Chapter 1

APRIORI ALGORITHM

1.1 Objectives:

To learn the basics of Apriori Algorithm in machine learning and implementing it.

1.2 Description:

The Apriori algorithm is a categorization algorithm. Some algorithms are used to create binary
appraisals of information or find a regression relationship. Others are used to predict trends and patterns that are originally identified. Apriori is a basic machine learning algorithm which
is used to sort information into categories. Sorting information can be incredibly helpful with any data management process. It ensures that data users are appraised of new information and can figure out the data that they are working with.

1.2.1 How Apriori works:

This machine learning algorithm works by identifying a particular characteristic of a data set and attempting to note how frequently that characteristic pops up throughout the set. This idea
requires some extra work on the part of the person implementing the design and, later, the machine itself. The definition of "frequent" is inherently relative and only makes sense in context. Therefore, the idea is implemented in the Apriori algorithm through a pre-arranged amount determined by either the operator or the algorithm. A "frequent" data characteristic is one that occurs above that pre-arranged amount, known as a support.

The characteristics that are frequent can then be analyzed and place into pairs. This process helps to point out more relationships between relevant data points. Other forms of data can be pruned and placed into their own categories. Pruning helps to further differentiate between categories that do and do not reach the overall support amount. Next, the data set can be analyzed by looking for triplets. These triplets show even greater frequency. Analysis can detect more and more relations throughout the body of data until the algorithm has exhausted all of the possible.

1.2.2 Uses of Apriori Algorithm

Apriori is mainly used for sorting large amounts of data. Sorting data often occurs because of association rules. Rules help show what aspects of data different sets have in common. Categories can then be built around those association rules. With data in categories, algorithms
and users can spot new trends and structure data sets. They may have a better ability to point out trends over time. The algorithm can also be used to track how relationships develop and categories are built.

Apriori can be used as a basis for an artificial neural network. It can help the network make sense of large reams of data and sort data into categories by frequency almost instantaneously.

An artificial neural network using Apriori can also tweak the weighting on different categories
to expand or diminish the importance of those categories. As a result, an artificial neural network can process data, identify trends, and elaborate on patterns that would otherwise be missed. Apriori is incredibly helpful for data analysts in numerous fields. Its importance will

only continue to grow as more and more fields use artificial intelligence to make sense of massive data sets. Apriori will continue to be an essential tool in the growth of machine learning and artificial intelligence for years to come.

1.3 Implementation of Apriori Algorithm:

Dataset: movie_dataset.csv
Goal: Implementing Apriori algorithm to:

(i) Find all frequent movie sets.
(ii) Generate association rules with minimum support, length, confidence and lift.
(iii) Find support and confidence and lift of each rule.

1.4 Source Code:

In [33]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori

data = pd.read_csv('movie_dataset.csv', header=None)
data.head()
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	The Revenant	13 Hours	Allied	Zootopia	Jigsaw	Achorman	Grinch	Fast and Furious	Ghostbusters	Wolverine	Mad Max	John Wick	La La Land	The Good Dunosaur
1	Beirut	Martian	Get Out	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Deadpool	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	X-Men	Allied	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Ninja Turtles	Moana	Ghost in the Shell	Ralph Breaks the Internet	John Wick	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [3]:
```
len(data)
```

Out[3]:
```
7501
```

In [27]:
```
lists = []
for i in range(0, len(data)):
    lists.append([str(data.values[i,j]) for j in range(0, 20)])

association_rules = apriori(lists, min_support=0.0053, min_confidence=0.2, min_lift=3, min_length=2)
association_results = list(association_rules)
print(association_results[0])
```

```
RelationRecord(items=frozenset({'Green Lantern', 'Red Sparrow'}), support=0.005732568990801226, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Red Sparrow'}), items_add=frozenset({'Green Lantern'}), confidence=0.3006993006993007, lift=3.790832696715049)])
```

In [32]:
```
for item in association_results:

    pair = item[0]
    items = [x for x in pair]
    print("Rule: ",end='')
    for i in range(0,len(items)-1):
        if i is len(items)-2:
            print(items[i],end='')
        else:
            print(items[i],end=' & ')
    print(" --> "+items[i+1])

    print("Support: " + str(item[1]))

    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("====================================")
```

```
Rule: Green Lantern --> Red Sparrow
Support: 0.005732568990801226
Confidence: 0.3006993006993007
Lift: 3.790832696715049
====================================
Rule: Green Lantern --> Star Wars
Support: 0.005865884548726837
Confidence: 0.3728813559322034
Lift: 4.700811850163794
====================================
Rule: Jumanji --> Kung Fu Panda
Support: 0.015997866951073192
Confidence: 0.3234501347708895
Lift: 3.2919938411349285
====================================
Rule: Jumanji --> Wonder Woman
Support: 0.005332622317024397
Confidence: 0.3773584905660377
Lift: 3.840659481324083
```

Chapter 2

K-NEAREST NEIGHBOUR

2.1 Objectives:

To learn the basics of KNN Algorithm and implementing it.

2.1 Description:

KNN can be used for both classification and regression predictive problems. However, it is more
widely used in classification problems in the industry. To evaluate any technique we generally
look at 3 important aspects:
1. Ease to interpret output
2. Calculation time
3. Predictive Power

KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of
interpretation and low calculation time.

2.1.1 Steps of KNN Algorithm:

We can implement a KNN model by following the below steps:
1. Load the data.
2. Initialize the value of k.
3. For getting the predicted class, iterate from 1 to total number of training data points.
    a. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    b. Sort the calculated distances in ascending order based on distance values
    c. Get top k rows from the sorted array
4. Get the most frequent class of these rows
5. Return the predicted class

In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming
majority vote between the K most similar instances to a given "unseen" observation.
Similarity is defined according to a distance metric between two data points. A popular one is the Euclidean
distance method. Other methods are Manhattan, Minkowski, and Hamming distance methods.
For categorical variables, the hamming distance must be used.

2.3 Implementation of KNN with Sci-kit Learn:

Dataset: The dataset diabetes.csv is used to implement KNN which contains information about
different indications of diabetes
Goal: Given the diabetes.csv dataset we have to find out for the information of the

indications whether the outcome is having diabetes or not. If Outcome is 1 then it means to have diabetes and otherwise not having diabetes.

2.4 Source Code:

In [43]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

np.set_printoptions(precision=2)


data = pd.read_csv('diabetes.csv')
data.head()
```

Out[43]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [45]:
```
X=data[['Pregnancies','Glucose','BloodPressure','SkinThickness','DiabetesPedigreeFunction','Age']]
y = data['Outcome']
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [5]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train_normalized, y_train)


train_score=knn.score(X_train_normalized, y_train)
test_score=knn.score(X_test_normalized, y_test)

print("Score of training dataset = ",train_score)
print("Score of test dataset = ",test_score)
```

```
Score of training dataset =  0.8055555555555556
Score of test dataset =  0.71875
```

In [6]:
```
knn.fit(X,y)
y_pred=knn.predict(X)

data['Predicted Outcome']=y_pred
data.head()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Predicted Outcome
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	0
2	8	183	64	0	0	23.3	0.672	32	1	1
3	1	89	66	23	94	28.1	0.167	21	0	0
4	0	137	40	35	168	43.1	2.288	33	1	1

Chapter 3

DECISION TREE

3.1 Objectives:

To learn the basics of Decision Tree Algorithm and implementing it.

3.2 Description:

Decision tree algorithm falls under the category of supervised learning. They can be used to solve
both regression and classification problems. Decision tree uses the tree representation to solve the
problem in which each leaf node corresponds to a class label and attributes are represented on the
internal node of the tree. We can represent any boolean function on discrete attributes using the
decision tree.

3.2.1 How does the Decision Tree algorithm work:

• At the beginning, we consider the whole training set as the root.
• Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
• On the basis of attribute values records are distributed recursively.
• We use statistical methods for ordering attributes as root or the internal node.

In Decision Tree the major challenge is to identification of the attribute for the root node in each
level. This process is known as the attribute selection. We have two popular attribute selection
measures:
1. Information Gain
2. Gini Index

3.2.2 Steps of Decision Tree Algorithm:

A general algorithm for a decision tree can be described as follows:

1. Pick the best attribute/feature. The best attribute is one which best splits or separates the data.
2. Ask the relevant question.
3. Follow the answer path.
4. Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets.

3.3 Implementation of Decision Tree Algorithm with Sci-kit Learn:

Dataset: The dataset diabetes.csv is used to implement Decision Tree algorithm which

contains information about
different indications of diabetes

Goal: Given the diabetes.csv dataset we have to find out for the information of the indications whether the outcome is having diabetes or not. If Outcome is 1 then it means to have diabetes and otherwise not having diabetes.

3.4 Source Code:

In [1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

data = pd.read_csv('diabetes.csv')
data.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [2]:
```
X=data[['Pregnancies','Glucose','BloodPressure','SkinThickness','DiabetesPedigreeFunction','Age']]
y = data['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier().fit(X_train, y_train)


train_score=clf.score(X_train, y_train)
test_score=clf.score(X_test, y_test)

print("Score of training dataset = ",train_score)
print("Score of test dataset = ",test_score)
```

```
Score of training dataset =  1.0
Score of test dataset =  0.6927083333333334
```

In [3]:
```
clf.fit(X,y)
y_pred=clf.predict(X)

data['Predicted Outcome']=y_pred
data
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Predicted Outcome
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	0
2	8	183	64	0	0	23.3	0.672	32	1	1
3	1	89	66	23	94	28.1	0.167	21	0	0
4	0	137	40	35	168	43.1	2.288	33	1	1
5	5	116	74	0	0	25.6	0.201	30	0	0
6	3	78	50	32	88	31.0	0.248	26	1	1
7	10	115	0	0	0	35.3	0.134	29	0	0
8	2	197	70	45	543	30.5	0.158	53	1	1
9	8	125	96	0	0	0.0	0.232	54	1	1
10	4	110	92	0	0	37.6	0.191	30	0	0
11	10	168	74	0	0	38.0	0.537	34	1	1
12	10	139	80	0	0	27.1	1.441	57	0	0
13	1	189	60	23	846	30.1	0.398	59	1	1
14	5	166	72	19	175	25.8	0.587	51	1	1
15	7	100	0	0	0	30.0	0.484	32	1	1
16	0	118	84	47	230	45.8	0.551	31	1	1

Chapter 4

KMEANS CLUSTERING

4.1 Objectives:

To learn the basics of K-means clustering and implementing it.

4.2 Description:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.
Typically, unsupervised algorithms make inferences from datasets using only input vectors without
referring to known, or labelled, outcomes.
A cluster refers to a collection of data points aggregated together because of certain similarities.
A target number k, which refers to the number of centroids we need in the dataset. A centroid is
the imaginary or real location representing the center of the cluster.
Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.
In other words, the K-means algorithm identifies k number of centroids, and then allocates every
data point to the nearest cluster, while keeping the centroids as small as possible.
The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.
How the K-means algorithm works
To process the learning data, the K-means algorithm in data mining starts with a first group of
randomly selected centroids, which are used as the beginning points for every cluster, and then
performs iterative (repetitive) calculations to optimize the positions of the centroids
It halts creating and optimizing clusters when either:
• The centroids have stabilized — there is no change in their values because the clustering has
been successful.
• The defined number of iterations has been achieved.

4.2.1 Application of K-means Algorithm:

This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:

- Behavioral segmentation:
    - Segment by purchase history
    - Segment by activities on application, website, or platform
    - Define personas based on interests
    - Create profiles based on activity monitoring

- Inventory categorization:
    - Group inventory by sales activity
    - Group inventory by manufacturing metrics
- Sorting sensor measurements:
    - Detect activity types in motion sensors
    - Group images
    - Separate audio
    - Identify groups in health monitoring
- Detecting bots or anomalies:
    - Separate valid activity groups from bots
    - Group valid activity to clean up outlier detection

In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

4.2.2 Steps of K-means Algorithm:

The *K*-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters *K* and the data set. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the *K* centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm iterates between steps one and two until a criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

4.2.3 Choosing the value of K:

The algorithm described above finds the clusters and data set labels for a particular pre-chosen *K*. To find the number of clusters in the data, the user needs to run the *K*-means clustering algorithm for a range of *K* values and compare the results. In general, there is no method for determining exact value of *K*, but an accurate estimate can be obtained using the following techniques.

One of the metrics that is commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points.

Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of *K* is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K*.

4.3 Implementation of K-means clustering Algorithm with Sci-kit Learn:

Dataset: The dataset diabetes.csv is used to implement K-means algorithm which contains information about
different indications of diabetes

Goal: Given the diabetes.csv dataset we have to find out for the information of the indications whether the outcome is having diabetes or not. If Outcome is 1 then it means to have diabetes and otherwise not having diabetes.

4.4 Source Code:

In [47]:
```
%matplotlib inline
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt

df=pd.read_csv('diabetes.csv')
df.head()
```

Out[47]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [50]:
```
km=KMeans(n_clusters=2)

X=df[['Pregnancies','Glucose','BloodPressure','SkinThickness','DiabetesPedigreeFunction','Age']]
X_normalized = MinMaxScaler().fit(X).transform(X)
```

In [54]:
```
y_pred=km.fit_predict(X_normalized)

df['Cluster']=y_pred
df.head()
```

Out[54]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Cluster
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	0
2	8	183	64	0	0	23.3	0.672	32	1	1
3	1	89	66	23	94	28.1	0.167	21	0	0
4	0	137	40	35	168	43.1	2.288	33	1	0

Chapter 5

LINEAR REGRESSION

5.1 Objectives:

To learn the basics of Linear Regression and implementing it.

5.2 Description:

Regression is a method of modelling a target value based on independent predictors. This method
is mostly used for forecasting and finding out the cause and effect relationship between variables.
Regression techniques mostly differ based on the number of independent variables and the type of
relationship between the independent and dependent variables.
Simple linear regression is a type of regression analysis where the number of independent variables
is one and there is a linear relationship between the independent(x) and dependent(y) variable. The
red line in the above graph is referred to as the best fit straight line. Based on the given data points,
we try to plot a line that models the points the best. The line can be modelled based on the linear
equation shown below.

$y = a_0 + a_1 * x$

The motive of the linear regression algorithm is to find the best values for a_0 and a_1.

The cost function helps us to figure out the best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values for a_0 and a_1, we convert this search problem into a minimization problem where we would like to minimize the error
between the predicted value and the actual value.
We choose the above function to minimize. The difference between the predicted values and ground
truth measures the error difference. We square the error difference and sum over all data points and
divide that value by the total number of data points. This provides the average squared error over
all the data points. Therefore, this cost function is also known as the Mean Squared Error (MSE)
function. Now, using this MSE function we are going to change the values of a_0 and a_1 such that
the MSE value settles at the minima.
Gradient descent is a method of updating a_0 and a_1 to reduce the cost function (MSE). The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost. Gradient descent helps us on how to change the values.

5.3 Implementation of Linear Regression with Sci-kit Learn:

Dataset: The dataset headbrain.csv is used to implement Linear Regression algorithm which contains information about different factors on which the size of the brain differs.
Goal: Given the headbrain.csv dataset we have to find out different sizes of the brain for different values of the features given.

5.4 Source Code:

In [25]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (20.0,10.0)

#Reading data
data = pd.read_csv('headbrain.csv')
print(data.shape)
```

```
(237, 4)
```

In [26]:
```
data.head()
```

Out[26]:

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

In [27]:
```
X=data['Head Size(cm^3)'].values
Y=data['Brain Weight(grams)'].values

mean_x=np.mean(X)
mean_y=np.mean(Y)

l=len(X)

num = 0
den = 0
for i in range(l):
    x_diff =(X[i]-mean_x)
    y_diff =(Y[i]-mean_y)
    num+=x_diff*y_diff
    den+=x_diff**2

m=num/den
c=mean_y - (m*mean_x)

print("Gradient: ",m)
print("Intercept: ",c)
```

```
Gradient:  0.26342933948939945
Intercept:  325.57342104944223
```

In [28]:
```
sst = 0
ssr = 0
for i in range(l):
    pred_y=m*X[i]+c
    sst+=(Y[i]-mean_y)**2
    ssr+=(Y[i]-pred_y)**2
r2=1-(ssr/sst)

print("R2 Score : ",r2)
```

```
R2 Score :  0.6393117199570003
```

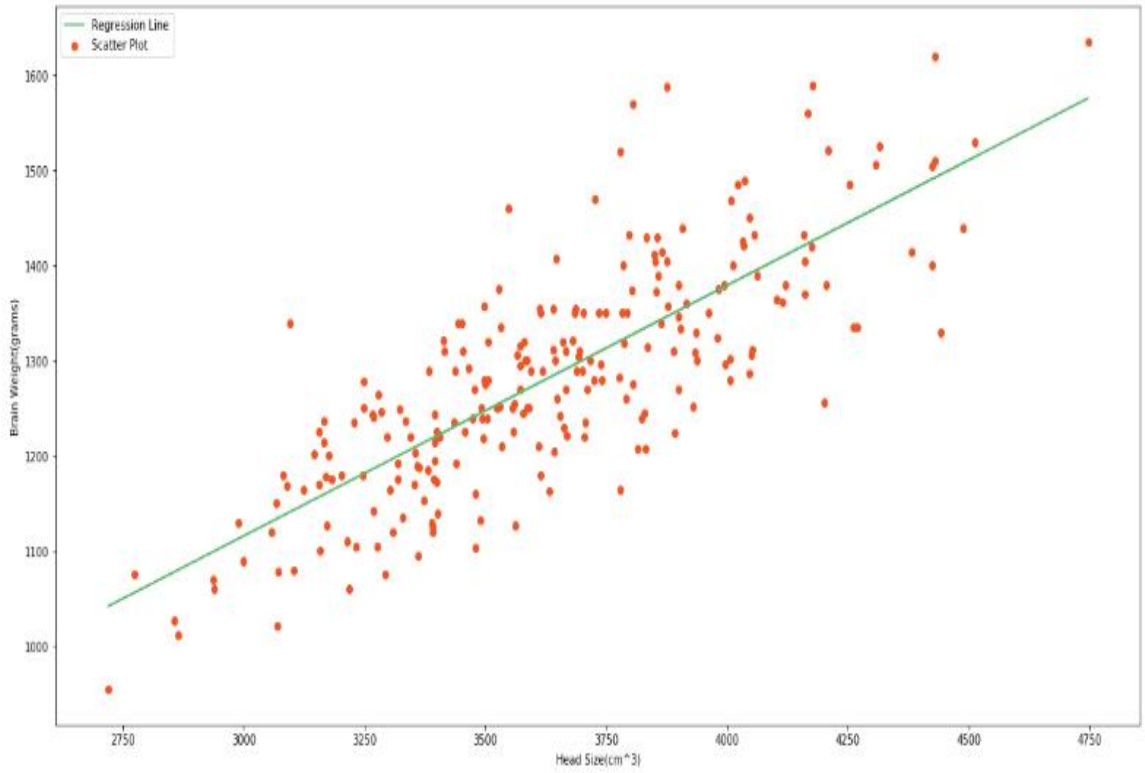
In [29]:
```
#max_x = np.max(X)+100
#min_x = np.max(Y)-100

#x=np.linspace(min_x,max_x,1000)
y=m*X+c



plt.plot(X,y,color = '#58b970', label='Regression Line')
plt.scatter(X,Y,color = '#ef5423', label='Scatter Plot')

plt.xlabel('Head Size(cm^3)')
plt.ylabel('Brain Weight(grams)')
plt.legend()
plt.show()
```



In [30]:

```
print("Actual Brain Weight(grams) ----> Predicted Brain Weight(grams)\n")
n = len(Y)
for i in range(n):
    y_pred=m*X[i]+c
    print("             "+str(Y[i])+"          ---->    "+str(y_pred))
```

```
Actual Brain Weight(grams) ----> Predicted Brain Weight(grams)

             1530          ---->    1514.1666008256125
             1297          ---->    1310.2722920608173
             1335          ---->    1448.0458366137732
             1282          ---->    1320.546036300904
             1590          ---->    1425.9177720966638
             1300          ---->    1269.9676031189392
             1400          ---->    1322.6534710168191
             1255          ---->    1263.118440292215
             1355          ---->    1277.3436246246424
             1375          ---->    1374.549050896231
             1340          ---->    1232.5606369114446
             1380          ---->    1377.4467736306142
             1355          ---->    1284.4562167908562
             1522          ---->    1434.0840816208351
             1208          ---->    1335.034649972821
             1405          ---->    1346.6255409103546
             1358          ---->    1246.785821243872
```

In [ ]:

Discussion:

In this sessional we have come to know about the implementation of different machine learning algorithms. I have collected datasets from different sources and implemented the algorithms on them. The codes are implemented using Python Programming language. Here I have used Anaconda to implement our code. I hope that would be able to improve myself more to implement the algorithms more efficiently. I am grateful to my teachers for helping me complete the sessional with greater knowledge.