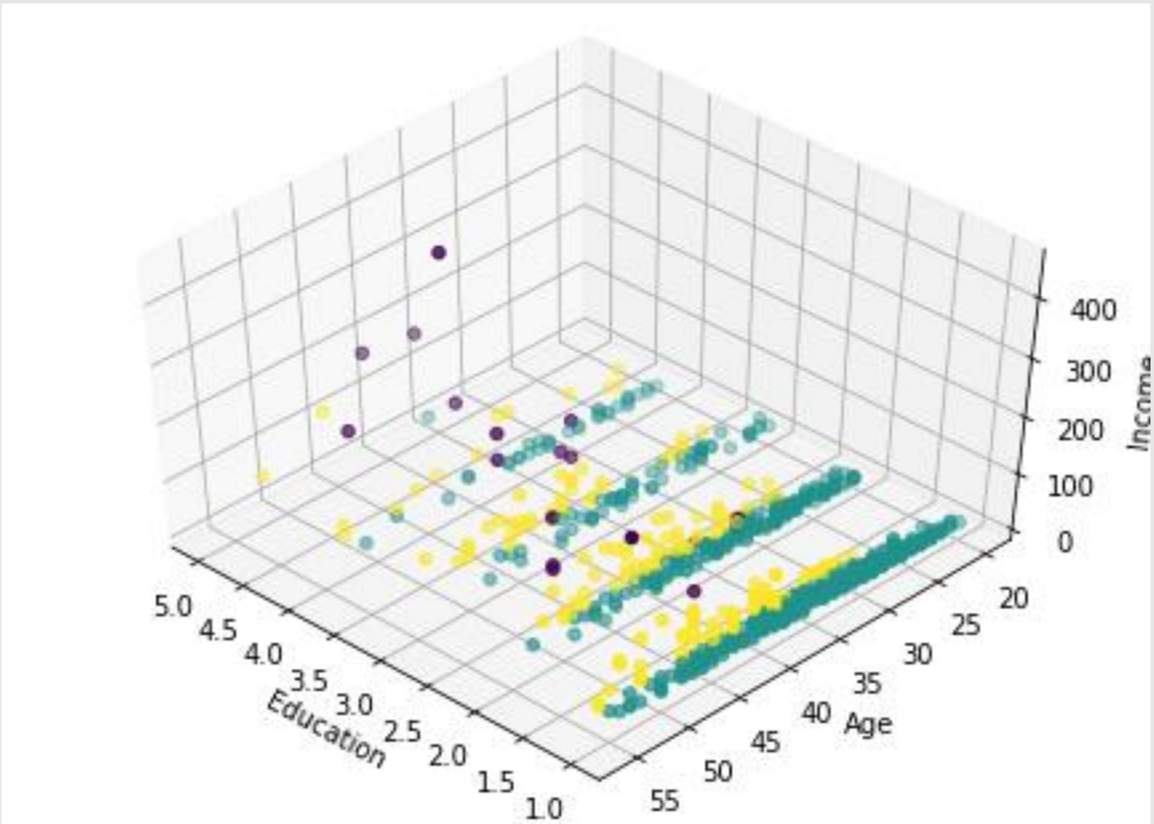
```
# Remove x-axis ticks
ax.set_xticks(())

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

# Show the plot
plt.show()
# use the real  data
df=pd.read_csv('Cust_Segmentation.csv')

# address is not used
df.drop('Address',axis=1,inplace=True)
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X) # replace nan with zero
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet

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

df['Clus_km']=labels
df.groupby('Clus_km').mean()
```

```
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.scatter()
```




```
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))
```

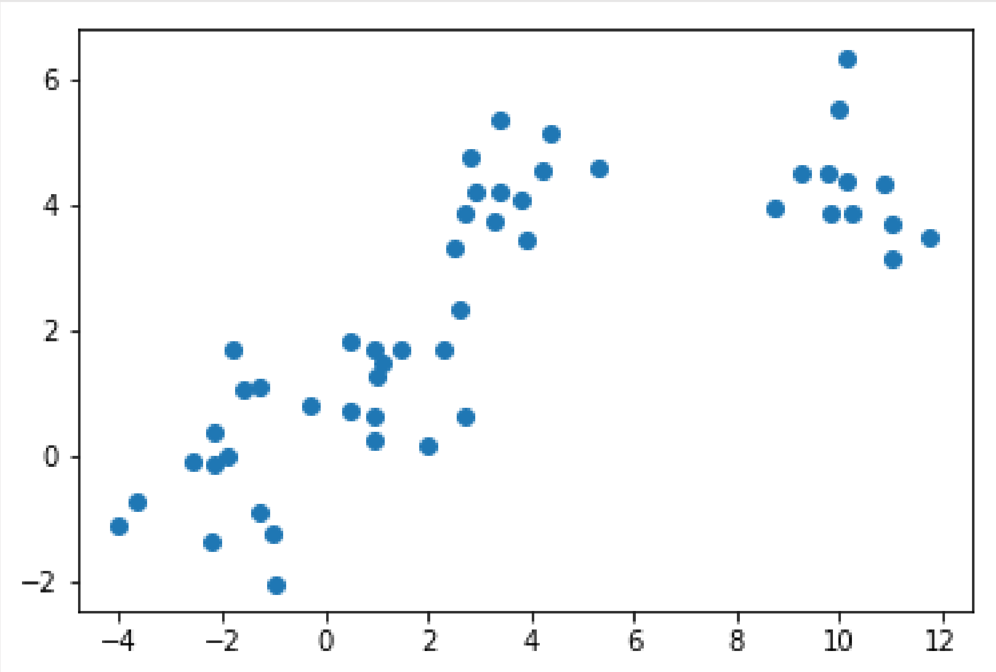


Hierarchical

Code:

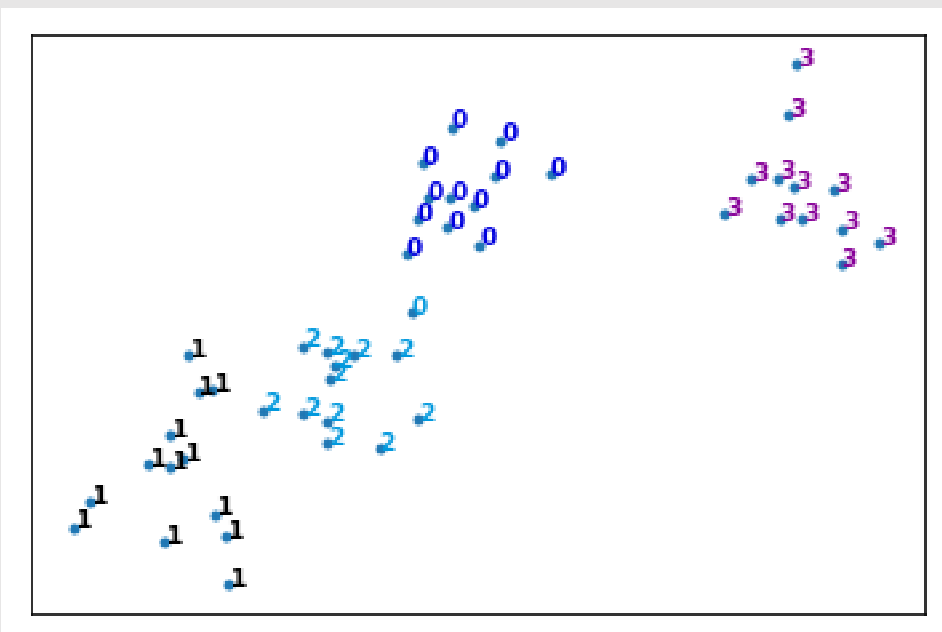
```
import numpy as np

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


# generate data using make_blobs function

X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster_std=0.9)

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




```
# choose how many clusters to form

agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')

agglom.fit(X1,y1)


# 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='.')
```