Experiment - 1

Aim-To predict Bicarbonate(ppm) present in the well water of Northwest Texas data via Linear Regression Machine learning moodel.

```
cd /content/drive/MyDrive/Machine Learning/Colab Notebooks/ML
Practicals/1_Practical/Linear regression P1
```

```
/content/drive/MyDrive/Machine Learning/Colab Notebooks/ML
Practicals/1_Practical/Linear regression P1
```

```
ls
```

```
 edcCO2.csv     'Ground Water Survey.csv'
 fruitohms.csv  'Linear regression_1.ipynb'
```

Importing Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
#import data set
dataset = pd.read_csv('Ground Water Survey.csv')
X= dataset.iloc[:,:-1].values
Y= dataset.iloc[:,1].values
dataset.head()
```

```
     X    Y
0  7.6  157
1  7.1  174
2  8.2  175
3  7.5  188
4  7.4  171
```

In the following data

X = pH of well water

Y = Bicarbonate (parts per million) of well water

The data is by water well from a random sample of wells in Northwest Texas. Reference: Union Carbide Technical Report K/UR-1

```
dataset.tail()
```

```
      X    Y
29  8.5   48
30  7.8  147
31  6.7  117
```

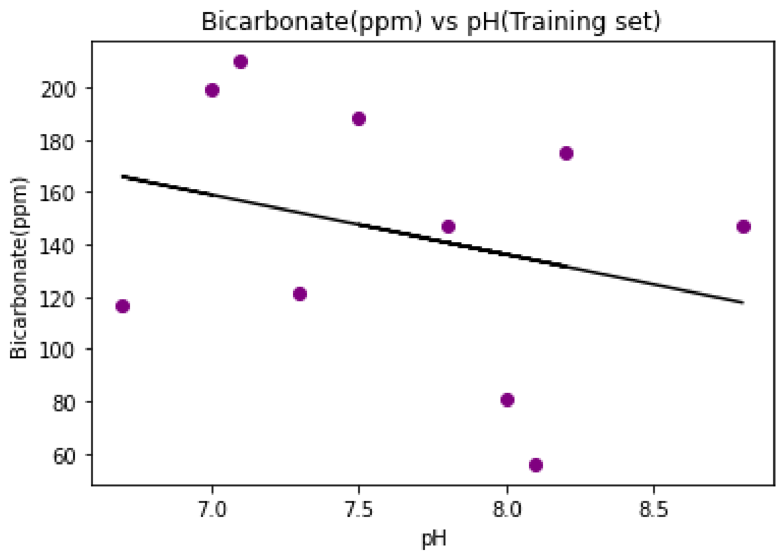
```
32  7.1  182
33  7.3   87
```

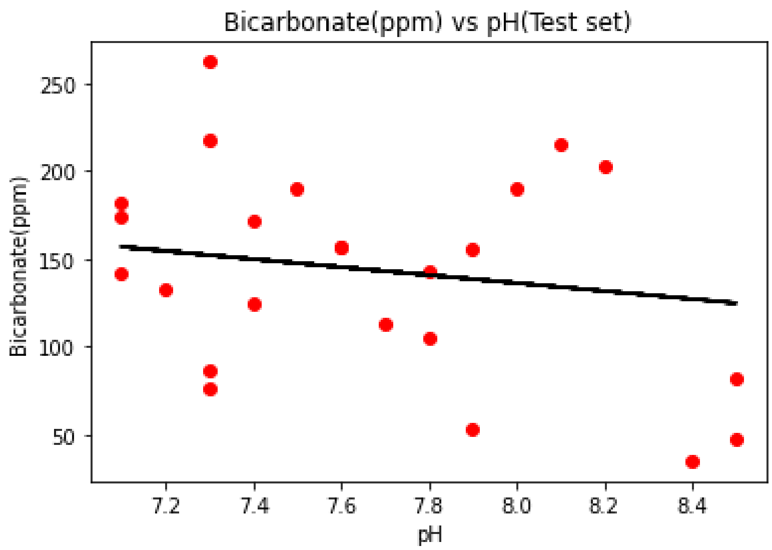
Bicarbonate can be found in water with a pH between 4.3 and 12.3. Above a pH of 8.3, carbonate is also present.

```
#Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size= 0.7)

#Fitting Simple Linear Regression ipynb
#This is called Model
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
##Predicting the test results
Y_pred= regressor.predict(X_test)

#Visualising the training set Results

plt.scatter(X_train, Y_train, color='Purple')
plt.plot(X_train, regressor.predict(X_train), color='black')
plt.title('Bicarbonate(ppm) vs pH(Training set)')
plt.xlabel('pH')
plt.ylabel('Bicarbonate(ppm)')
plt.show()

plt.scatter(X_test, Y_test, color='red')
plt.plot(X_test, regressor.predict(X_test), color='black')
plt.title('Bicarbonate(ppm) vs pH(Test set)')
plt.xlabel('pH')
plt.ylabel('Bicarbonate(ppm)')
plt.show()
```






```
print(regressor.predict([[7.6]]))
```

```
[145.2435247]
```

Now we will perform the prediction of Bicarbonate(ppm) Present in the well water.

```
a=float(input("What is the pH of your well water? "))
print('The Bicarbonate (parts per million) in your well water',
regressor.predict([[a]]))
```

```
What is the pH of your well water? 7.1
The Bicarbonate (parts per million) in your well water [156.6787717]
```

Conclusion- Hence we are able to predict the Bicarbonate(ppm) present in the well water of Northeast Texas by training the Linear Regression model with the Water Survey Dataset.

Experiment – 2

Aim- To predict Coronary Heart Disease using Logistic Regression Classifier

Logistic Regression: The target variable has three or more nominal categories such as predicting the type of Wine. Ordinal Logistic Regression: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5. Model building in Scikit-learn Let's build the diabetes prediction model.

Here, We are going to predict Coronary Heart Disease using Logistic Regression Classifier.

Let's first load the required Coronary Heart Disease dataset using the pandas' read CSV function.

We will download data from the following link:
https://www.kaggle.com/datasets/billbasener/coronary-heart-disease?resource=download

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
cd /content/gdrive/MyDrive/Machine Learning/Colab Notebooks/ML_Practicals/1_Practical/Logistic regression P2
```

```
/content/gdrive/MyDrive/Machine Learning/Colab Notebooks/ML_Practicals/1_Practical/Logistic regression P2
```

```
ls
```

```
CHDdata.csv  CHD_Data.csv  CHDdata.gsheet  Logistic_Regression.ipynb
```

```
import pandas as pd
col_names = ['Systolic BP', 'Tobacco', 'low-density lipoprotein', 'Adiposity', 'Famhist', 'typea', 'Obesity', 'Alcohol', 'Age', 'Chd']
# load dataset
CHD = pd.read_csv("CHD_Data.csv", header=None, names=col_names)
```

Context

The data set CHDdata.csv contains cases of coronary heart disease (CHD) and variables associated with the patient's condition: systolic blood pressure, yearly tobacco use (in kg), low density lipoprotein (Idl), adiposity, family history (0 or 1), type A personality score (typea), obesity (body mass index), alcohol use, age, and the diagnosis of CHD (0 or 1).

```
CHD.head()
```

```
   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea  \
1          160    12.00                     5.73      23.11  Present     49   
2          144     0.01                     4.41      28.61   Absent     55   
3          118     0.08                     3.48      32.28  Present     52   
4          170     7.50                     6.41      38.03  Present     51   
5          134    13.60                     3.50      27.78  Present     60   

   Obesity  Alcohol  Age  Chd  
1    25.30    97.20   52    1  
2    28.87     2.06   63    1  
3    29.14     3.81   46    0  
4    31.99    24.26   58    1  
5    25.99    57.34   49    1  
```

```
CHD.tail()
```

```
     Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea  \
458          214      0.4                     5.98      31.72   Absent     64   
459          182      4.2                     4.41      32.10   Absent     52   
460          108      3.0                     1.59      15.23   Absent     40   
461          118      5.4                    11.61      30.79   Absent     64   
462          132      0.0                     4.82      33.41  Present     62   

     Obesity  Alcohol  Age  Chd  
458    28.45     0.00   58    0  
459    28.61    18.72   52    1  
460    20.09    26.64   55    0  
461    27.35    23.97   40    0  
462    14.70     0.00   46    1  
```

```
CHD.drop('Famhist', inplace=True, axis=1)
```

Selecting Feature Here, we need to divide the given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

```
#split dataset in features and target variable
feature_cols = ['Systolic BP', 'Tobacco', 'low-density lipoprotein', 
'Adiposity', 'typea', 'Obesity', 'Alcohol', 'Age', ]
X = CHD[feature_cols] # Features
y = CHD.Chd # Target variable
```

Splitting Data To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size. Additionally, We can use random_state to select records randomly.

```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.65,random_stat
e=0)
```

Here, the Dataset is broken into two parts in a ratio of 75:25. It means 75% data will be used for model training and 25% for model testing.

Model Development and Prediction First, import the Logistic Regression module and create a Logistic Regression classifier object using LogisticRegression() function.

Then, fit our model on the train set using fit() and perform prediction on the test set using predict().

```
#import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. We can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class-wise.

```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[174,  28],
       [ 56,  43]])
```

Here, we can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification. We have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions. In the output, 174 and 28 are actual predictions, and 56 and 43 are incorrect predictions.

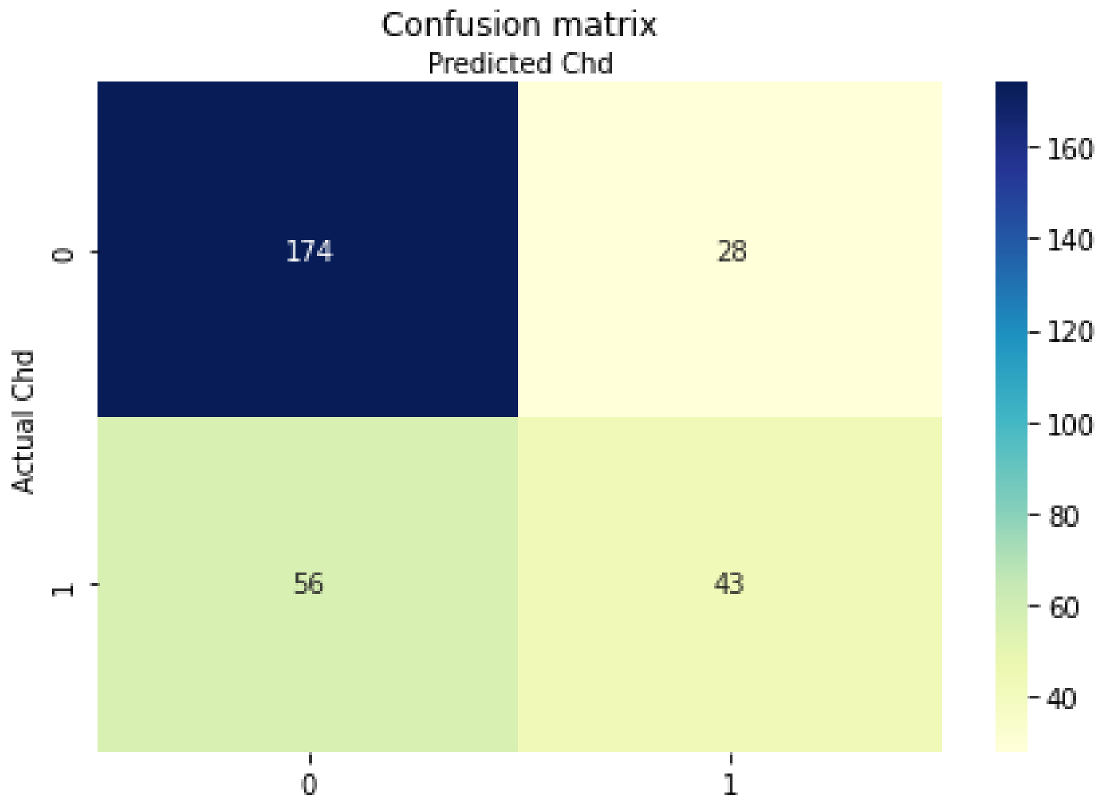
Visualizing Confusion Matrix using Heatmap Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

Here, we will visualize the confusion matrix using Heatmap.

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual Chd')
plt.xlabel('Predicted Chd')
```

```
Text(0.5, 257.44, 'Predicted Chd')
```



Confusion Matrix Evaluation Metrics Let's evaluate the model using model evaluation metrics such as accuracy, precision, and recall.

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7209302325581395
Precision: 0.6056338028169014
Recall: 0.43434343434343436
```

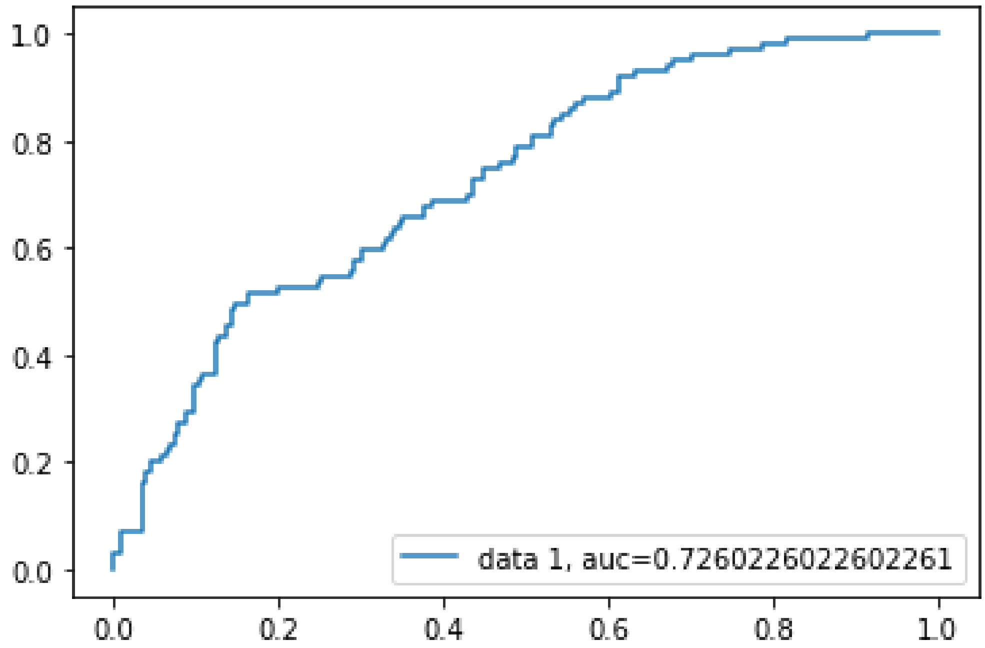
Well, we got a classification rate of 72%, considered as good accuracy.

Precision: Precision is about being precise, i.e., how accurate our model is. In other words, we can say, when a model makes a prediction, how often it is correct. In our prediction case, when our Logistic Regression model predicted patients are going to suffer from diabetes, that patients have 60% of the time.

Recall: If there are patients who have diabetes in the test set and our Logistic Regression model can identify it 43% of the time.

ROC Curve Receiver Operating Characteristic(ROC) curve is a plot of the true positive rate against the false positive rate. It shows the tradeoff between sensitivity and specificity.

```
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



AUC score for the case is 0.73. AUC score 1 represents perfect classifier, and 0.5 represents a worthless classifier.

Conclusion In this Notebook we were able to measure and evaluate the Accuracy, Recall, Precision of the data thoroughly.

Experiment – 3

Aim - To observe the performance of dataset using Decision Tree Algorithm. Attribute Selection Measures Information Gain, Gain Ratio Gini index Optimizing Decision Tree Performance Classifier Building in Scikit-learn Pros and Cons Conclusion.

Here, we are going to predict coronary heart disease using Decision Tree Classifier.

Let's first load the required coronary heart disease dataset using the pandas' read CSV function.

We will download data from the following link:
https://www.kaggle.com/datasets/billbasener/coronary-heart-disease?resource=download

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

from google.colab import drive

drive.mount('/content/gdrive', force_remount=True)

Mounted at /content/gdrive

cd /content/gdrive/MyDrive/Machine_Learning/Colab Notebooks/ML_Practicals/1_Practical/P3_Decision Tree

/content/gdrive/MyDrive/Machine_Learning/Colab Notebooks/ML_Practicals/1_Practical/P3_Decision Tree

ls

CHDdata.csv  CHD_Data.csv  CHDdata.gsheet  Decision_Tree.ipynb  diabetes.png

import pandas as pd
col_names = ['Systolic BP', 'Tobacco', 'low-density lipoprotein', 'Adiposity', 'Famhist', 'typea', 'Obesity', 'Alcohol', 'Age', 'Chd']
# load dataset
CHD = pd.read_csv("CHD_Data.csv", header=None, names=col_names)

CHD.head()

   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea  \
1          160    12.00                     5.73      23.11  Present     49
```

```
2           144     0.01                     4.41       28.61   Absent     55
3           118     0.08                     3.48       32.28  Present     52
4           170     7.50                     6.41       38.03  Present     51
5           134    13.60                     3.50       27.78  Present     60

   Obesity  Alcohol  Age  Chd
1    25.30    97.20   52    1
2    28.87     2.06   63    1
3    29.14     3.81   46    0
4    31.99    24.26   58    1
5    25.99    57.34   49    1
```

Feature Selection Here, we need to divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

```
CHD.drop('Famhist', inplace=True, axis=1)
```

```
CHD.head()
```

```
   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  typea  Obesity
\
1          160    12.00                     5.73      23.11     49    25.30
2          144     0.01                     4.41      28.61     55    28.87
3          118     0.08                     3.48      32.28     52    29.14
4          170     7.50                     6.41      38.03     51    31.99
5          134    13.60                     3.50      27.78     60    25.99

   Alcohol  Age  Chd
1    97.20   52    1
2     2.06   63    1
3     3.81   46    0
4    24.26   58    1
5    57.34   49    1
```

```
#split dataset in features and target variable
feature_cols = ['Systolic BP', 'Tobacco', 'low-density lipoprotein',
'Adiposity', 'typea', 'Obesity', 'Alcohol', 'Age', ]
X = CHD[feature_cols] # Features
y = CHD.Chd # Target variable
```

Splitting Data To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split the dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size.

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=1) # 70% training and 30% test
```

Building Decision Tree Model Let's create a Decision Tree Model using Scikit-learn.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Evaluating Model Let's estimate, how accurately the classifier or model can predict the type of cultivars.

Accuracy can be computed by comparing actual test set values and predicted values.

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

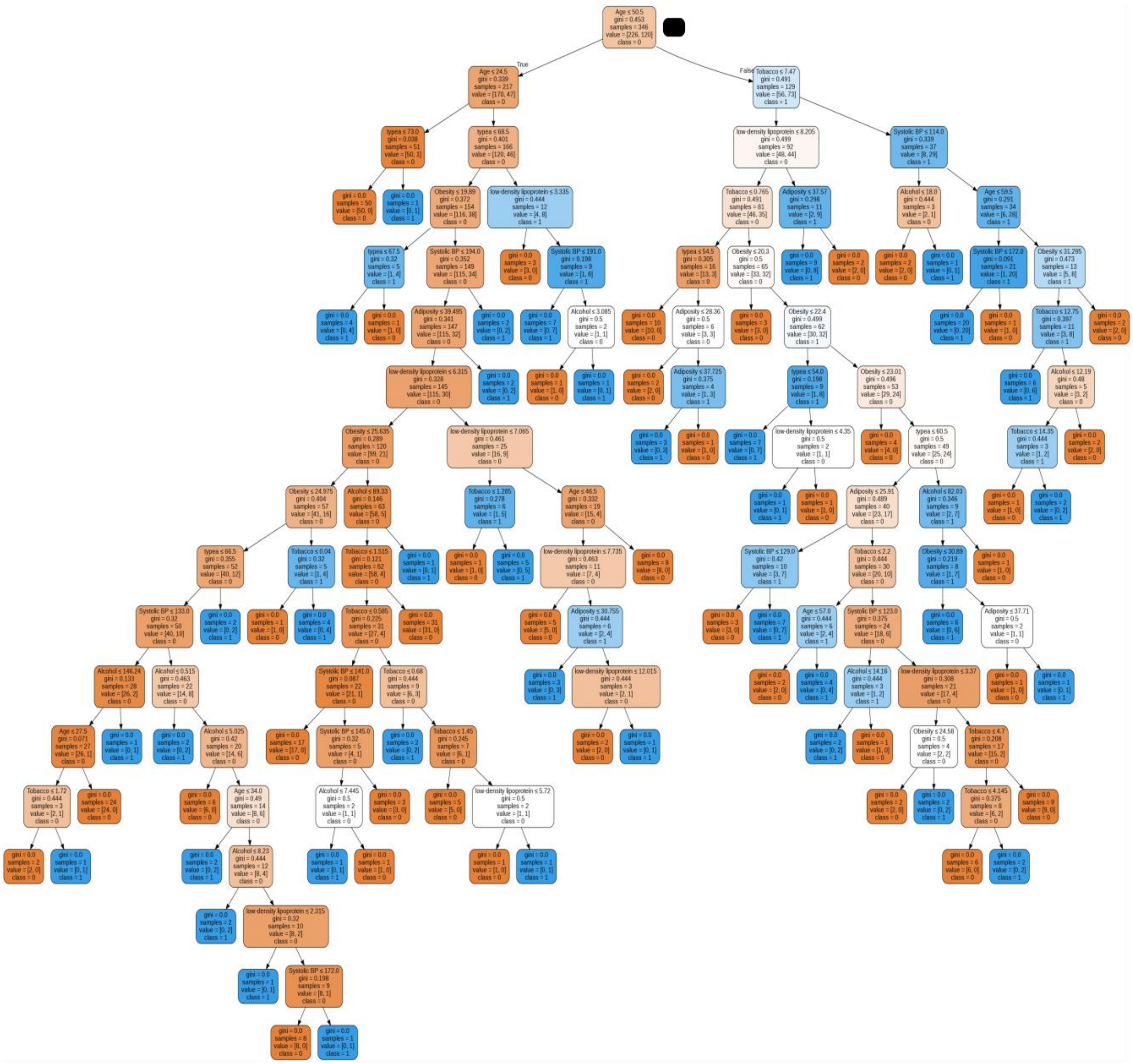
```
Accuracy: 0.6896551724137931
```

Well, We got a classification rate of 68%, considered as good accuracy. We can improve this accuracy by tuning the parameters in the Decision Tree Algorithm.

Indented block

Visualizing Decision Trees You can use Scikit-learn's export_graphviz function for display the tree within a Jupyter notebook. For plotting tree, We also need to install graphviz and pydotplus.

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred as Gini ratio, which measures the impurity of the node. You can say a node is pure when all of its records belong to the same class, such nodes known as the leaf node.

Here, the resultant tree is unpruned. This unpruned tree is unexplainable and not easy to understand. In the next section, let's optimize it by pruning.

Optimizing Decision Tree Performance criterion: optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

splitter :string, optional (default=”best”) or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depth : int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

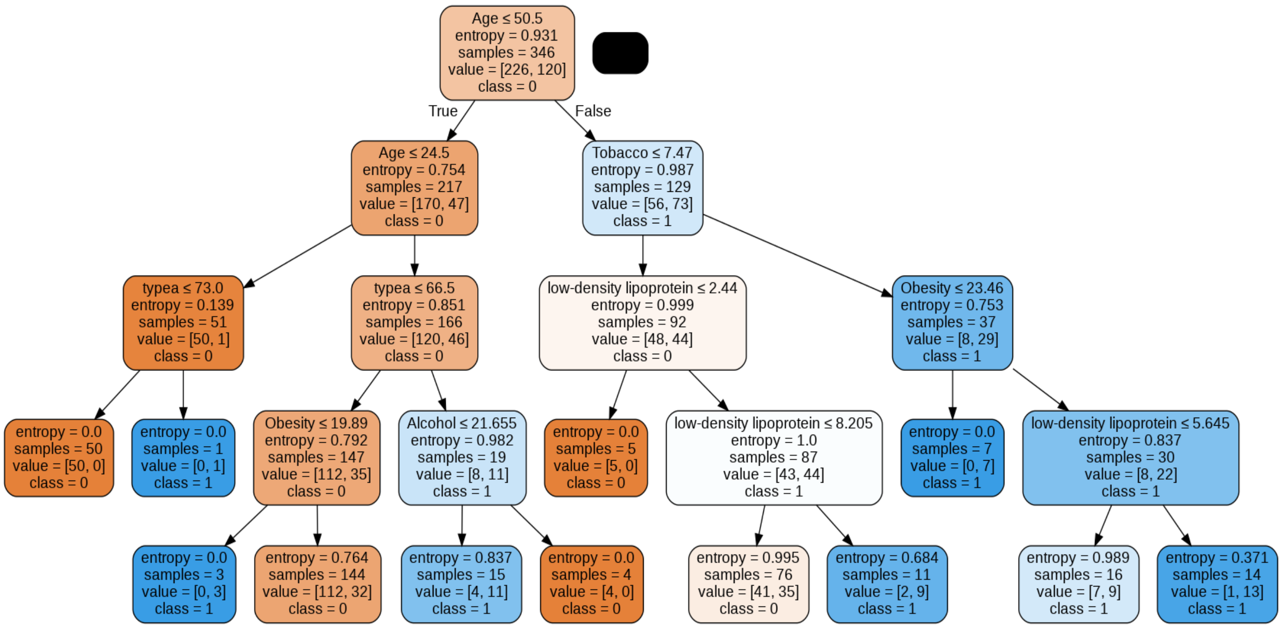
In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, you can plot a decision tree on the same data with max_depth=3. Other than pre-pruning parameters, You can also try other attribute selection measure such as entropy.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=4)


# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)


#Predict the response for test dataset
y_pred = clf.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7068965517241379
```

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names =
```

```
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



Conclusion Finally we were able to ovserve the performance of dataset using Decision Tree Algorithm. Attribute Selection Measures Information Gain Gain Ratio Gini index Optimizing Decision Tree Performance Classifier Building in Scikit-learn Pros and Cons Conclusion.