Name: Zuhair Naqi

Class: BSSE(SEC-A) BATCH-18

Seat # : EP-1850147

Lab: Final Exam

Teacher: Ma'am Bakhtawar Rizwi

Lab 01 - Conditions

Q. Find indexes of search value without iterating loop

[11]:
```python
# find indexes of repeated element
list = ['a', 'b', 'c', 'd', 'a', 'e', 'f', 'a']
searchValue = 'a'
indexes = []
count = 0
length = len(list)

def findIndex():
    global count
    if count == length:
        return;
    elif list[count] == searchValue:
        indexes.append(count)
    count += 1
    findIndex()

findIndex()
print(indexes)
```

```
[0, 4, 7]
```

Lab 02 - Loops

Prime Numbers

Q1. Write a for loop in python to print prime numbers from the list of 50 different numbers. The loop should be able to show non-prime numbers as well.

```python
for num in range(2,50):
    isPrimeNumber = True
    for i in range(2, num):
        if(num%i == 0):
            isPrimeNumber = False
            break
    if isPrimeNumber:
        print("PrimeNumber = ", num)
    else:
        print("Non-PrimeNumber = ", num)
```

Output:

```
PrimeNumber =  2
PrimeNumber =  3
Non-PrimeNumber =  4
PrimeNumber =  5
Non-PrimeNumber =  6
PrimeNumber =  7
Non-PrimeNumber =  8
Non-PrimeNumber =  9
Non-PrimeNumber =  10
PrimeNumber =  11
Non-PrimeNumber =  12
PrimeNumber =  13
Non-PrimeNumber =  14
Non-PrimeNumber =  15
Non-PrimeNumber =  16
PrimeNumber =  17
Non-PrimeNumber =  18
PrimeNumber =  19
Non-PrimeNumber =  20
Non-PrimeNumber =  21
Non-PrimeNumber =  22
PrimeNumber =  23
Non-PrimeNumber =  24
Non-PrimeNumber =  25
Non-PrimeNumber =  26
Non-PrimeNumber =  27
Non-PrimeNumber =  28
PrimeNumber =  29
Non-PrimeNumber =  30
```

```
Non-PrimeNumber =  22
PrimeNumber =  23
Non-PrimeNumber =  24
Non-PrimeNumber =  25
Non-PrimeNumber =  26
Non-PrimeNumber =  27
Non-PrimeNumber =  28
PrimeNumber =  29
Non-PrimeNumber =  30
PrimeNumber =  31
Non-PrimeNumber =  32
Non-PrimeNumber =  33
Non-PrimeNumber =  34
Non-PrimeNumber =  35
Non-PrimeNumber =  36
PrimeNumber =  37
Non-PrimeNumber =  38
Non-PrimeNumber =  39
Non-PrimeNumber =  40
PrimeNumber =  41
Non-PrimeNumber =  42
PrimeNumber =  43
Non-PrimeNumber =  44
Non-PrimeNumber =  45
Non-PrimeNumber =  46
PrimeNumber =  47
Non-PrimeNumber =  48
Non-PrimeNumber =  49
```

Q2. Write a while loop in python to print prime numbers from the list of 50 different numbers. The loop should be able to show non-prime numbers as well.

```
count = 2
rangeCount = 50
while count < rangeCount:
  isPrimeNumber = True;
  initialCount = 2
  while initialCount < count:
    if(count%initialCount == 0):
      isPrimeNumber = False
      break
    initialCount += 1
  if isPrimeNumber:
    print("PrimeNumber = ", count)
  else:
    print("Non-PrimeNumber = ", count)
  count += 1
```

Output:

```
PrimeNumber =  2
PrimeNumber =  3
Non-PrimeNumber =  4
PrimeNumber =  5
Non-PrimeNumber =  6
PrimeNumber =  7
Non-PrimeNumber =  8
Non-PrimeNumber =  9
Non-PrimeNumber =  10
PrimeNumber =  11
Non-PrimeNumber =  12
PrimeNumber =  13
Non-PrimeNumber =  14
Non-PrimeNumber =  15
Non-PrimeNumber =  16
PrimeNumber =  17
Non-PrimeNumber =  18
PrimeNumber =  19
Non-PrimeNumber =  20
Non-PrimeNumber =  21
Non-PrimeNumber =  22
PrimeNumber =  23
Non-PrimeNumber =  24
Non-PrimeNumber =  25
Non-PrimeNumber =  26
Non-PrimeNumber =  27
Non-PrimeNumber =  28
PrimeNumber =  29
Non-PrimeNumber =  30
PrimeNumber =  31
Non-PrimeNumber =  32
Non-PrimeNumber =  33
```

```
PrimeNumber =  19
Non-PrimeNumber =  20
Non-PrimeNumber =  21
Non-PrimeNumber =  22
PrimeNumber =  23
Non-PrimeNumber =  24
Non-PrimeNumber =  25
Non-PrimeNumber =  26
Non-PrimeNumber =  27
Non-PrimeNumber =  28
PrimeNumber =  29
Non-PrimeNumber =  30
PrimeNumber =  31
Non-PrimeNumber =  32
Non-PrimeNumber =  33
Non-PrimeNumber =  34
Non-PrimeNumber =  35
Non-PrimeNumber =  36
PrimeNumber =  37
Non-PrimeNumber =  38
Non-PrimeNumber =  39
Non-PrimeNumber =  40
PrimeNumber =  41
Non-PrimeNumber =  42
PrimeNumber =  43
Non-PrimeNumber =  44
Non-PrimeNumber =  45
Non-PrimeNumber =  46
PrimeNumber =  47
Non-PrimeNumber =  48
Non-PrimeNumber =  49
>
```

Class Tasks:

Q.1 Mark 2.

~~while~~

Q1 write down a while loop which prints only odd numbers range(20).

```
count = 1.
while count < 20.
    if (count % 2 != 0)
        Print(count)
    count++.
```

Q2 write down a for loop which prints only even numbers range (20).

```
for x in range (20)
    if (x % 2 == 0)
        Print (x)
```

Lab 03 - Data Cleaning

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

     data = np.random.rand(3, 4) # row, column
     print(data)
```

```
[[0.3857201  0.465476   0.07190713 0.90717523]
 [0.62400624 0.76970064 0.44919913 0.44680153]
 [0.21054095 0.37881681 0.08590543 0.46644336]]
```

```
[5]: dataFrame = pd.DataFrame(data, index=[1, 2, 3], columns=['c1','c2','c3','c4'])
     print(dataFrame)
```

```
         c1        c2        c3        c4
1  0.385720  0.465476  0.071907  0.907175
2  0.624006  0.769701  0.449199  0.446802
3  0.210541  0.378817  0.085905  0.466443
```

```
[6]: # Data frame 2
     # Arranging index and adding new rows with null values
     newFrame =  dataFrame.reindex([1, 4, 2, 3])
     print(newFrame)
```

```
         c1        c2        c3        c4
1  0.385720  0.465476  0.071907  0.907175
4       NaN       NaN       NaN       NaN
2  0.624006  0.769701  0.449199  0.446802
3  0.210541  0.378817  0.085905  0.466443
```

```
[7]: # Check if value of coloum is null
     print(newFrame['c2'].isnull())
     print(newFrame[newFrame['c2'].isnull()])
```

```
1    False
4     True
2    False
3    False
Name: c2, dtype: bool
   c1  c2  c3  c4
4 NaN NaN NaN NaN
```

```
[8]: # Replace missing data
print(newFrame['c2'].fillna("Zuhair"))
```

```
1    0.465476
4      Zuhair
2    0.769701
3    0.378817
Name: c2, dtype: object
```

```
[9]: #Drop missing data
print(newFrame['c1'].dropna())
```

```
1    0.385720
2    0.624006
3    0.210541
Name: c1, dtype: float64
```

```
[10]: #Fill the null data in columns with specific value
for col in newFrame:
    newFrame[col] = newFrame[col].fillna('filled')
print(newFrame)
```

```
         c1        c2         c3        c4
1   0.38572  0.465476  0.0719071  0.907175
4    filled    filled     filled    filled
2  0.624006  0.769701   0.449199  0.446802
3  0.210541  0.378817  0.0859054  0.466443
```

```
[12]: # Change specific column index
newFrame['c1'][2] = 'Specific value'
print(newFrame)
```

```
               c1        c2         c3        c4
1         0.38572  0.465476  0.0719071  0.907175
4          filled    filled     filled    filled
2  Specific value  0.769701   0.449199  0.446802
3        0.210541  0.378817  0.0859054  0.466443
```

```
[13]: # Replace specific value
print(newFrame.replace('filled', 'replace'))
```

```
               c1        c2         c3        c4
1         0.38572  0.465476  0.0719071  0.907175
4         replace   replace    replace   replace
2  Specific value  0.769701   0.449199  0.446802
3        0.210541  0.378817  0.0859054  0.466443
```

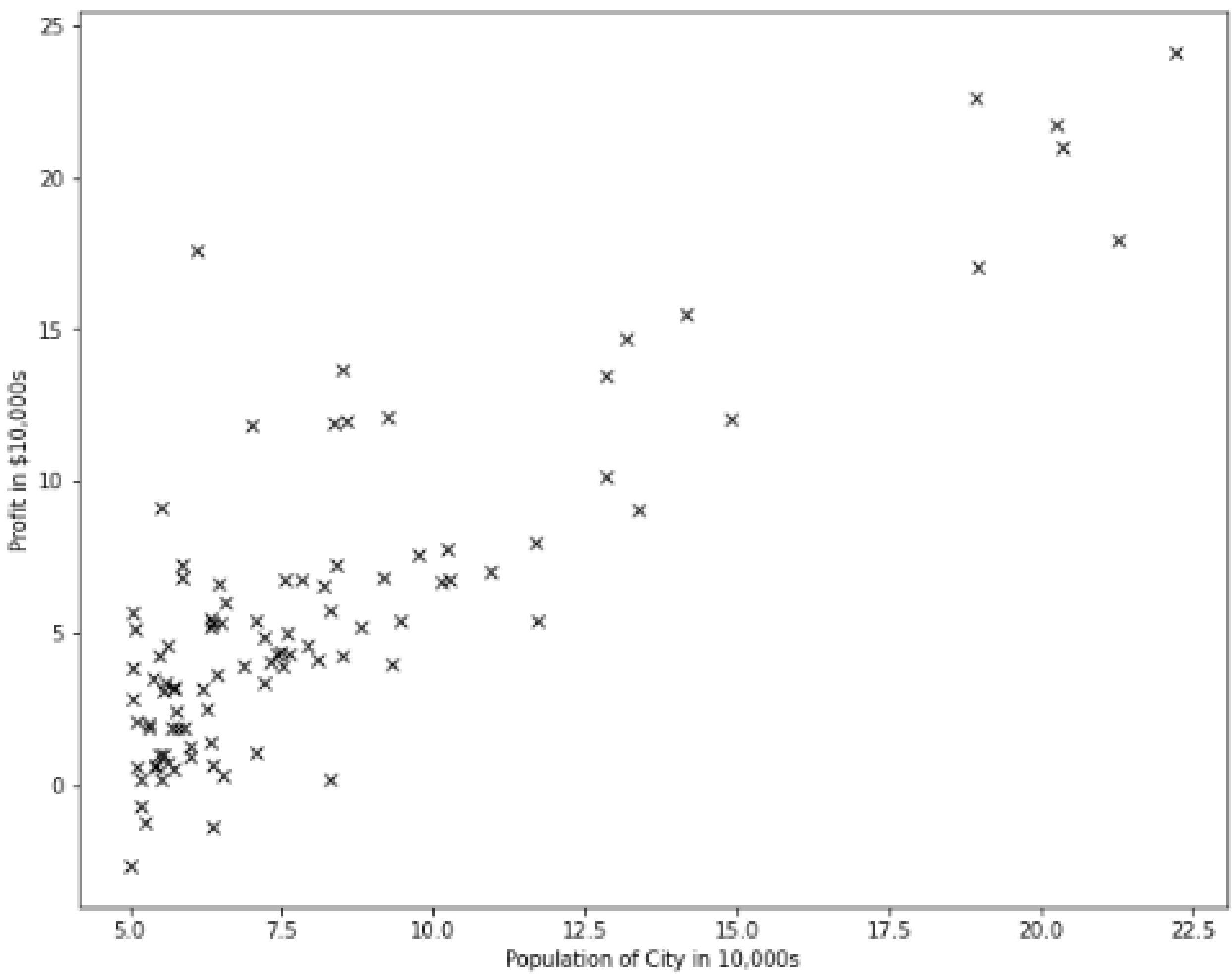
Lab 04 – Gradient Descent For Linear Regression

```
[1]: import pandas as pd;
import matplotlib.pyplot as plt
import numpy as np

p = pd.read_csv('ex1data1.txt', names=['population', 'profit'])

# Split population and profit in separate variable
population = pd.DataFrame(p.population)
profit = pd.DataFrame(p.profit)
m = len(profit)

print(population)
print(profit)
```

```
    population
0       6.1101
1       5.5277
2       8.5186
3       7.0032
4       5.8598
..         ...
92      5.8707
93      5.3054
94      8.2934
95     13.3940
96      5.4369

[97 rows x 1 columns]
      profit
0   17.59200
1    9.13020
2   13.66200
3   11.85400
4    6.82330
..       ...
92   7.20290
93   1.98690
94   0.14454
95   9.05510
96   0.61705

[97 rows x 1 columns]
```

[2]:
```
# Plot the data using matplotlib.pyplot.plot( ) function as

plt.figure(figsize=(10,8))
plt.plot(population, profit, 'kx')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
```

[2]: Text(0, 0.5, 'Profit in $10,000s')



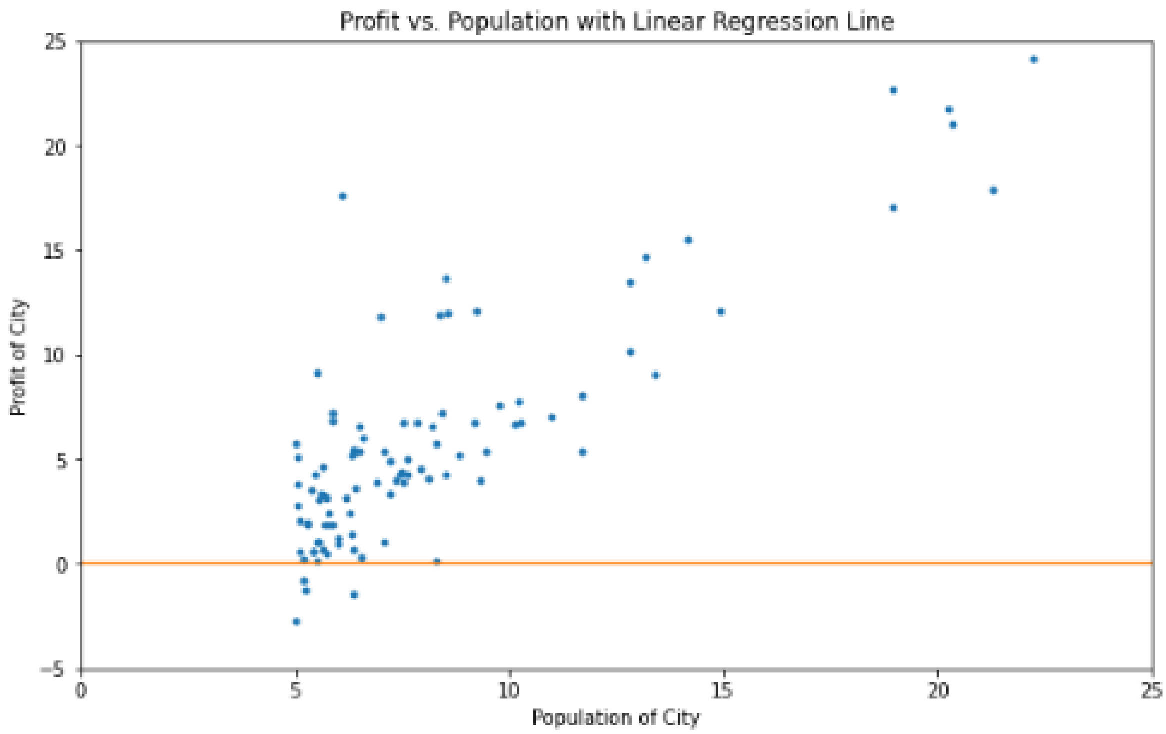
[3]:
```
# For gradient descent
iter = 1000
alpha = 0.01

population['intercept'] = 1
X = np.array(population)
y = np.array(profit).flatten()
theta = np.array([0, 0])
```

[4]:
```
# Define function for cost Linear regression and gradient descent
def cost_function(X, y, theta):
    m = len(y)

    ## Calculate the cost with the given parameters
    J = np.sum((X.dot(theta)-y)**2)/2/m

    return J

def gradient_descent(X, y, theta, alpha, iterations):

    cost_history = [0] * iterations

    for iteration in range(iterations):
        hypothesis = X.dot(theta)

        loss = hypothesis-y
        gradient = X.T.dot(loss)/m
        theta = theta - alpha*gradient
        cost = cost_function(X, y, theta)
        cost_history[iteration] = cost

    return theta, cost_history
```

[6]:
```
cost_function(X, y, theta)
gd = gradient_descent(X,y,theta,alpha, iter)
```

[7]:

```
best_fit_x = np.linspace(0, 25, 20)
best_fit_y = [theta[1] + theta[0]*xx for xx in best_fit_x]

plt.figure(figsize=(10,6))
plt.plot(population.population, profit, '.')
plt.plot(best_fit_x, best_fit_y, '-')
plt.axis([0,25,-5,25])
plt.xlabel('Population of City')
plt.ylabel('Profit of City')
plt.title('Profit vs. Population with Linear Regression Line')
plt.show()
```



Q- Search a dataset for Linear Regression and apply same algorithm on your dataset. Print the optimized parameters and visualizations and attach in your file. Also attach the code of this part in your file.

```
[ ]: import pandas as pd;
     import matplotlib.pyplot as plt
     import numpy as np

     data = pd.read_csv('insurance.csv')
```

```
[ ]: data.head()
```

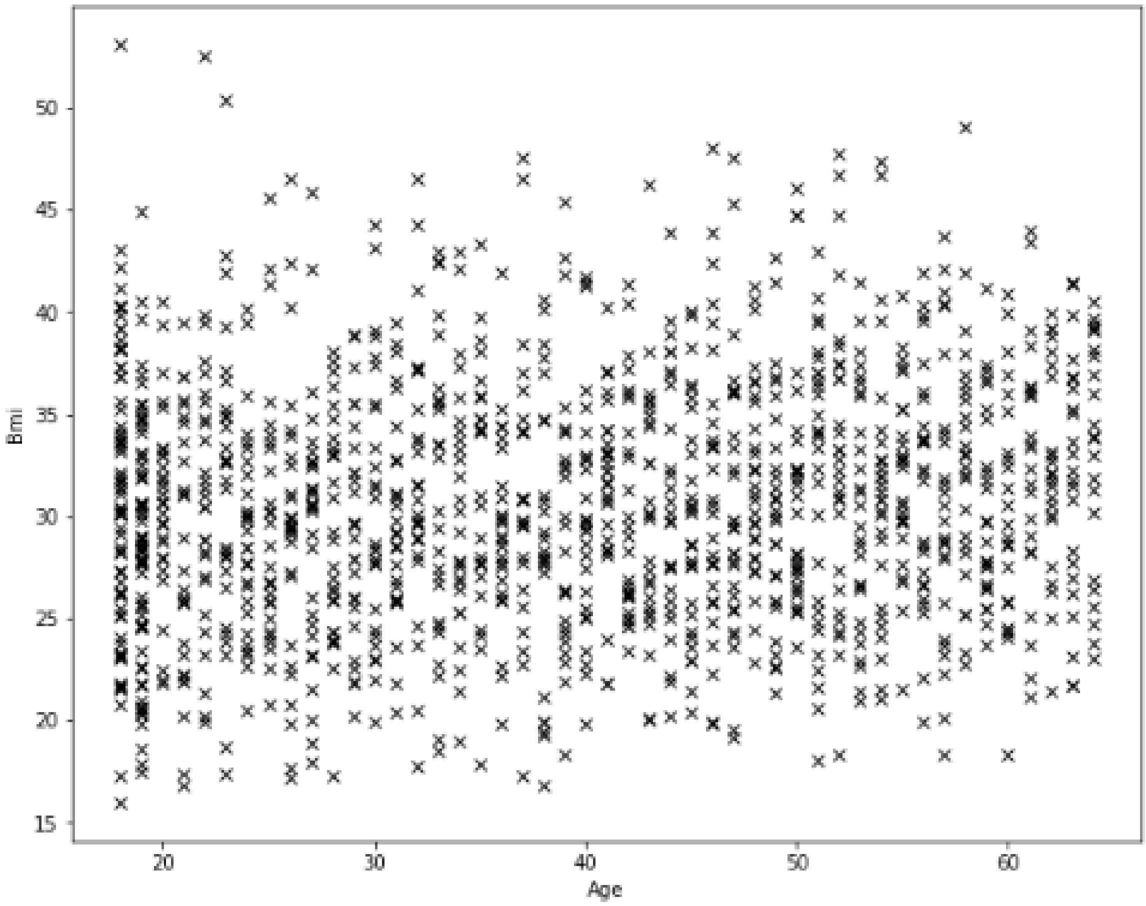
```
[12]: # Split age and bmi in separate variable
      age = pd.DataFrame(p.age)
      bmi = pd.DataFrame(p.bmi)
      m = len(bmi)

      print(age)
      print(bmi)
```

```
      age
0      19
1      18
2      28
3      33
4      32
...   ...
1333   50
1334   18
1335   18
1336   21
1337   61

[1338 rows x 1 columns]
```

```
         bmi
0     27.900
1     33.770
2     33.000
3     22.705
4     28.880
...      ...
1333  30.970
1334  31.920
1335  36.850
1336  25.800
1337  29.070

[1338 rows x 1 columns]
```

[13]:
```python
# Plot the data using matplotlib.pyplot.plot( ) function

plt.figure(figsize=(10,8))
plt.plot(age, bmi, 'kx')
plt.xlabel('Age')
plt.ylabel('Bmi')
```

[13]: Text(0, 0.5, 'Bmi')

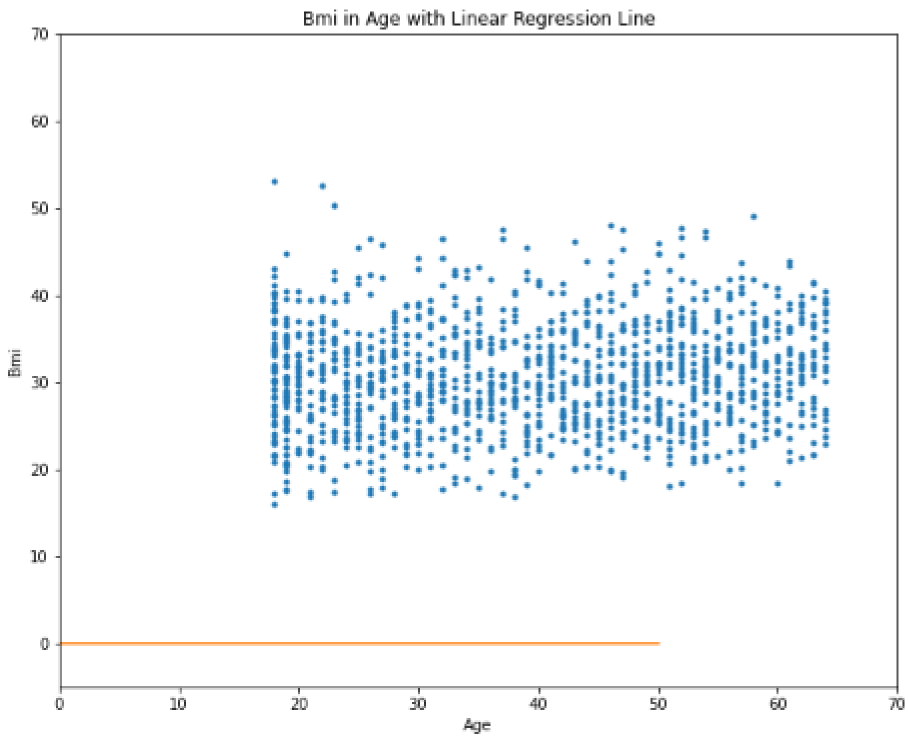


[14]:
```
# For gradient descent
iter = 1000
alpha = 0.01

age['intercept'] = 1
X = np.array(age)
y = np.array(bmi).flatten()
theta = np.array([0, 0])
```

[15]:
```
# Define function for cost Linear regression and gradient descent
def cost_function(X, y, theta):
    m = len(y)

    ## Calculate the cost with the given parameters
    J = np.sum((X.dot(theta)-y)**2)/2/m

    return J

def gradient_descent(X, y, theta, alpha, iterations):

    cost_history = [0] * iterations

    for iteration in range(iterations):
        hypothesis = X.dot(theta)

        loss = hypothesis-y
        gradient = X.T.dot(loss)/m
        theta = theta - alpha*gradient
        cost = cost_function(X, y, theta)
        cost_history[iteration] = cost

    return theta, cost_history
```

[25]:

```
cost_function(X, y, theta)
best_fit_x = np.linspace(0, 50, 60)
best_fit_y = [theta[1] + theta[0]*xx for xx in best_fit_x]

plt.figure(figsize=(10,8))
plt.plot(age.age, bmi, '.')
plt.plot(best_fit_x, best_fit_y, '-')
plt.axis([0,70,-5,70])
plt.xlabel('Age')
plt.ylabel('Bmi')
plt.title('Bmi in Age with Linear Regression Line')
plt.show()
```



Lab 05– Native Bayes

```
from sklearn.naive_bayes import GaussianNB
import numpy as np

#assigning predictor and target variables
x= np.array([[-3,7],[1,5], [1,2], [-2,0], [2,3], [-4,0], [-1,1], [1,1], [-2,2], [2,7]
, [-4,1], [-2,7]])
Y = np.array([3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 4, 4])
```

```
#Create a Gaussian Classifier
model = GaussianNB()
# Train the model using the training sets
model.fit(x, Y)
```

```
GaussianNB()
```

```
#Predict Output
predicted= model.predict([[1,2],[3,4]])
print (predicted)
```

```
[3 4]
```

Convert the “Play tennis” example discussed in class into numeric form and initialize x and y values based on that example. Now run the code for the new x values as discussed in class and print the output. Attach code and output in file.

```python
# indexes --> [Overcast, Sunny, Rainy]
X_data = np.array([[1,0],[0,1],[2,1],[1,1],[1,1],[0,1],[2,0],[2,0],
 [1,1],[2,1],[1,0],[0,1],[0,1],[2,0]])
```

```python
Y_data = np.array([0,0,1,1,1,0,1,0,1,1,1,1,1,0])
```

```python
model = GaussianNB()
model.fit(X_data, Y_data)
```

```
GaussianNB()
```

```python
predicted= model.predict([[2,0],[2,1],[2,2]])
print (predicted)
```

```
[0 1 1]
```

Lab 06– Descision Tree Using Sklearn

```
[12]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree


balance_data = pd.read_csv('balance-scale.data', sep= ',', header= None)
print(balance_data)

print("Dataset Lenght:: ", len(balance_data))
print("Dataset Shape:: ", balance_data.shape)

X = balance_data.values[:, 1:5]
Y = balance_data.values[:,0]

X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, random_state = 100)

clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,  max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
print(clf_entropy)

# Prediction for Decision Tree classifier with criterion as information gain
y_pred_en = clf_entropy.predict(X_test)
print(y_pred_en)


# Calculate accuracy score
```

```
# Calculate accuracy score
print("Accuracy is ", accuracy_score(y_test,y_pred_en)*100)

# convert the trained fruit classifier into graphviz object and saves into the txt file.
with open("balanceScale.txt", "w") as f:
    f = tree.export_graphviz(clf_entropy, out_file=f)
```

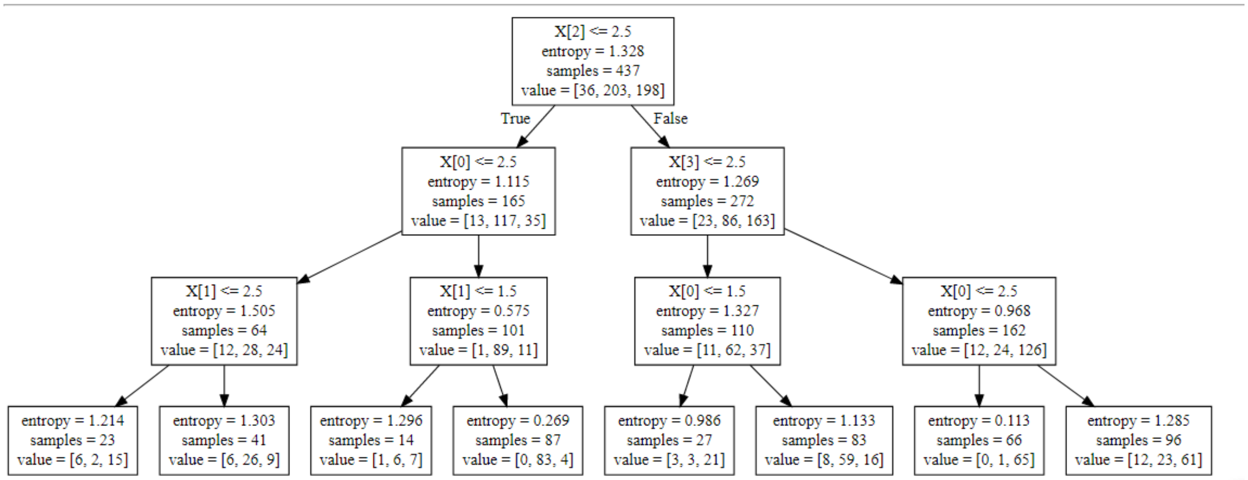
```
     0  1  2  3  4
0    B  1  1  1  1
1    R  1  1  1  2
2    R  1  1  1  3
3    R  1  1  1  4
4    R  1  1  1  5
..  .. .. .. .. ..
620  L  5  5  5  1
621  L  5  5  5  2
622  L  5  5  5  3
623  L  5  5  5  4
624  B  5  5  5  5

[625 rows x 5 columns]
Dataset Lenght::  625
Dataset Shape::  (625, 5)
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=5,
                       random_state=100)
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
```

All work visualization and output:

BalanceScale.txt

```
digraph Tree {
node [shape=box] ;
0 [label="X[2] <= 2.5\nentropy = 1.328\nsamples = 437\nvalue = [36, 203, 198]"] ;
1 [label="X[0] <= 2.5\nentropy = 1.115\nsamples = 165\nvalue = [13, 117, 35]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="X[1] <= 2.5\nentropy = 1.505\nsamples = 64\nvalue = [12, 28, 24]"] ;
1 -> 2 ;
3 [label="entropy = 1.214\nsamples = 23\nvalue = [6, 2, 15]"] ;
2 -> 3 ;
4 [label="entropy = 1.303\nsamples = 41\nvalue = [6, 26, 9]"] ;
2 -> 4 ;
5 [label="X[1] <= 1.5\nentropy = 0.575\nsamples = 101\nvalue = [1, 89, 11]"] ;
1 -> 5 ;
6 [label="entropy = 1.296\nsamples = 14\nvalue = [1, 6, 7]"] ;
5 -> 6 ;
7 [label="entropy = 0.269\nsamples = 87\nvalue = [0, 83, 4]"] ;
5 -> 7 ;
8 [label="X[3] <= 2.5\nentropy = 1.269\nsamples = 272\nvalue = [23, 86, 163]"] ;
0 -> 8 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
9 [label="X[0] <= 1.5\nentropy = 1.327\nsamples = 110\nvalue = [11, 62, 37]"] ;
8 -> 9 ;
10 [label="entropy = 0.986\nsamples = 27\nvalue = [3, 3, 21]"] ;
9 -> 10 ;
11 [label="entropy = 1.133\nsamples = 83\nvalue = [8, 59, 16]"] ;
9 -> 11 ;
12 [label="X[0] <= 2.5\nentropy = 0.968\nsamples = 162\nvalue = [12, 24, 126]"] ;
8 -> 12 ;
13 [label="entropy = 0.113\nsamples = 66\nvalue = [0, 1, 65]"] ;
12 -> 13 ;
14 [label="entropy = 1.285\nsamples = 96\nvalue = [12, 23, 61]"] ;
12 -> 14 ;
}
```

Graph:


X[2] <= 2.5
entropy = 1.328
samples = 437
value = [36, 203, 198]
True
False
X[0] <= 2.5
entropy = 1.115
samples = 165
value = [13, 117, 35]
X[3] <= 2.5
entropy = 1.269
samples = 272
value = [23, 86, 163]
X[1] <= 2.5
entropy = 1.505
samples = 64
value = [12, 28, 24]
X[1] <= 1.5
entropy = 0.575
samples = 101
value = [1, 89, 11]
X[0] <= 1.5
entropy = 1.327
samples = 110
value = [11, 62, 37]
X[0] <= 2.5
entropy = 0.968
samples = 162
value = [12, 24, 126]
entropy = 1.214
samples = 23
value = [6, 2, 15]
entropy = 1.303
samples = 41
value = [6, 26, 9]
entropy = 1.296
samples = 14
value = [1, 6, 7]
entropy = 0.269
samples = 87
value = [0, 83, 4]
entropy = 0.986
samples = 27
value = [3, 3, 21]
entropy = 1.133
samples = 83
value = [8, 59, 16]
entropy = 0.113
samples = 66
value = [0, 1, 65]
entropy = 1.285
samples = 96
value = [12, 23, 61]

Lab 07– Performance Metrics

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

X_actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
Y_predic = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]

results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix :', results)
```

```
Confusion Matrix : [[3 3]
 [1 3]]
```

```
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
print('AUC-ROC:',roc_auc_score(X_actual, Y_predic))
print('LOGLOSS Value is',log_loss(X_actual, Y_predic))
```

```
Accuracy Score is 0.6
Classification Report :
              precision    recall  f1-score   support

           0       0.75      0.50      0.60         6
           1       0.50      0.75      0.60         4

    accuracy                           0.60        10
   macro avg       0.62      0.62      0.60        10
weighted avg       0.65      0.60      0.60        10

AUC-ROC: 0.625
LOGLOSS Value is 13.815750437193334
```

Q- Why we use performance matrices in machine learning?

- It is used for finding the correctness and accuracy of the model.
- It is used for Classification problem where the output can be of two or more types of classes.
- We can use classification performance metrics such as Log-Loss, Accuracy, AUC(Area under Curve) etc in ML. Another example of metric for evaluation of machine learning algorithms is precision, recall, which can be used for sorting algorithms primarily used by search engines.

Q- We have a confusion matric

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

This indicated the number of cancer patients tested and who came actually true . write the code in python to calculate the classification accuracy and classification report of the given data.

```
X_actual = [1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 0]
```

```
Y_predic = [1, 1, 1, 0, 1, 0, 1, 1, 0, 0,
 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 0]
```

```
results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix')
print(results)
```

```
Confusion Matrix
[[ 50  10]
 [  5 100]]
```

```
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
```

```
Accuracy Score is 0.9090909090909091
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        60
           1       0.91      0.95      0.93       105

    accuracy                           0.91       165
   macro avg       0.91      0.89      0.90       165
weighted avg       0.91      0.91      0.91       165
```

Lab 09– K-Mean Algorithm

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

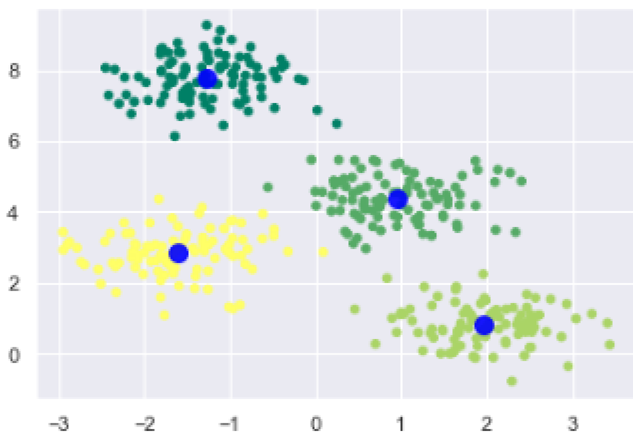
```
C:\Users\Perfect\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: Futu
eprecated in version 0.22 and will be removed in version 0.24. The corresponding cl
sets. Anything that cannot be imported from sklearn.datasets is now part of the pri
  warnings.warn(message, FutureWarning)
```

```
X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60, random_state=0)
```

```
plt.scatter(X[:, 0], X[:, 1], s=20);
plt.show()
```

```
# Create Clusters and train the dataset on a variable and do predictions

kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
# check and visualized the centers which have been picked by K-mean.
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20, cmap='summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```



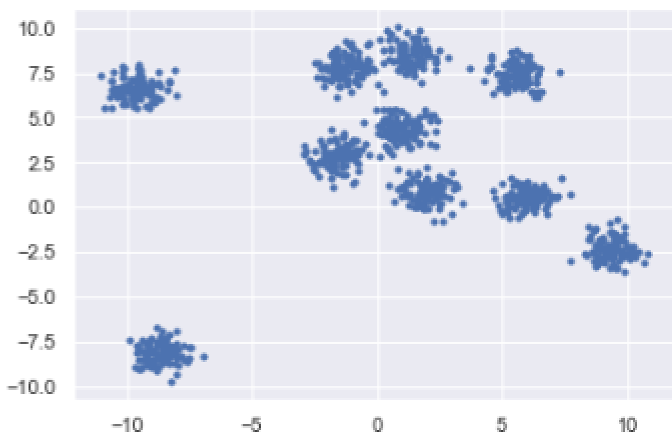
Q- What is importance of K- mean theorem in clustering algorithms od machine learning?.

Ans: K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.

Q- Write a code snippet in python to perform k mean algorithm implementation on a data set. create 10 clusters and calculate ceroids of data. And visualized them.

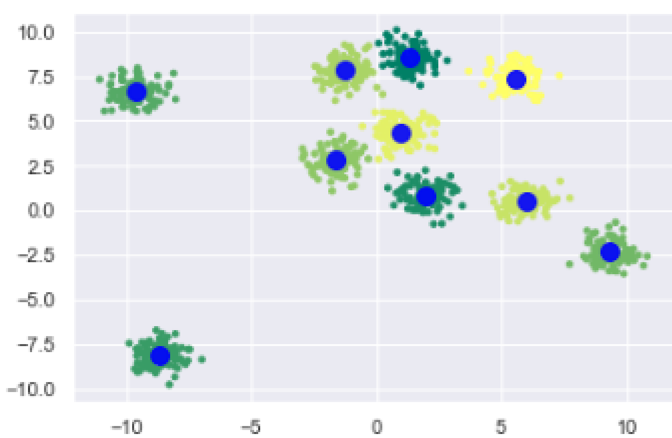
```
X, y_true = make_blobs(n_samples=1024, centers=10, cluster_std=0.6, random_state=0)
```

```
plt.scatter(X[:, 0], X[:, 1], s=10);
plt.show()
```




```
kmeans = KMeans(n_clusters=10)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=10, cmap='summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```



Lab 10– Herarical Clustering

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
%matplotlib inline
```

```
data=pd.read_csv('Wholesale customers data.csv')
data.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

```
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
data_scaled.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286
2	0.000125	0.000187	0.396552	0.549792	0.479632	0.150119	0.219467	0.489619
3	0.000065	0.000194	0.856837	0.077254	0.272650	0.413659	0.032749	0.115494
4	0.000079	0.000119	0.895416	0.214203	0.284997	0.155010	0.070358	0.205294

```python
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')
```

```
<matplotlib.lines.Line2D at 0x20aa5daf520>
```