You have just triggered a mouse gesture. Would you like to learn more about mouse gestures?

jupyter Untitled8 Last Checkpoint: 9 minutes ago (unsaved changes)

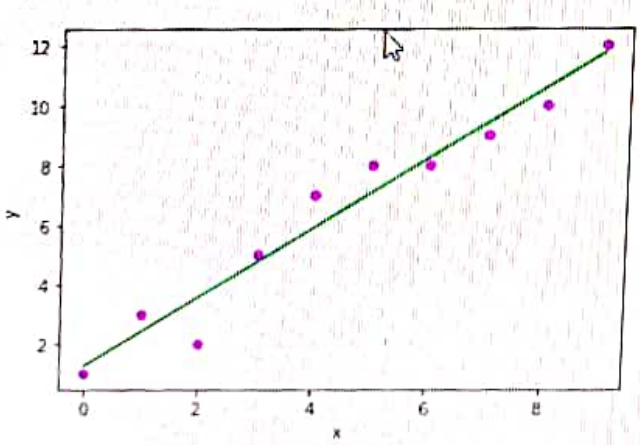
File Edit View Insert Cell Kernel Widgets Help

Run Code

```
    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



In [ ]:

Type here to search

Experiment - 6

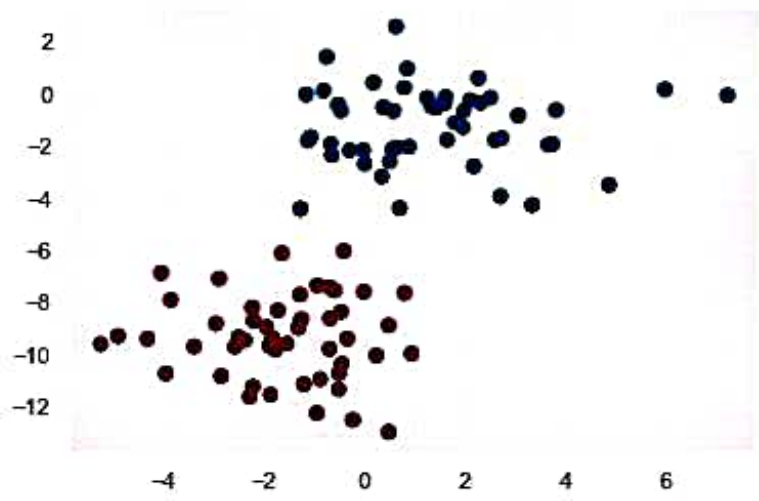
Implement linear regression using python

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x,y):
# number of observations /points
    n=np.size(x)
# mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)
# Calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
# Calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return (b_0, b_1)
def plot_regression_line(x,y,b):
# plotting the actual points as scatter plot
    plt.scatter(x,y, color = "m", marker = "o", s= 30)
# predicted response vector
        y_pred = b[0]+ b[1]*x
```

```
# plotting the regression line
    plt.plot (x,y_pred, color = "g")
# putting Labels
   plt.xlabel ('x')
   plt.ylabel ('y')
# function to show plot
    plt.show()
   def main():
   # observations / data
   x = np.array ([0,1,2,3,4,5,6,7,8,9])
   y = np.array ([1,3,2,5,7,8,8,9,10,12])
   # estimating coefficients
     b = estimate_coef (x,y)
  Print (" Estimated coefficients: \n b_0 = {} \\
                                  nb_1 = {} ".format (b[0], b[1])
   # plotting regression line
     plot_regression_line (x, y, b)
     if __name__ == "__main__":
      main()
```

In [6]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x222116037f0>
```



Experiment 7

Example 1:

Implement Naive Bayes theorem to classify the english text.

Source Code:

```
%matplotlib inline
import numpy as np
import matplotlib.pylot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
x,y = make_blobs (100,2, centers=2, random_state=2,
                  cluster_std=1.5)
Plt.scatter (x[:,0], x[:,1], c=y, s=50, cmap='RdBu').
```

localhost:8889/notebooks/Untitled3.ipynb

azon.in AliExpress

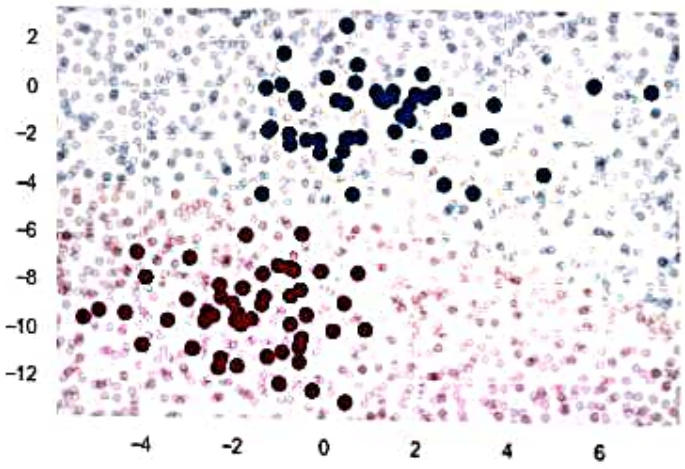
jupyter Untitled3 Last Checkpoint: 10 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

```
yprob = model.predict_proba(Xnew)
yprob[-8:].round(2)
```

Out[7]:
```
array([[0.89, 0.11],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.  , 1.  ],
       [0.15, 0.85]])
```



Example 2:

Implement Naive Bayes theorem to classify the english text

Source code:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
X,y = make_blobs(100, 2, centers=2, random_state=2, cluster_
                                                std=1.5)
plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='RdBu');
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X,y);
rng = np.random.RandomState(0)
Xnew = [-6,-14] + [14,18]*rng.rand(2000,2)
Ynew = model.predict(Xnew)
plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='RdBu')
lim = plt.axis()
plt.scatter(Xnew[:,0], Xnew[:,1], c=ynew, s=20, cmap='RdBu',
                                                alpha=0.1)
plt.axis(lim);
yprob = model.predict_proba(Xnew)
yprob[-8:].round(2)
```

localhost:8889/notebooks/Untitled3.ipynb

AliExpress

ter Untitled3 Last Checkpoint: 12 minutes ago (unsaved changes)

dit View Insert Cell Kernel Widgets Help

Run Code

```
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
# perform the genetic algorithm search
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

```
>0, new best f([1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1]) = -15.000
>1, new best f([1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]) = -16.000
>3, new best f([1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]) = -17.000
>4, new best f([1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -18.000
>4, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -19.000
>7, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000
Done!
f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000000
```

[ ]:

Experiment 8:

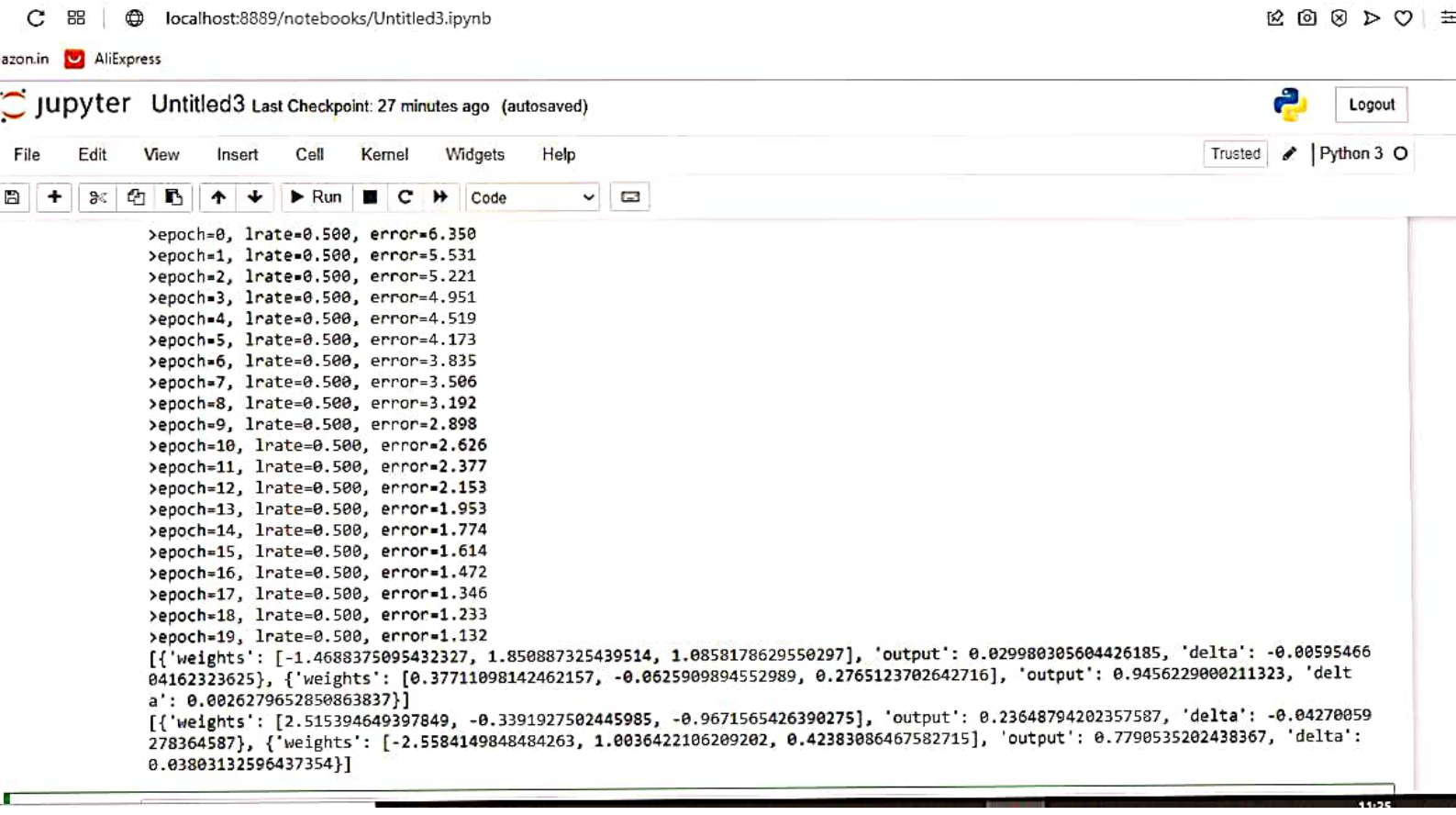
#genetic algorithm to evaluates a binary string based on the number of 1's in the string. a bitstring with a length of 20 bits will have a score of 20 for a string of all 1's in the string.
11111111111111111111 = 20, 11111111110000000000 = 10)

```
from numpy.random import randint
from numpy.random import rand
# objective function
def onemax(x):
    return -sum(x)
# tournament selection
def selection(pop, scores, k=3):
# first random selection
    selection_ix = randit(len(pop))
    for ix in randint(0,len(pop),k-1):
# check if better (e.g. perform a tournament)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
        return pop[selection_ix]
# crossover two parents to create two children
def crossover(p1,p2,r_cross):
# children are copies of parents by default
    c1,c2 = p1.copy(), p2.copy()
# check for recombination
    if rand() < r_cross:
```

```
# select crossover point that is not on the end of the string.
pt = randint(1, len(p1)-2)
# perform crossover
    c1 = p1[:pt] + p2[pt:]
    c2 = p2[:pt] + p1[pt:]
    return [c1, c2]
# mutation operator
def mutation(bitstring, r_mut):
    for i in range(len(bitstring)):
# check for a mutation
    if rand() < r_mut:
    # flip the bit
    bitstring[i] = 1 - bitstring[i]
# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop,
                      r_cross, r_mut):
# initial population of random bitstring
pop = [randit(0, 2, n_bits).tolist() for _ in range(n_pop)]
# keep track of best solution
        best, best_eval = 0,
        objective(pop[0])
# enumerate generations
    for gen in range(n_iter):
# evaluate all candidates in the population
    scores = [objective(c) for c in pop]
```

```
# check for new best solution
    for i in range (n_pop):
    if [ scores [i] < best_eval:
    best, best_eval = pop[i];
    scores[i]
Print(">%d, new best f(%s) = %.3f" % (gen, pop[i], scores[i])
# select parents
selected = [selection (pop, scores) for _ in range (n_pop)]
# create the next generation
    children = list()
    for i in range (0, n_pop, 2):
# get selected parents in pairs
    p1, p2 = selected [i], selected [i+1]
# crossover and mutation
    for c in crossover (p1, p2, r_cross):
# mutation
    mutation (c, r_mut)
# store for next generation
    children . append (c)
# replace population
    pop = children
    return [best, best_eval]
# define the total iterations
    n_iter = 100
# bits
    n_bits = 20
```

```
#define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
#perform the genetic algorithm search
best, score = genetic_algorithm(one_max, n_bits, n_iter,
                                n_pop, r_cross, r_mut)
Print('Done!')
Print('f(%s) = %f' %(best, score))
```


localhost:8889/notebooks/Untitled3.ipynb
azon.in
AliExpress
jupyter
Untitled3 Last Checkpoint: 27 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output': 0.029980305604426185, 'delta': -0.00595466
04162323625}, {'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output': 0.9456229000211323, 'delt
a': 0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output': 0.23648794202357587, 'delta': -0.04270059
278364587}, {'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta':
0.03803132596437354}]

Experiment 9:

Implement the finite words classification system using Back Propogation algorithm.

Source code:

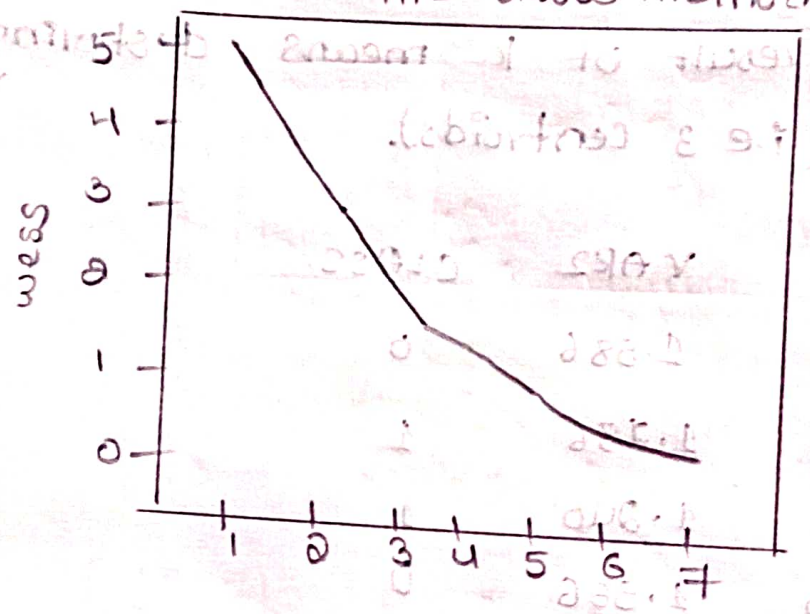
```
from math import exp
from random import seed
from random import random
import matplotlib.pyplot as plt
#initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range
                    (n_inputs+1)]}
    for i in range(n_hidden)]
    network.append(hidden_Layer)
    output_layer = [{'weights':[random() for i in range
                    (n_hidden+1)]}
    for i in range(n_outputs)]
    network.append(output_layer)
    return network
# Calculate neuron activation for an input
    def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
        return activation
```

```
# Transfer neuron activation
def transfer (activation):
return 1.0 / (1.0 + exp(-activation))
# Forward propagate input to a network output
def forward - propagate (network, row):
    inputs = row
for layer in network:
    new_ inputs = []
for neuron in Layer:
activation = activate (neuron ['weights'], inputs)
neuron ['output'] = transfer (activation)
new_inputs. append (neuron ['output'])
    inputs = new_ inputs
    return inputs
# calculate the derivative of an neuron output
    def transfer _ derivative (output):
    return output * (1.0 - output)
# Back propagate error and store in neurons
    def
backward - propagate _ error (network, expected):
for i in reversed (range (len (network))):
    Layer = network [i]
    errors = list()
    if i != len (network) - 1:
    for j in range (len ( Layer)):
        error = 0.0
```

```
for neuron in network[i+1]:
    error += (neuron['weights'][j] * neuron['delta'])
    errors.append(error)
else:
    for j in range(len(layer)):
        neuron = layer[j]
        neuron['delta'] = errors[j] *
transfer_derivative(neuron['output'])
#update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[i-1]
        if i!=0:
            inputs = [neuron['output'] for neuron in network[i-1]]
    for neuron in network[i]:
        for j in range(len(inputs)):
            neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
        neuron['weights'][-1] += l_rate * neuron['delta']
# Train a network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i] - outputs[i]**2
```

```
        for i in range (len (expected))])
        backward - propagate_error (network, expected)
        update_weights (network, row, l_rate)
    print ('>epoch = %d, lrate = %.3f, error = %.3f' % (epoch,
                                                l_rate, sum_error))
# Test training backprop algorithm seed (1)
dataset = [[2.7810836, 2.550537003, 0],
           [1.465489372, 2.362125076, 0],
           [3.396561688, 4.400293529, 0],
           [1.38807019, 1.850220317, 0],
           [3.06407232, 3.005305973, 0],
           [7.627531214, 2.759262235, 1],
           [5.332441248, 2.088626775, 1],
           [6.922596716, 1.77106367, 1],
           [8.675418651, -0.242068655, 1],
           [7.673756466, 3.508563011, 1]]
n_inputs = len (dataset[0]) - 1
n_outputs = len (set ([row[-1] for row in dataset]))
network = initialize_network (n_inputs, 2, n_outputs)
train_network (network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print (layer)
```

o/p [5.631208500000005, 2.487442083333333,
1.385325, 0.8148231166666669, 0.5584978
333334, 0.3542455000000005, 0.1953015000
000008]



o/p :

KMeans(n_clusters=3, random_state=0)

[0 1 1 2 0 2 1 2]

Experiment - 4

Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1 = 0.906 and VAR2 = 0.606, using the result of k-means clustering with 3 means (i.e., 3 centroids)

VAR 1	VAR 2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Source code:

```
# importing the libraries
    import numpy as np
    import matplotlib.PyPlot as Plt
    import pandas as pd.
# importing the Iris dataset with pandas
  dataset = Pd.read_csv [r'c:|users|Desktop|kl_data.csv')
  Print (dataset)
```

```
X = dataset.iloc[:, (1,2)].values
print(x)
#finding the optimum number of clusters for k means classification
from sklearn.cluster import KMeans
wess = []
for i in range(1,8);
kmeans = kmeans(n_clusters = i, init = k_means++;
                max_iter = 300, n_init = 10, random_state = 0)
kmeans.fit(x)
wess.opened(kmeans.inertia)
Print(wess)
# plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1,8),wess)
plt.tittle('The elbow method)
plt xlabel('Number of clusters)
plt.ylabel('Wess')
plt.show()
```

output:

un conditional probability of golf is: 0.4

conditional probability of single with medrisk:

status	risk	
married	highrisk	0.666667
	lowrisk	0.666667
	medrisk	0.833333
single	highrisk	0.666667
	medrisk	0.666667
	lowrisk	0.833333

Experiment - 5

The following training examples map descriptions of individuals onto high, medium and low credit-worthiness

medium	skiing	design	single	twenties	no→highrisk
high	golf	trading	married	forties	yes→lowrisk
low	Speedway	transport	married	thirties	yes→medrisk
medium	football	banking	single	thirties	yes→lowrisk
high	flying	media	married	fifties	yes→highrisk
low	football	security	single	twenties	no→medrisk
medium	golf	media	married	thirties	yes→medrisk
medium	golf	transport	single	forties	Yes→lowrisk
high	skiing	banking	married	thirties	Yes→highrisk
low	golf	unemployed	single	forties	Yes→highrisk

Input attributies are (from left to right) income, recreation, job, status, age-group, how-owner. find the unconditional probability of 'golf' and the conditional probability of 'single' given 'medRisk' in the dataset?

Source code:

```
import pandas as pd
import numpy as np
dataset = pd.read_csv(r'C:\Users\Desktop\credit.csv')
df = Pd.dataframe(dataset)
l = len(df)
```

```
rec = df['recreation'].values.tolist()
c=0
for i in rec:
if (i=="golf"):
  cl = 1
Print(" unconditional probability of golf is: ", c/l];
  cl = 0
  risk = df['risk'].values.tolist()
  for i in the risk:
   if (i=='med risk'):
      clt = 1
 Print(" Conditional probability of single with med risk:")
Print(df.groupby('status') ~~('visit')~~ ('risk').value_counts
                                                ()/cl)
```