CHITTAGONG UNIVERSITY OF ENGINEERING & TECHNOLOGY

DEPERTMENT OF COMPUTER SCIENCE & ENGINEERING

A Study on the Depression Level of Undergraduate Students

By

Alamgir Hossain
Dept. of CSE,CUET

Supervised By:

Dr. Md. Iqbal Hasan Sarker

Associate Professor

Dept. of CSE, CUET

October 4, 2021

Contents

Abstract

Everyone experiences times when they feel a little bit gloomy. Being human, it is a very normal part of our life. Depression, however, is a medical condition that is quite different from everyday moodiness. Several different types of depression are dependent on how an individual's depression symptoms manifest themselves. Depression s a very complex subject and offers many possible topics to focus on. Depression symptoms may vary in severity or in what is causing them. Since depression affects the mental state, the patient will find it difficult to communicate his/her condition to the doctor. Nowadays, a vast proportion of undergraduate students in various universities all over the world are observed to be depressed in various ways. In this report, We focused on the attributes that can make an academic undergraduate depressed. So we built a corpus containing the attributes and corresponding feeling/mood made by those attributes. Doctors must generally rely upon the patient's set of symptoms and what they can observe about him during their examination to make a diagnosis. But, they realize that it is not only a matter of medical tests but various factors that may lead him/her to Depression. So, our corpus was decorated considering factors that may affect someone's mood. This report also reviews several depression detection systems that were applied on the corpus to recognize one's state of mood.

1 Introduction

We applied several Machine Learning techniques to our built dataset. The Algorithms are:

1. Apriori Algorithm
2. Decision Tree Algorithm
3. K Means Clustering Algorithm
4. Linear Regression
5. Artificial Neural Network

1.1 Apriori Algorithm

Apriori Algorithm is designed to work on the databases that contain transactions. This algorithm was given by R. Agrawal and Srikant in the year 1994. and is designed to work on the databases that contain transactions. The algorithm uses frequent itemsets to generate association rules. Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset. Suppose there are the two transactions: A= 1,2,3,4,5, and B= 2,3,7, in these two transactions, 2 and 3 are the frequent itemsets. The association rule learning is one of the very important concepts of machine learning. For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. So, to measure the associations between thousands of data items, there are several metrics. Some of them are given below:

1. **Support:** It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$Support(X) = \frac{Frequency(X)}{T} \tag{1}$$

2. **Confidence:** It is the ratio of the transaction that contains X and Y itemsets to the number of records that contain X.

$$Confidence = \frac{Frequency(X, Y)}{Frequency(X)} \tag{2}$$

3. **Lift:** It is the strength of any rule, which can be defined as below formula:

$$Lift = \frac{Support(X, Y)}{Support(X).Support(Y)} \tag{3}$$

1.2 Decision Tree Algorithm

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example. Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

1.3 K Means Clustering Algorithm

K Means Clustering is an unsupervied Learning approach. One of the most straightforward tasks we can perform on a data set without labels is to find groups of data in our dataset which are similar to one another – what we call clusters. K-Means is one of the most popular "clustering" algorithms. K-means stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid. K-Means finds the best centroids by alternating between assigning data points to clusters based on the current centroids chosing centroids (points which are the center of a cluster) based on the current assignment of data points to clusters.

1.4 Linear Regression

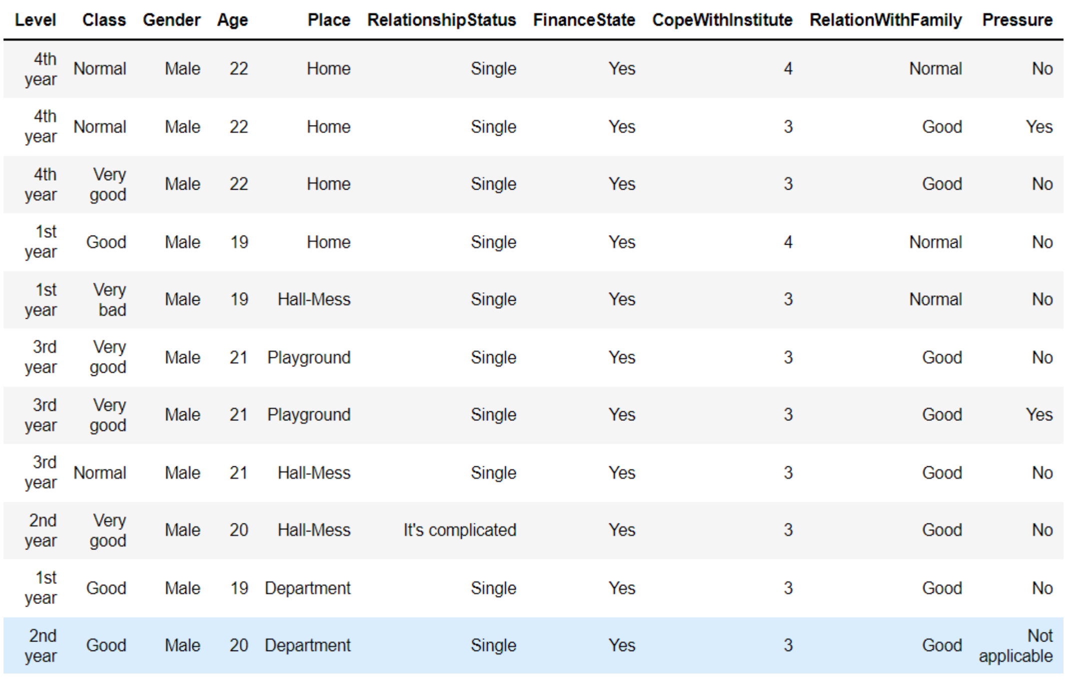
One of the most common and comprehensive statistical and machine learning algorithms are linear regression. Linear regression is used to find a linear relationship between one or more predictors. The linear regression has two types: simple regression and multiple regression (MLR). This paper discusses various works by different researchers on linear regression and polynomial regression and compares their performance using the best approach to optimize prediction and precision. Almost all of the articles analyzed in this review is focused on datasets; in order to determine a model's efficiency, it must be correlated with the actual values obtained for the explanatory variables. It is a supervised machine learning algorithm. It tries to find out the best linear relationship that describes the data you have. It assumes that there exists a linear relationship between a dependent variable and independent variable(s). The value of the dependent variable of a linear regression model is a continuous value i.e. real numbers.

1.5 Artificial Neural Network

Artificial neural networks (ANN) are constructed to simulate processes of the central nervous system of higher creatures. An ANN consists of a set of processing units (nodes) that simulate neurons and are interconnected via a set of "weights" (analogous to synaptic connections in the nervous system) in a way that allows signals to travel through the network in parallel. The nodes (neurons) are simple computing elements. They accumulate input from other neurons employing a weighted sum. If a certain threshold is reached the neuron sends information to all other connected neurons otherwise it remains quiescent. One major difference compared with traditional statistical or rule-based systems is the learning aptitude of an ANN. At the very beginning of a training process, an ANN contains no explicit information. Then a large number of cases with a known outcome are presented to the system and the weights of the inter-neuronal connections are changed by a training algorithm designed to minimize the total error of the system. A trained network has extracted rules that are represented by the matrix of the weights between the neurons.

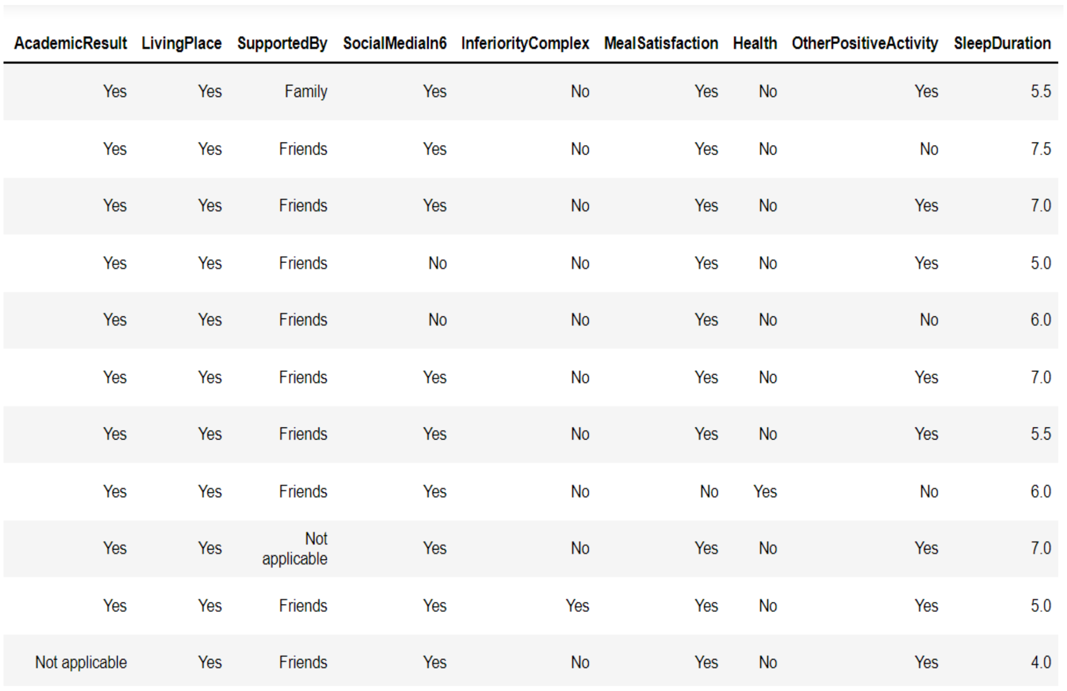
2 Dataset Description

We accumulated our data manually through a google form. We have five classed in our dataset. They are **Very good**, **Good**, **Normal**, **Bad**,**and Very Bad**. We included 22 factors that may affect the feeling of an undergraduate. It turns out that one can be depressed for many reasons. For example, some academic factors may lead one to depression or happiness. But one may have some family issues including the academic factors that are affecting him/her very much. Someone could be supportless while he is going through some tragedy. Someone may visit a place and that makes him/her happy. Considering all these scenarios we put 21 factors that are describing one's steps to feel very good/good/normal/bad/very bad. Overall 3000 samples are put to build the corpus. The samples were taken at various time and in various moods in various situations. The sample of our corpus is shown In figure 2



Level	Class	Gender	Age	Place	RelationshipStatus	FinanceState	CopeWithInstitute	RelationWithFamily	Pressure
4th year	Normal	Male	22	Home	Single	Yes	4	Normal	No
4th year	Normal	Male	22	Home	Single	Yes	3	Good	Yes
4th year	Very good	Male	22	Home	Single	Yes	3	Good	No
1st year	Good	Male	19	Home	Single	Yes	4	Normal	No
1st year	Very bad	Male	19	Hall-Mess	Single	Yes	3	Normal	No
3rd year	Very good	Male	21	Playground	Single	Yes	3	Good	No
3rd year	Very good	Male	21	Playground	Single	Yes	3	Good	Yes
3rd year	Normal	Male	21	Hall-Mess	Single	Yes	3	Good	No
2nd year	Very good	Male	20	Hall-Mess	It's complicated	Yes	3	Good	No
1st year	Good	Male	19	Department	Single	Yes	3	Good	No
2nd year	Good	Male	20	Department	Single	Yes	3	Good	Not applicable

Figure 1: Sample Dataset



AcademicResult	LivingPlace	SupportedBy	SocialMediaIn6	InferiorityComplex	MealSatisfaction	Health	OtherPositiveActivity	SleepDuration
Yes	Yes	Family	Yes	No	Yes	No	Yes	5.5
Yes	Yes	Friends	Yes	No	Yes	No	No	7.5
Yes	Yes	Friends	Yes	No	Yes	No	Yes	7.0
Yes	Yes	Friends	No	No	Yes	No	Yes	5.0
Yes	Yes	Friends	No	No	Yes	No	No	6.0
Yes	Yes	Friends	Yes	No	Yes	No	Yes	7.0
Yes	Yes	Friends	Yes	No	Yes	No	Yes	5.5
Yes	Yes	Friends	Yes	No	No	Yes	No	6.0
Yes	Yes	Not applicable	Yes	No	Yes	No	Yes	7.0
Yes	Yes	Friends	Yes	Yes	Yes	No	Yes	5.0
Not applicable	Yes	Friends	Yes	No	Yes	No	Yes	4.0

Figure 2: Sample Dataset

3 Methodology & Implementation

3.1 Apriori Algorithm

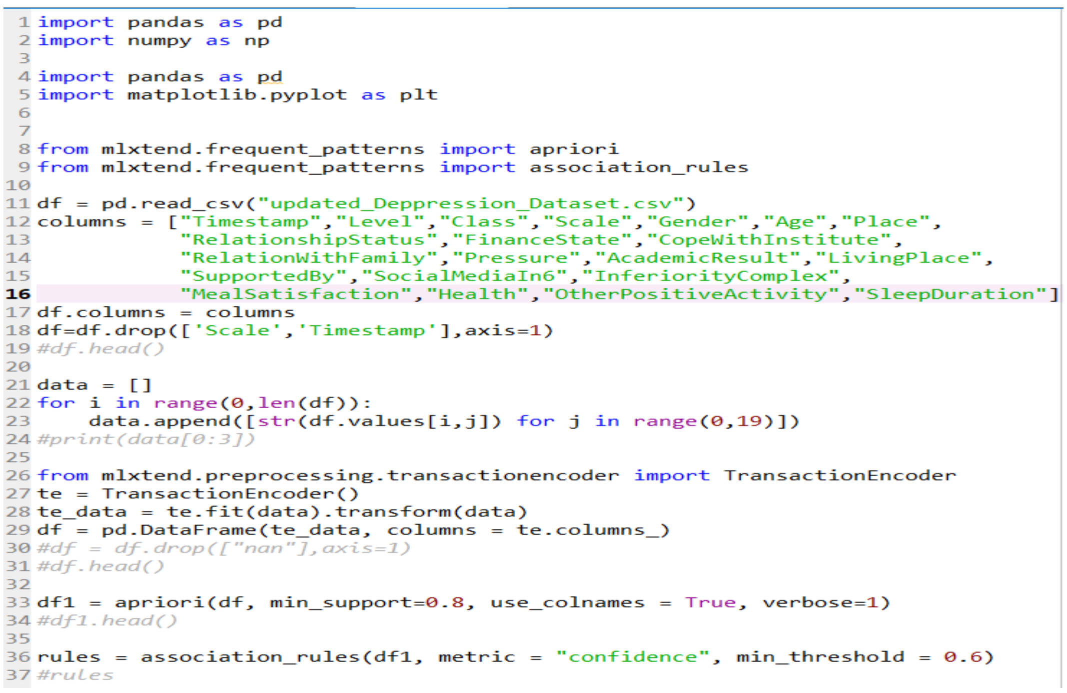
This was done within three steps. At first, the dataset was preprocessed by avoiding NULL values. The attribute was the mixed type of string and float numbers. So, we encoded the data. Then we apply the Apriori model on our encoded data as shown in figure 3.

3.1.1 Source Code

The overall source code is as follows:

3.1.2 Outcome

The frequent itemsets was gained by running on our encoded dataset, as shown in figure 4 Figure 5 shows that, the association rules for the encoded dataset were found on the basis of the Confidence metric. The minimum threshold value was set at 0.60.



```python
import pandas as pd
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt


from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

df = pd.read_csv("updated_Deppression_Dataset.csv")
columns = ["Timestamp","Level","Class","Scale","Gender","Age","Place",
           "RelationshipStatus","FinanceState","CopeWithInstitute",
           "RelationWithFamily","Pressure","AcademicResult","LivingPlace",
           "SupportedBy","SocialMediaIn6","InferiorityComplex",
           "MealSatisfaction","Health","OtherPositiveActivity","SleepDuration"]
df.columns = columns
df=df.drop(['Scale','Timestamp'],axis=1)
#df.head()

data = []
for i in range(0,len(df)):
    data.append([str(df.values[i,j]) for j in range(0,19)])
#print(data[0:3])

from mlxtend.preprocessing.transactionencoder import TransactionEncoder
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data, columns = te.columns_)
#df = df.drop(["nan"],axis=1)
#df.head()

df1 = apriori(df, min_support=0.8, use_colnames = True, verbose=1)
#df1.head()

rules = association_rules(df1, metric = "confidence", min_threshold = 0.6)
#rules
```

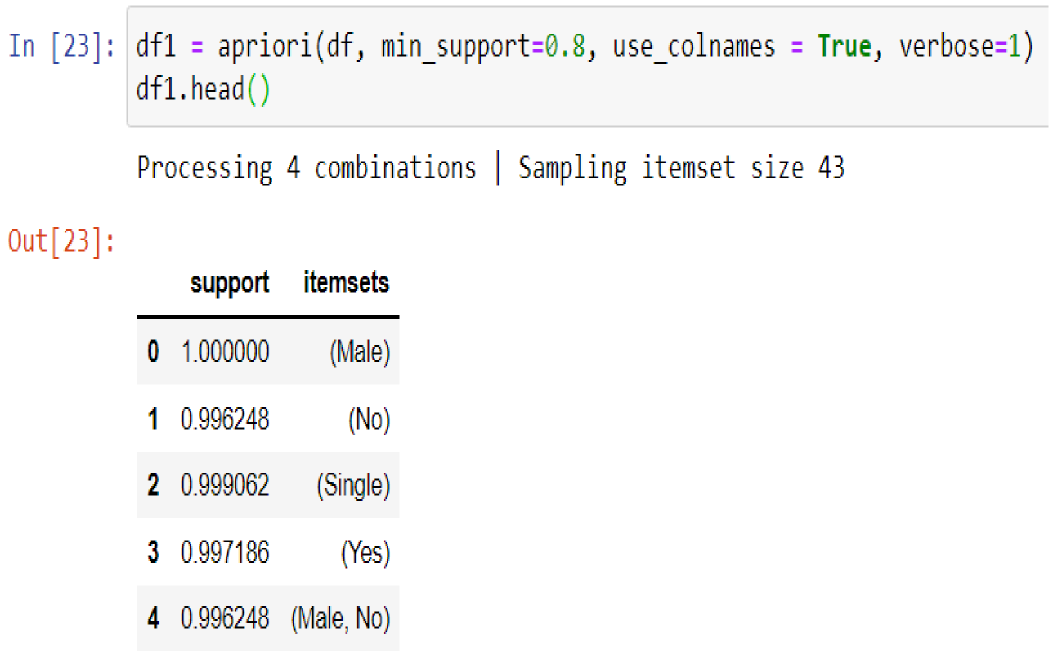
Figure 3: Apriori Model

3.2 Decision Tree Algorithm

Entropy is the main concept of this algorithm, which helps determine a feature or attribute that gives maximum information about a class is called Information gain or ID3 algorithm. By using this method, we can reduce the level of entropy from the root node to the leaf node. Mathematical Formula :

$$E(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

'p', denotes the probability of E(S), which denotes the entropy. The feature or attribute with the highest ID3 gain is used as the root for the splitting. It begins with the original set S as the root node. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Entropy(H) and Information gain(IG) of this attribute. It then selects the attribute which has the smallest Entropy or Largest Information gain. The set S is then split by the selected attribute to produce a subset of the data. The algorithm continues to recur on each subset, considering only attributes never selected before.



```
In [23]: df1 = apriori(df, min_support=0.8, use_colnames = True, verbose=1)
         df1.head()

         Processing 4 combinations | Sampling itemset size 43

Out[23]:
```

	support	itemsets
0	1.000000	(Male)
1	0.996248	(No)
2	0.999062	(Single)
3	0.997186	(Yes)
4	0.996248	(Male, No)

Figure 4: Frequent Itemsets

3.2.1 Source Code

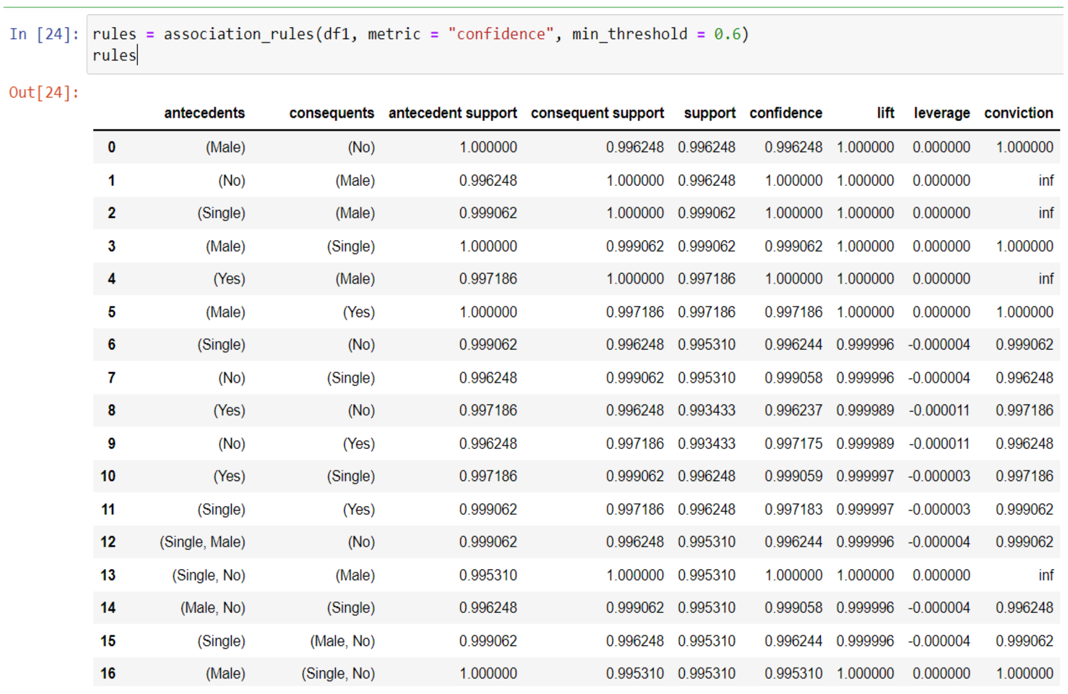
The Decision Tree Classifier was run as in figure 6

3.2.2 Outcome

Here the confusion matrix is shown in figure 7. The Decision Tree Classifier gained an accuracy value of 71.03%.

3.3 K Means Clustering Algorithm

As shown in figure 8, We choose the number of clusters manually by plotting the graph using *seaborn* libraries. The cluster centroid was set randomly. After that, it began iterating until the centroid was fixed to its optimal position. Then all the data points were checked to be in the right cluster. And, finally, we applied(figure 10) the algorithm on our encoded data and predict(figure 9) which cluster the data points belong to.



```
In [24]: rules = association_rules(df1, metric = "confidence", min_threshold = 0.6)
         rules
```

Out[24]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Male)	(No)	1.000000	0.996248	0.996248	0.996248	1.000000	0.000000	1.000000
1	(No)	(Male)	0.996248	1.000000	0.996248	1.000000	1.000000	0.000000	inf
2	(Single)	(Male)	0.999062	1.000000	0.999062	1.000000	1.000000	0.000000	inf
3	(Male)	(Single)	1.000000	0.999062	0.999062	0.999062	1.000000	0.000000	1.000000
4	(Yes)	(Male)	0.997186	1.000000	0.997186	1.000000	1.000000	0.000000	inf
5	(Male)	(Yes)	1.000000	0.997186	0.997186	0.997186	1.000000	0.000000	1.000000
6	(Single)	(No)	0.999062	0.996248	0.995310	0.996244	0.999996	-0.000004	0.999062
7	(No)	(Single)	0.996248	0.999062	0.995310	0.999058	0.999996	-0.000004	0.996248
8	(Yes)	(No)	0.997186	0.996248	0.993433	0.996237	0.999989	-0.000011	0.997186
9	(No)	(Yes)	0.996248	0.997186	0.993433	0.997175	0.999989	-0.000011	0.996248
10	(Yes)	(Single)	0.997186	0.999062	0.996248	0.999059	0.999997	-0.000003	0.997186
11	(Single)	(Yes)	0.999062	0.997186	0.996248	0.997183	0.999997	-0.000003	0.999062
12	(Single, Male)	(No)	0.999062	0.996248	0.995310	0.996244	0.999996	-0.000004	0.999062
13	(Single, No)	(Male)	0.995310	1.000000	0.995310	1.000000	1.000000	0.000000	inf
14	(Male, No)	(Single)	0.996248	0.999062	0.995310	0.999058	0.999996	-0.000004	0.996248
15	(Single)	(Male, No)	0.999062	0.996248	0.995310	0.996244	0.999996	-0.000004	0.999062
16	(Male)	(Single, No)	1.000000	0.995310	0.995310	0.995310	1.000000	0.000000	1.000000

Figure 5: Association Rules

3.4 Linear Regression

It is a classifier algorithm that which we can predict the label of data by following linear distance. We made the hypothesis mathematically and used the actual label of the data to check our accuracy.

3.4.1 Source Code

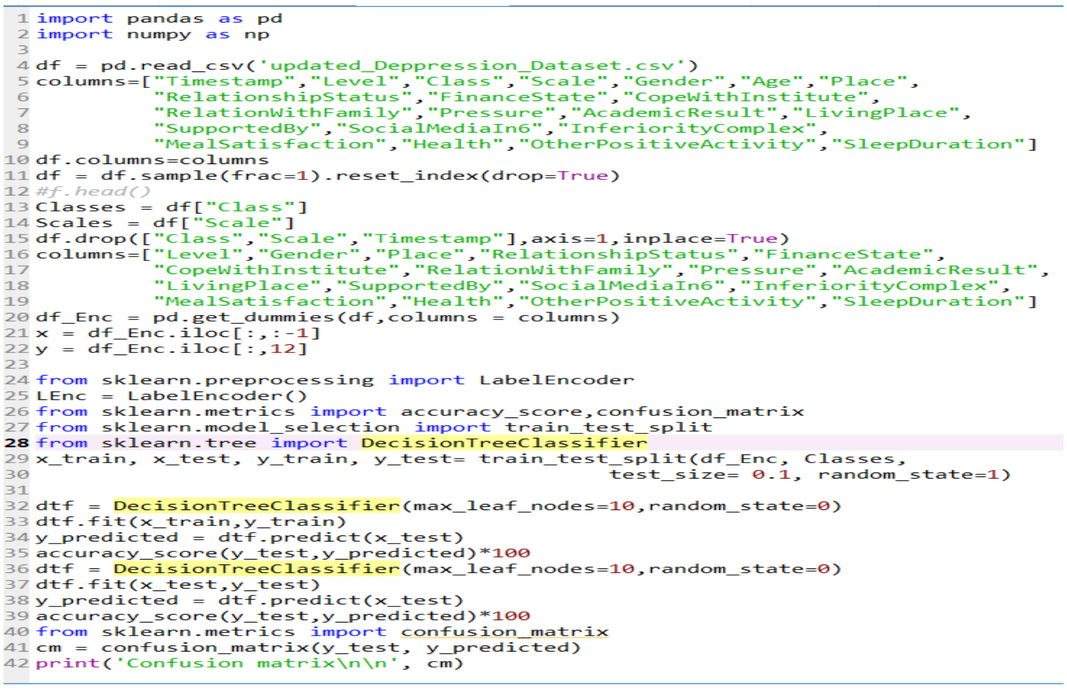
The first part of the code contains the raw model of Linear Regression, shown in figure 11.

This part shows the backend process of predicting new sample of data, in figure 12. The performance measure is presented in figure 13.

3.4.2 Outcome

3.5 Artificial Neural Network

At first, the corpus was encoded and the dummy variables were created. We used Artificial Neural Network from 14 at *Keras* model. We got the accuracy of 61.68%. as shown in figure 15.

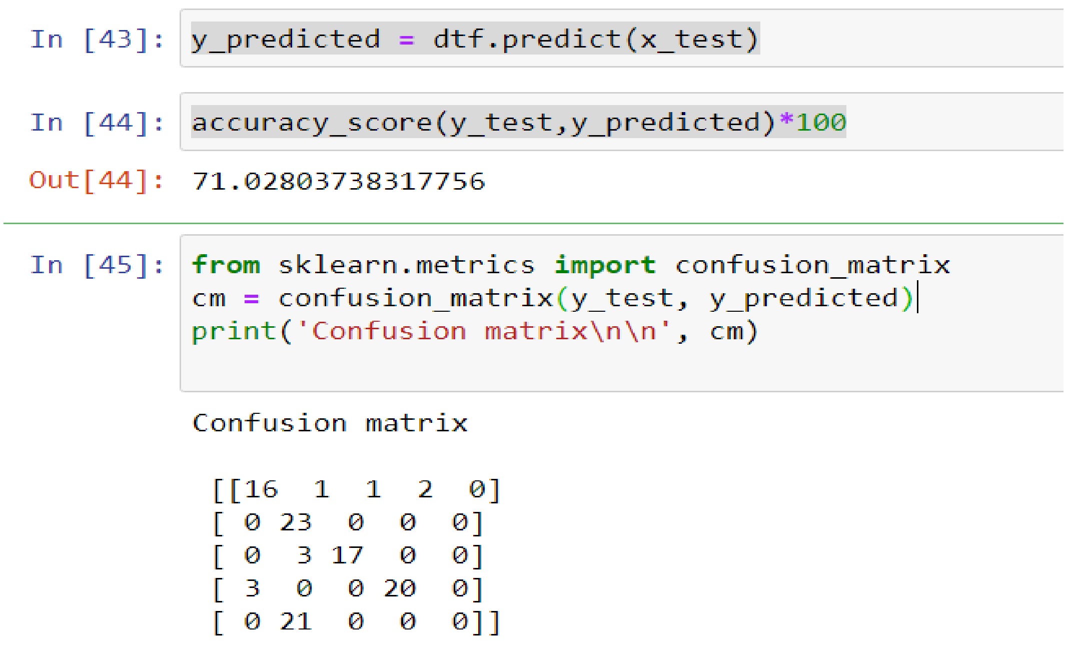


```python
import pandas as pd
import numpy as np

df = pd.read_csv('updated_Deppression_Dataset.csv')
columns=["Timestamp","Level","Class","Scale","Gender","Age","Place",
          "RelationshipStatus","FinanceState","CopeWithInstitute",
          "RelationWithFamily","Pressure","AcademicResult","LivingPlace",
          "SupportedBy","SocialMediaIn6","InferiorityComplex",
          "MealSatisfaction","Health","OtherPositiveActivity","SleepDuration"]
df.columns=columns
df = df.sample(frac=1).reset_index(drop=True)
#f.head()
Classes = df["Class"]
Scales = df["Scale"]
df.drop(["Class","Scale","Timestamp"],axis=1,inplace=True)
columns=["Level","Gender","Place","RelationshipStatus","FinanceState",
          "CopeWithInstitute","RelationWithFamily","Pressure","AcademicResult",
          "LivingPlace","SupportedBy","SocialMediaIn6","InferiorityComplex",
          "MealSatisfaction","Health","OtherPositiveActivity","SleepDuration"]
df_Enc = pd.get_dummies(df,columns = columns)
x = df_Enc.iloc[:,:-1]
y = df_Enc.iloc[:,12]

from sklearn.preprocessing import LabelEncoder
LEnc = LabelEncoder()
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
x_train, x_test, y_train, y_test= train_test_split(df_Enc, Classes,
                                             test_size= 0.1, random_state=1)

dtf = DecisionTreeClassifier(max_leaf_nodes=10,random_state=0)
dtf.fit(x_train,y_train)
y_predicted = dtf.predict(x_test)
accuracy_score(y_test,y_predicted)*100
dtf = DecisionTreeClassifier(max_leaf_nodes=10,random_state=0)
dtf.fit(x_test,y_test)
y_predicted = dtf.predict(x_test)
accuracy_score(y_test,y_predicted)*100
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
print('Confusion matrix\n\n', cm)
```

Figure 6: Decision Tree Classifier

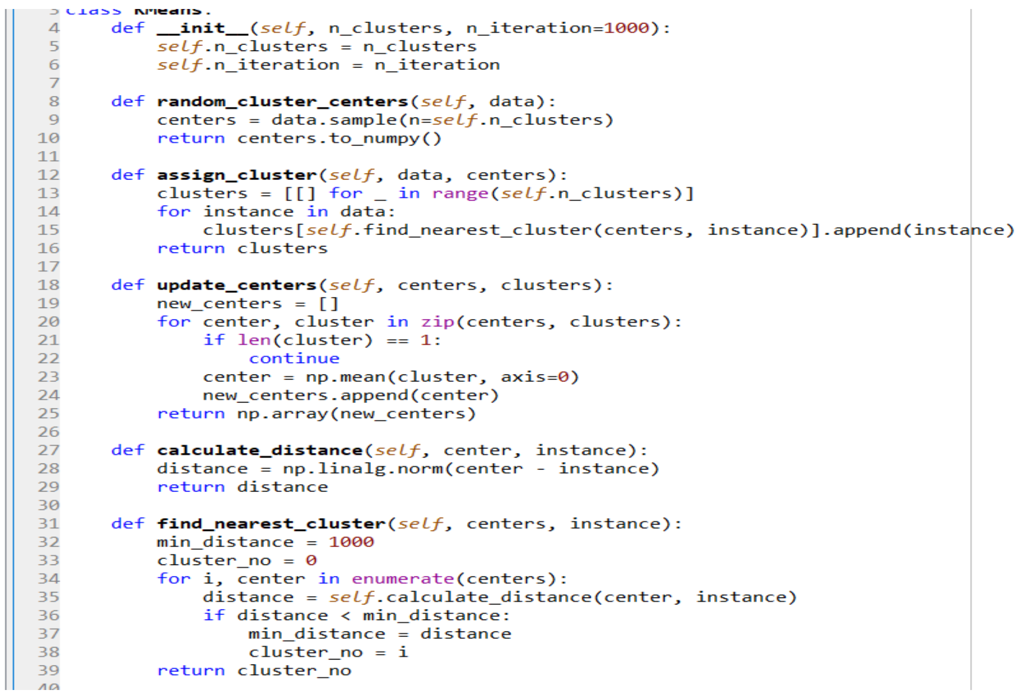
4 Conclusion

This report presents a review of various approaches that have been developed with the aim of analyzing the depression. Thoguh, on earth, people have different mindset. One may fell into depression based on some factors that could be normal to another. So the sample data may vary from person to person. Moreover, we only applied five model to analyze depression. But a topic such as Depression deserve more work that is our future target.



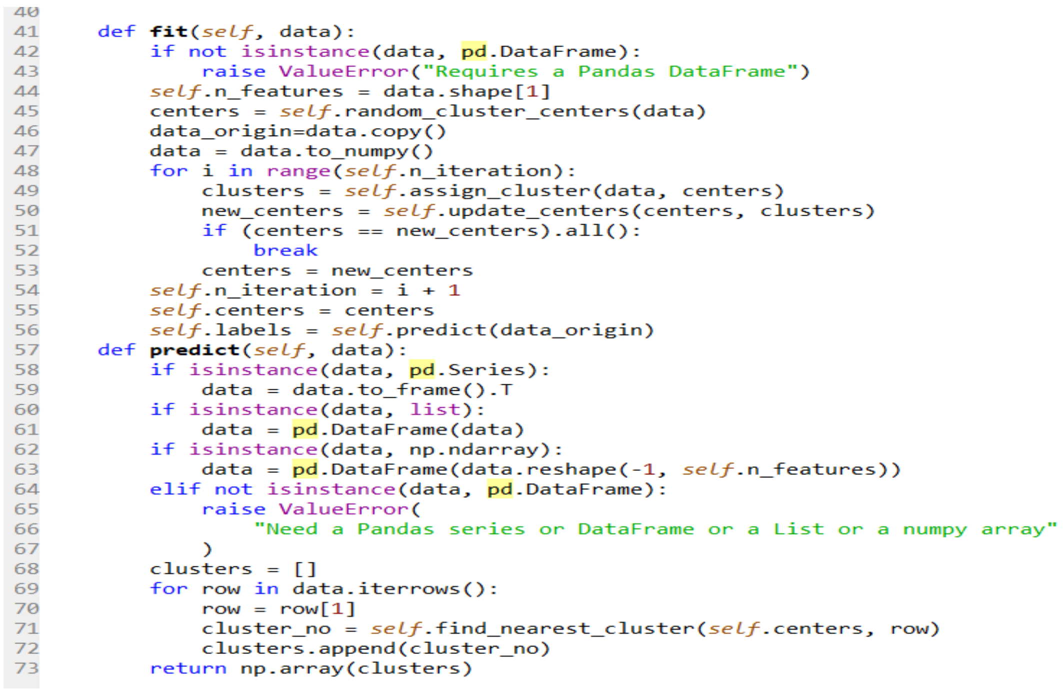
```
In [43]: y_predicted = dtf.predict(x_test)

In [44]: accuracy_score(y_test,y_predicted)*100

Out[44]: 71.02803738317756

In [45]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_predicted)
         print('Confusion matrix\n\n', cm)

         Confusion matrix

          [[16  1  1  2  0]
          [ 0 23  0  0  0]
          [ 0  3 17  0  0]
          [ 3  0  0 20  0]
          [ 0 21  0  0  0]]
```

Figure 7: Confusion Matrix & Accuracy



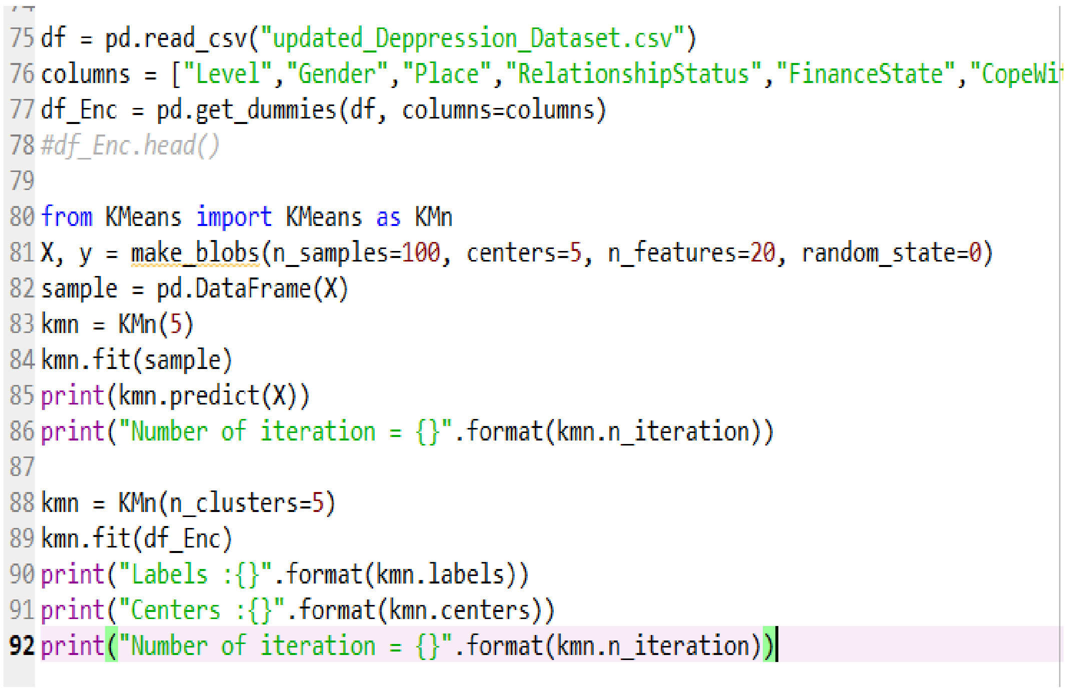
```
class KMeans:
    def __init__(self, n_clusters, n_iteration=1000):
        self.n_clusters = n_clusters
        self.n_iteration = n_iteration

    def random_cluster_centers(self, data):
        centers = data.sample(n=self.n_clusters)
        return centers.to_numpy()

    def assign_cluster(self, data, centers):
        clusters = [[] for _ in range(self.n_clusters)]
        for instance in data:
            clusters[self.find_nearest_cluster(centers, instance)].append(instance)
        return clusters

    def update_centers(self, centers, clusters):
        new_centers = []
        for center, cluster in zip(centers, clusters):
            if len(cluster) == 1:
                continue
            center = np.mean(cluster, axis=0)
            new_centers.append(center)
        return np.array(new_centers)

    def calculate_distance(self, center, instance):
        distance = np.linalg.norm(center - instance)
        return distance

    def find_nearest_cluster(self, centers, instance):
        min_distance = 1000
        cluster_no = 0
        for i, center in enumerate(centers):
            distance = self.calculate_distance(center, instance)
            if distance < min_distance:
                min_distance = distance
                cluster_no = i
        return cluster_no
```

Figure 8: Clustering Process



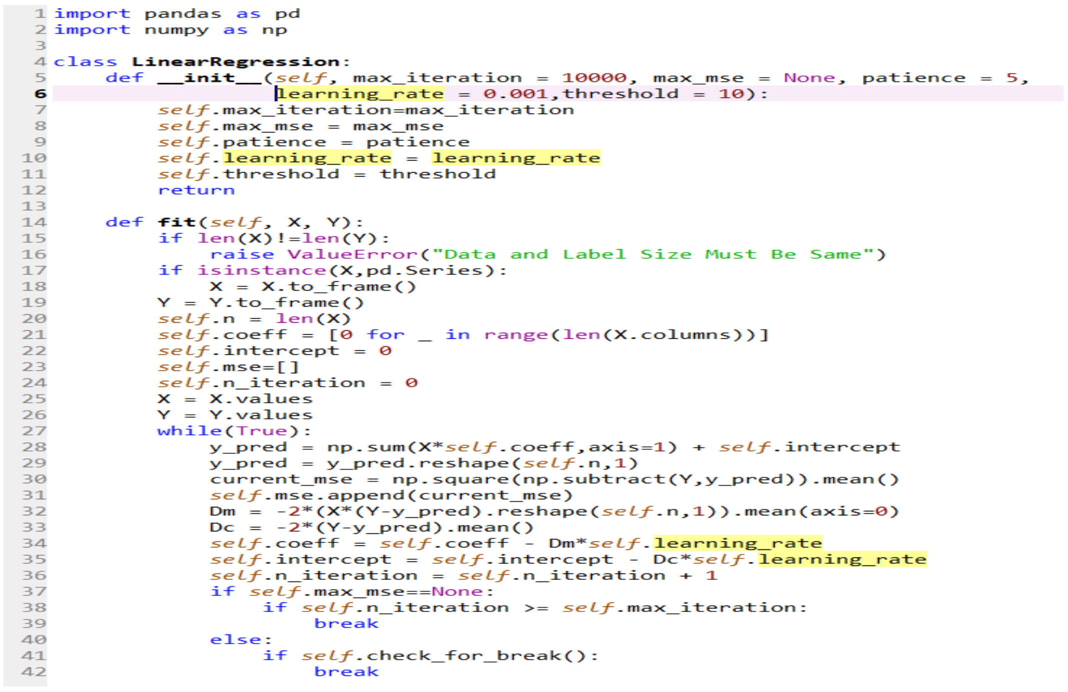
```
    def fit(self, data):
        if not isinstance(data, pd.DataFrame):
            raise ValueError("Requires a Pandas DataFrame")
        self.n_features = data.shape[1]
        centers = self.random_cluster_centers(data)
        data_origin=data.copy()
        data = data.to_numpy()
        for i in range(self.n_iteration):
            clusters = self.assign_cluster(data, centers)
            new_centers = self.update_centers(centers, clusters)
            if (centers == new_centers).all():
                break
            centers = new_centers
        self.n_iteration = i + 1
        self.centers = centers
        self.labels = self.predict(data_origin)
    def predict(self, data):
        if isinstance(data, pd.Series):
            data = data.to_frame().T
        if isinstance(data, list):
            data = pd.DataFrame(data)
        if isinstance(data, np.ndarray):
            data = pd.DataFrame(data.reshape(-1, self.n_features))
        elif not isinstance(data, pd.DataFrame):
            raise ValueError(
                "Need a Pandas series or DataFrame or a List or a numpy array"
            )
        clusters = []
        for row in data.iterrows():
            row = row[1]
            cluster_no = self.find_nearest_cluster(self.centers, row)
            clusters.append(cluster_no)
        return np.array(clusters)
```

Figure 9: Predicting Process



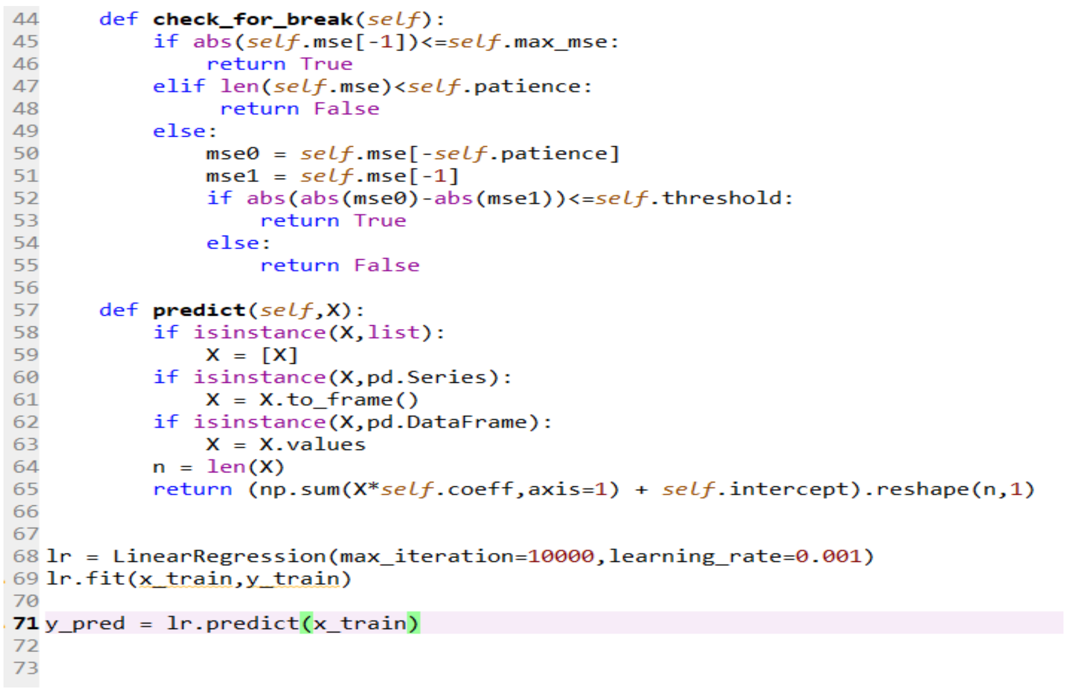
```
df = pd.read_csv("updated_Deppression_Dataset.csv")
columns = ["Level","Gender","Place","RelationshipStatus","FinanceState","CopeWi
df_Enc = pd.get_dummies(df, columns=columns)
#df_Enc.head()

from KMeans import KMeans as KMn
X, y = make_blobs(n_samples=100, centers=5, n_features=20, random_state=0)
sample = pd.DataFrame(X)
kmn = KMn(5)
kmn.fit(sample)
print(kmn.predict(X))
print("Number of iteration = {}".format(kmn.n_iteration))

kmn = KMn(n_clusters=5)
kmn.fit(df_Enc)
print("Labels :{}".format(kmn.labels))
print("Centers :{}".format(kmn.centers))
print("Number of iteration = {}".format(kmn.n_iteration))
```

Figure 10: Applying K Means Clustering Algorithm



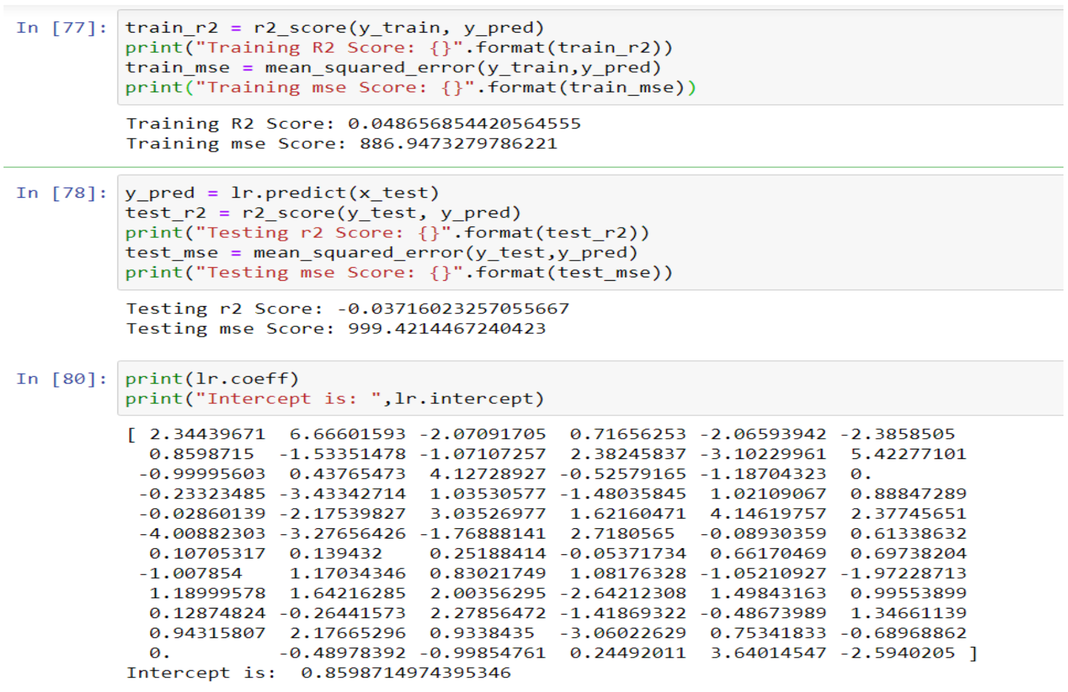
```python
import pandas as pd
import numpy as np

class LinearRegression:
    def __init__(self, max_iteration = 10000, max_mse = None, patience = 5,
                learning_rate = 0.001,threshold = 10):
        self.max_iteration=max_iteration
        self.max_mse = max_mse
        self.patience = patience
        self.learning_rate = learning_rate
        self.threshold = threshold
        return

    def fit(self, X, Y):
        if len(X)!=len(Y):
            raise ValueError("Data and Label Size Must Be Same")
        if isinstance(X,pd.Series):
            X = X.to_frame()
        Y = Y.to_frame()
        self.n = len(X)
        self.coeff = [0 for _ in range(len(X.columns))]
        self.intercept = 0
        self.mse=[]
        self.n_iteration = 0
        X = X.values
        Y = Y.values
        while(True):
            y_pred = np.sum(X*self.coeff,axis=1) + self.intercept
            y_pred = y_pred.reshape(self.n,1)
            current_mse = np.square(np.subtract(Y,y_pred)).mean()
            self.mse.append(current_mse)
            Dm = -2*(X*(Y-y_pred).reshape(self.n,1)).mean(axis=0)
            Dc = -2*(Y-y_pred).mean()
            self.coeff = self.coeff - Dm*self.learning_rate
            self.intercept = self.intercept - Dc*self.learning_rate
            self.n_iteration = self.n_iteration + 1
            if self.max_mse==None:
                if self.n_iteration >= self.max_iteration:
                    break
            else:
                if self.check_for_break():
                    break
```

Figure 11: Linear Regression Model



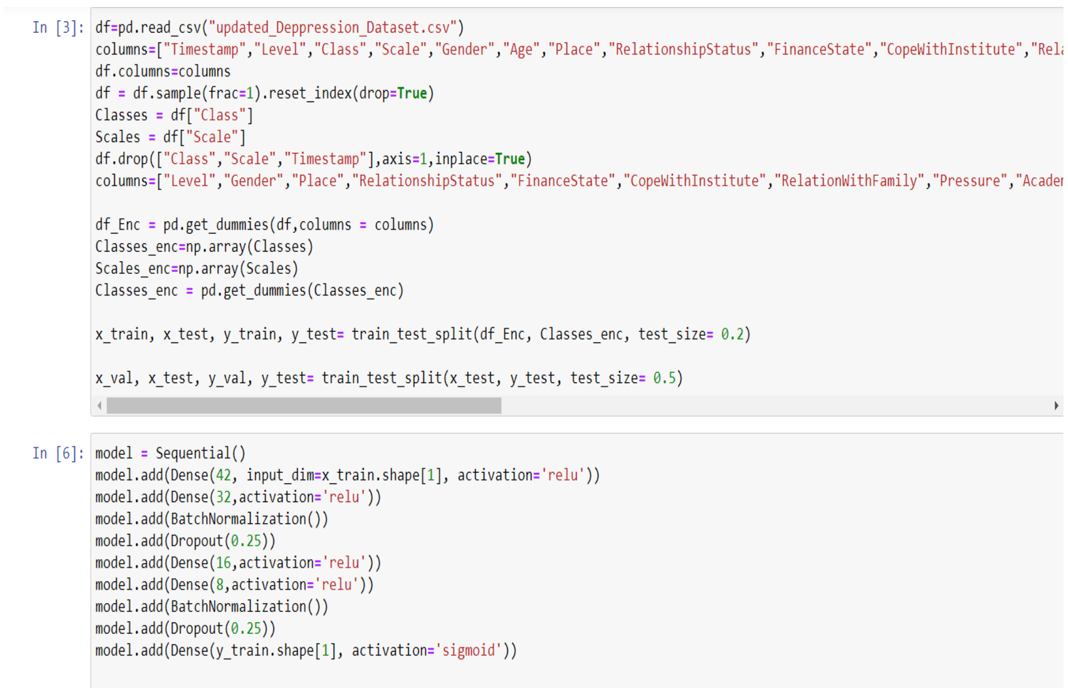
```python
    def check_for_break(self):
        if abs(self.mse[-1])<=self.max_mse:
            return True
        elif len(self.mse)<self.patience:
             return False
        else:
            mse0 = self.mse[-self.patience]
            mse1 = self.mse[-1]
            if abs(abs(mse0)-abs(mse1))<=self.threshold:
                return True
            else:
                return False

    def predict(self,X):
        if isinstance(X,list):
            X = [X]
        if isinstance(X,pd.Series):
            X = X.to_frame()
        if isinstance(X,pd.DataFrame):
            X = X.values
        n = len(X)
        return (np.sum(X*self.coeff,axis=1) + self.intercept).reshape(n,1)


lr = LinearRegression(max_iteration=10000,learning_rate=0.001)
lr.fit(x_train,y_train)

y_pred = lr.predict(x_train)
```

Figure 12: Linear Regression Predictor



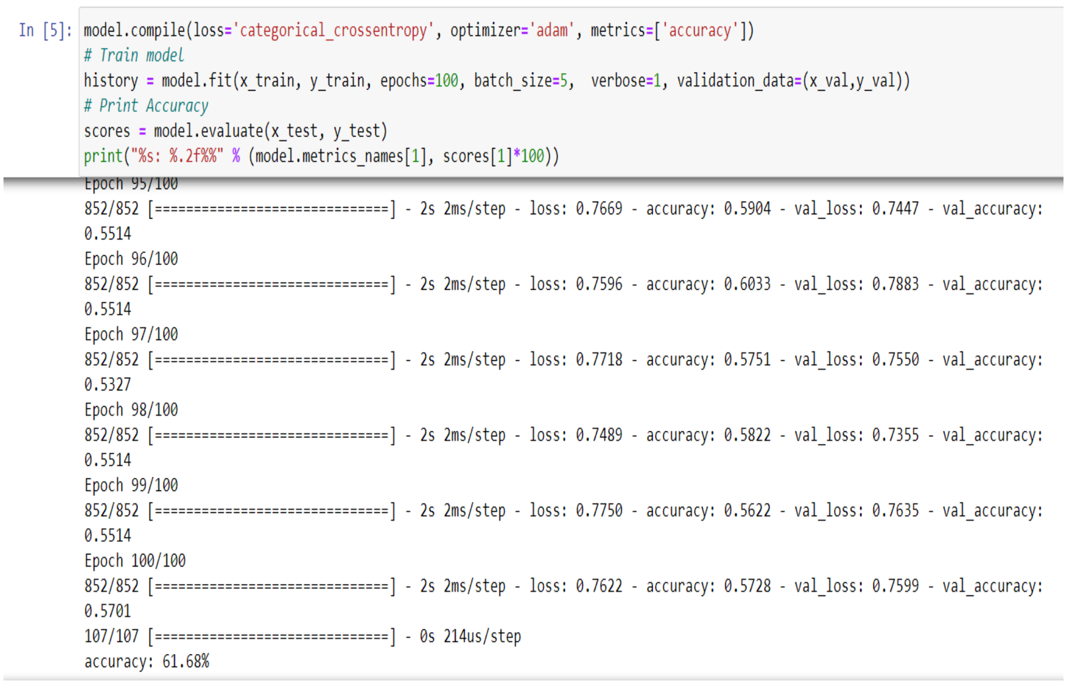
```
In [77]: train_r2 = r2_score(y_train, y_pred)
         print("Training R2 Score: {}".format(train_r2))
         train_mse = mean_squared_error(y_train,y_pred)
         print("Training mse Score: {}".format(train_mse))

         Training R2 Score: 0.048656854420564555
         Training mse Score: 886.9473279786221

In [78]: y_pred = lr.predict(x_test)
         test_r2 = r2_score(y_test, y_pred)
         print("Testing r2 Score: {}".format(test_r2))
         test_mse = mean_squared_error(y_test,y_pred)
         print("Testing mse Score: {}".format(test_mse))

         Testing r2 Score: -0.03716023257055667
         Testing mse Score: 999.4214467240423

In [80]: print(lr.coeff)
         print("Intercept is: ",lr.intercept)

         [ 2.34439671  6.66601593 -2.07091705  0.71656253 -2.06593942 -2.3858505
           0.8598715  -1.53351478 -1.07107257  2.38245837 -3.10229961  5.42277101
          -0.99995603  0.43765473  4.12728927 -0.52579165 -1.18704323  0.
          -0.23323485 -3.43342714  1.03530577 -1.48035845  1.02109067  0.88847289
          -0.02860139 -2.17539827  3.03526977  1.62160471  4.14619757  2.37745651
          -4.00882303 -3.27656426 -1.76888141  2.7180565  -0.08930359  0.61338632
           0.10705317  0.139432    0.25188414 -0.05371734  0.66170469  0.69738204
          -1.007854    1.17034346  0.83021749  1.08176328 -1.05210927 -1.97228713
           1.18999578  1.64216285  2.00356295 -2.64212308  1.49843163  0.99553899
           0.12874824 -0.26441573  2.27856472 -1.41869322 -0.48673989  1.34661139
           0.94315807  2.17665296  0.9338435  -3.06022629  0.75341833 -0.68968862
           0.         -0.48978392 -0.99854761  0.24492011  3.64014547 -2.5940205 ]
         Intercept is:  0.8598714974395346
```

Figure 13: Perfomance Measure for LR Model



```
In [3]: df=pd.read_csv("updated_Deppression_Dataset.csv")
        columns=["Timestamp","Level","Class","Scale","Gender","Age","Place","RelationshipStatus","FinanceState","CopeWithInstitute","Rel
        df.columns=columns
        df = df.sample(frac=1).reset_index(drop=True)
        Classes = df["Class"]
        Scales = df["Scale"]
        df.drop(["Class","Scale","Timestamp"],axis=1,inplace=True)
        columns=["Level","Gender","Place","RelationshipStatus","FinanceState","CopeWithInstitute","RelationWithFamily","Pressure","Acade

        df_Enc = pd.get_dummies(df,columns = columns)
        Classes_enc=np.array(Classes)
        Scales_enc=np.array(Scales)
        Classes_enc = pd.get_dummies(Classes_enc)

        x_train, x_test, y_train, y_test= train_test_split(df_Enc, Classes_enc, test_size= 0.2)

        x_val, x_test, y_val, y_test= train_test_split(x_test, y_test, test_size= 0.5)

In [6]: model = Sequential()
        model.add(Dense(42, input_dim=x_train.shape[1], activation='relu'))
        model.add(Dense(32,activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(0.25))
        model.add(Dense(16,activation='relu'))
        model.add(Dense(8,activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(0.25))
        model.add(Dense(y_train.shape[1], activation='sigmoid'))
```

Figure 14: ANN Model



```
In [5]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        # Train model
        history = model.fit(x_train, y_train, epochs=100, batch_size=5,  verbose=1, validation_data=(x_val,y_val))
        # Print Accuracy
        scores = model.evaluate(x_test, y_test)
        print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 95/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7669 - accuracy: 0.5904 - val_loss: 0.7447 - val_accuracy:
0.5514
Epoch 96/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7596 - accuracy: 0.6033 - val_loss: 0.7883 - val_accuracy:
0.5514
Epoch 97/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7718 - accuracy: 0.5751 - val_loss: 0.7550 - val_accuracy:
0.5327
Epoch 98/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7489 - accuracy: 0.5822 - val_loss: 0.7355 - val_accuracy:
0.5514
Epoch 99/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7750 - accuracy: 0.5622 - val_loss: 0.7635 - val_accuracy:
0.5514
Epoch 100/100
852/852 [==============================] - 2s 2ms/step - loss: 0.7622 - accuracy: 0.5728 - val_loss: 0.7599 - val_accuracy:
0.5701
107/107 [==============================] - 0s 214us/step
accuracy: 61.68%
```

Figure 15: Accuracy of ANN Model