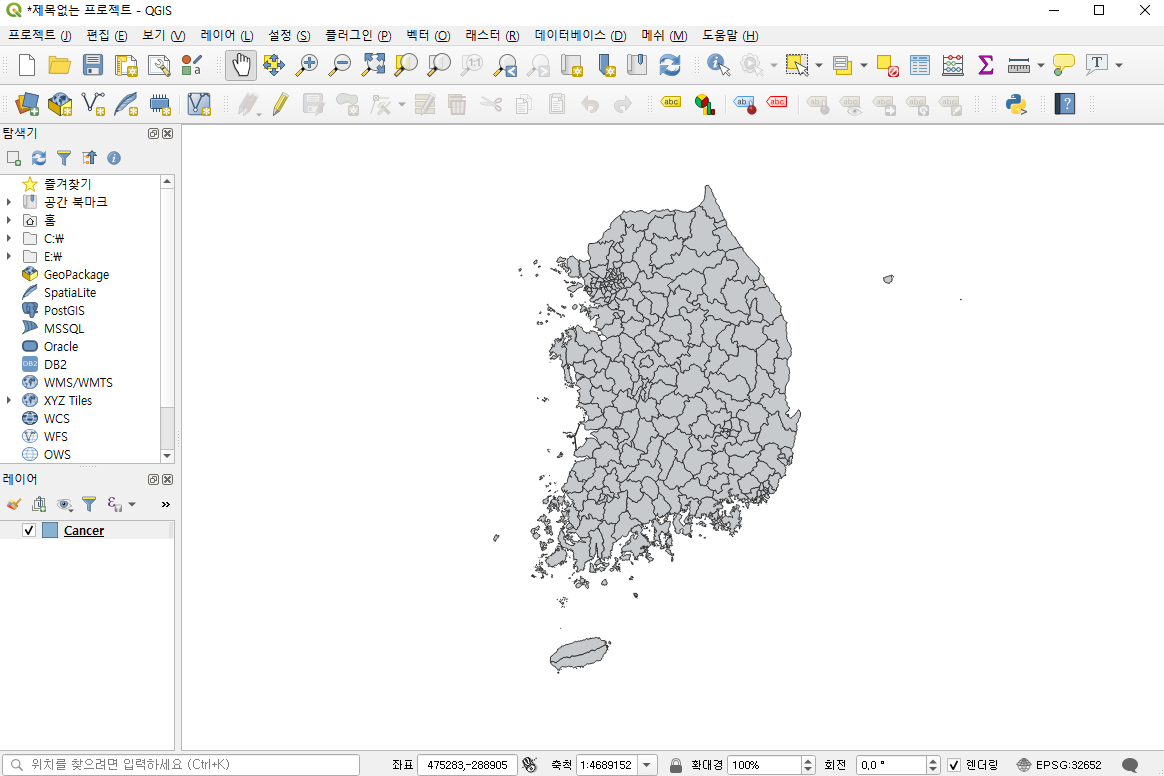
**Topic: Influential Variables of Regional Disease Prevalence Rate**

**General Notice**

The Ministry of Health and Welfare in Korea wants to check any spatial relationship exists between administrative district and disease prevalence and with respect to the disease of spatial dependency, MOHW wants to find out any regional factor, which contribute to higher or lower disease prevalence for the purpose of improving the public health in Korea. To solve this two problem, we will perform spatial autocorrelation analysis by “Geoda” program, and then perform decision tree analysis to find influential spatial factor by “R” which is statistical tool program and “Python” which is the most widely used easy programming language.

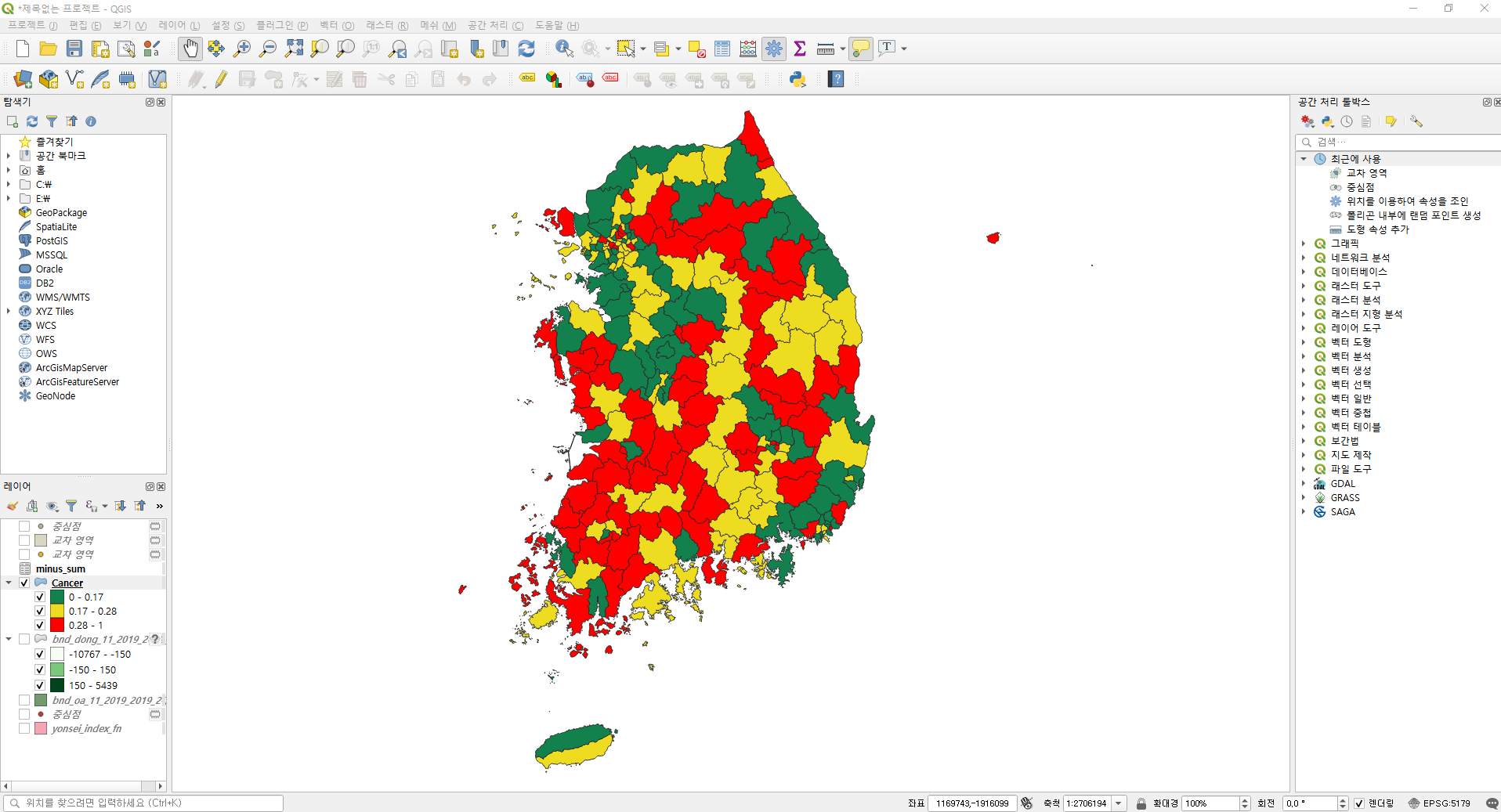
**Exercise 1: Present the disease prevalence data into a map**

First, use the [Add Layer] to place the shapefile containing the disease prevalence rate and the region-based shapefile on QGIS. [Layer] -> [Add Layer] -> [Add Vector Layer] (File encoding selects ECU-KR or cp949 to prevent broken Hangul)



**Figure 1**

And describe the disease prevalence rate in tertile map for low, middle, high. Click [Property] -> [Symbol] -> [Categorized(단계구분)]. Disease name column is disease prevalence rate and the rate will be divided into three stages: low, middle, and high, so the interval will be set at 3. You can also choose the color lamp as you want. Finally click [Apply], you can see Figure 2.



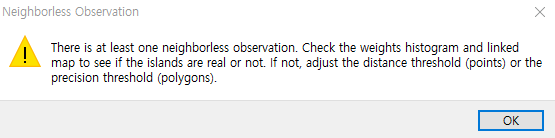
**Figure 2**

**Exercise 2: Data Analysis**

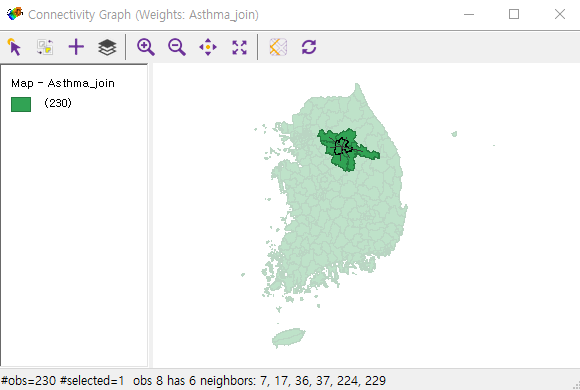
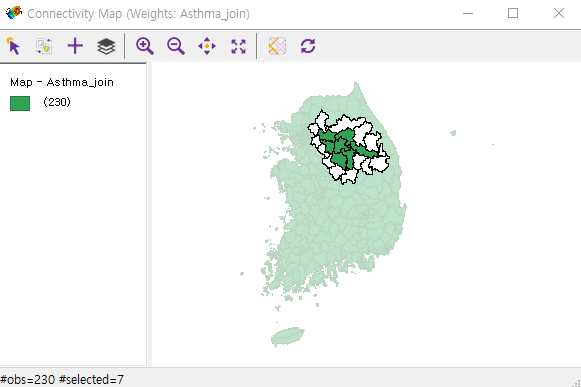
Using ‘Geoda’ to calculate spatial autocorrelation analysis, Moran’s I and use ‘R’ to calculate which variables are related to disease prevalence rate by region by conducting ‘Decision Tree’. Moran’s I is a correlation coefficient that measures the overall spatial autocorrelation of data set, and it measures how one object is similar to others surrounding it. And Decision Tree analyzes the data and shows the patterns that exist between them as a combination of predictable rules, and is called the decision tree because its shape is like a 'tree'.

**Step 1. Spatial Autocorrelation Analysis (Moran’s I)**

(You have to install Geoda program in google) First, open “Geoda” and upload the disease shapefile. Before retrieving Moran’s I, we have to make a weight about region. So click [W] icon -> [create]. Choose SOURCE\_ID for Variable and choose Contiguity Weight, the most commonly used Queen Contiguity. As you can see in Figure 3, the information windows come out because there is a place far away like an island. And Figure 4 is result.

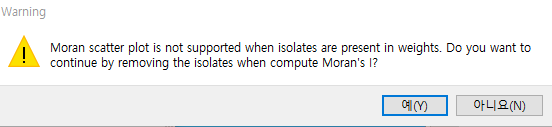


**Figure 3**

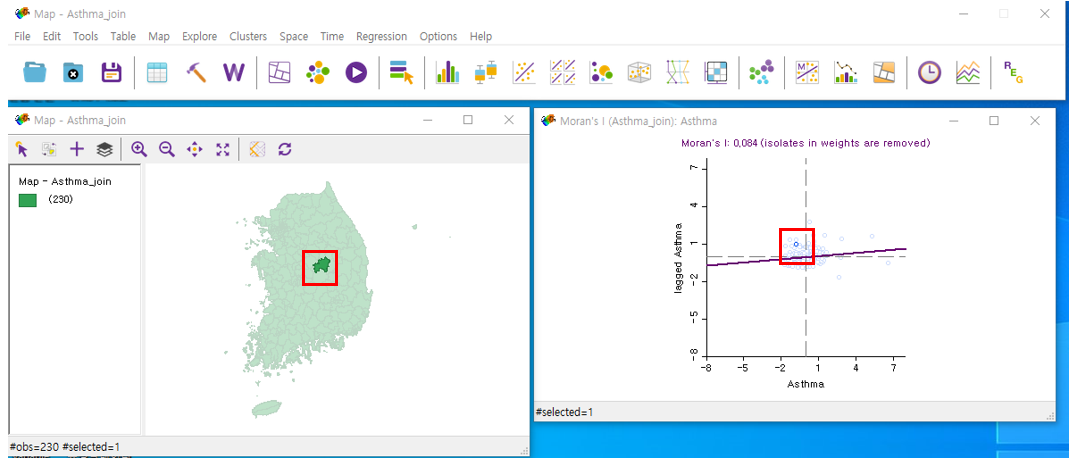


**Figure 4**

After retrieving weight of region, we can calculate Moran’s I. Click [space] -> [Univariate Moran’s I]. Because what we want to know is the value of Moran's I for disease, so choose disease as a variable. And as shown in Figure 3, because there is a remote area like an island, information window like Figure 5 appears. Finally, Figure 6 is result.

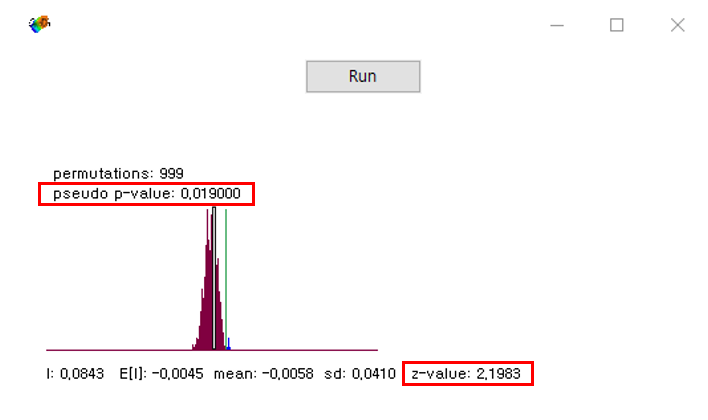


**Figure 5**



**Figure 6**

In addition to Moran's I, z-value and p-value should be obtained. Click Moran’s I scatter plot with right mouse button and click [randomization] -> [permutation]. Then as shown in Figure 7, you can see pseudo p-value and z-value.



**Figure 7**

And table 1 represents moran's i, z-value, and p-value for all diseases.

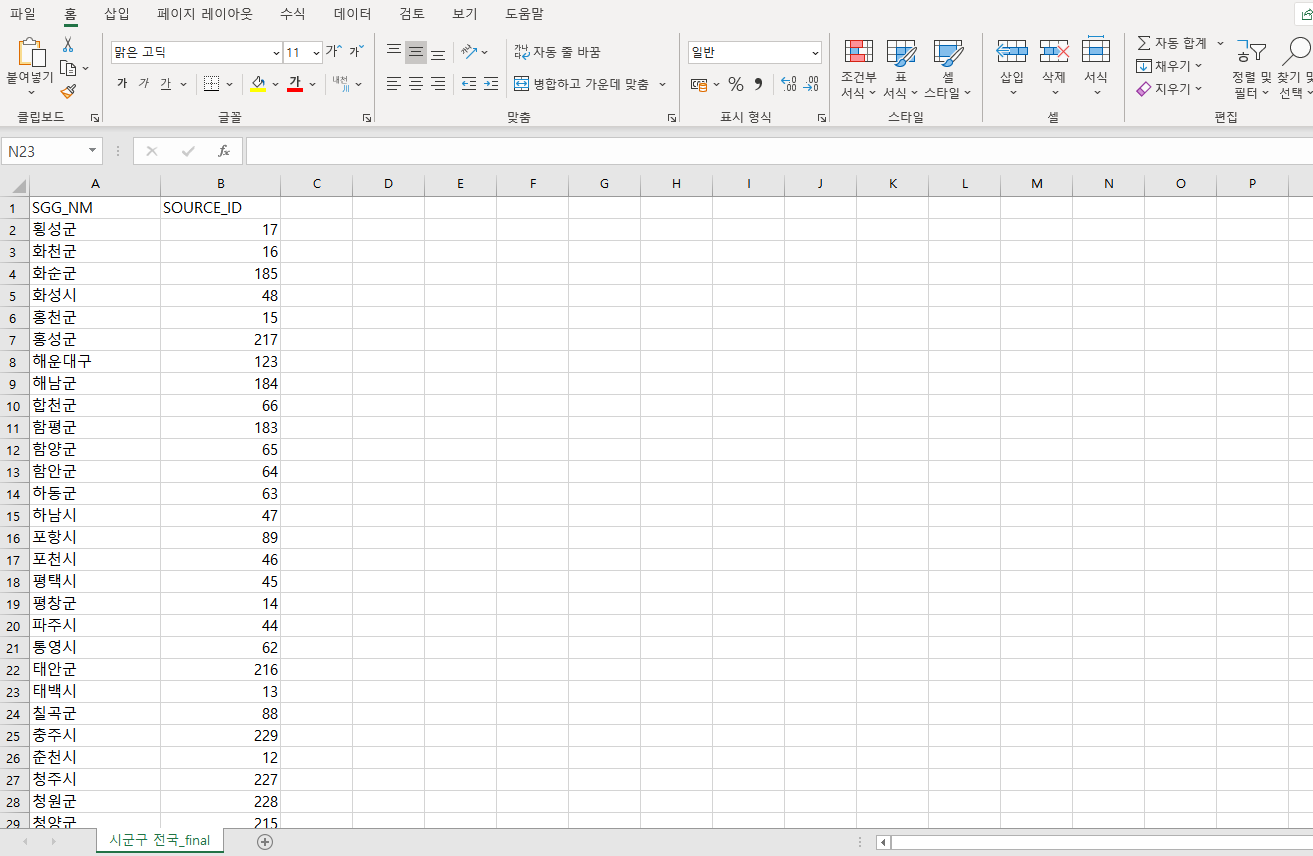
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Statistically significant Moran’s I | | | | 1. Statistically non-significant Moran’s I | | | |
| Disease | Moran’s I | z-score | p-value | Disease | Moran’s I | z-score | p-value |
| Angina\_pec | 0.189 | 4.6024 | 0.001 | **Anemia** | 0.027 | 0.6829 | 0.244 |
| Arthritisi | 0.506 | 12.2793 | 0.001 | **Asthma** | 0.084 | 2.1983 | 0.019 |
| Atopic\_der | 0.370 | 8.3490 | 0.001 | **Cancer** | 0.178 | 4.3513 | 0.001 |
| Cataract | 0.449 | 10.8673 | 0.001 | **Depression** | 0.169 | 3.7514 | 0.002 |
| Diabete\_me | 0.342 | 8.2740 | 0.001 | **Gastroduod** | 0.150 | 3.5144 | 0.002 |
| Dyslipidem | 0.184 | 4.2639 | 0.001 | **Hemorrhoid** | 0.113 | 2.875 | 0.004 |
| Hypertensi | 0.364 | 9.1752 | 0.001 | **Hepatiti\_1** | 0.056 | 1.3384 | 0.093 |
| Myocardial | 0.235 | 5.5911 | 0.001 | **Hepatitis** | -0.083 | -1.8367 | 0.025 |
| Osteoporos | 0.490 | 1.6865 | 0.001 | **Hypersensi** | 0.029 | 0.7640 | 0.221 |
| Otitis\_med | 0.227 | 5.2311 | 0.001 | **Pulmonary** | 0.138 | 3.3218 | 0.003 |
| Stroke | 0.412 | 9.7512 | 0.001 | **Thyroid\_di** | 0.176 | 4.0637 | 0.001 |
|  |  |  |  | **Urinary\_in** | -0.010 | -0.1394 | 0.449 |

(\*Allergic\_r data 제외)

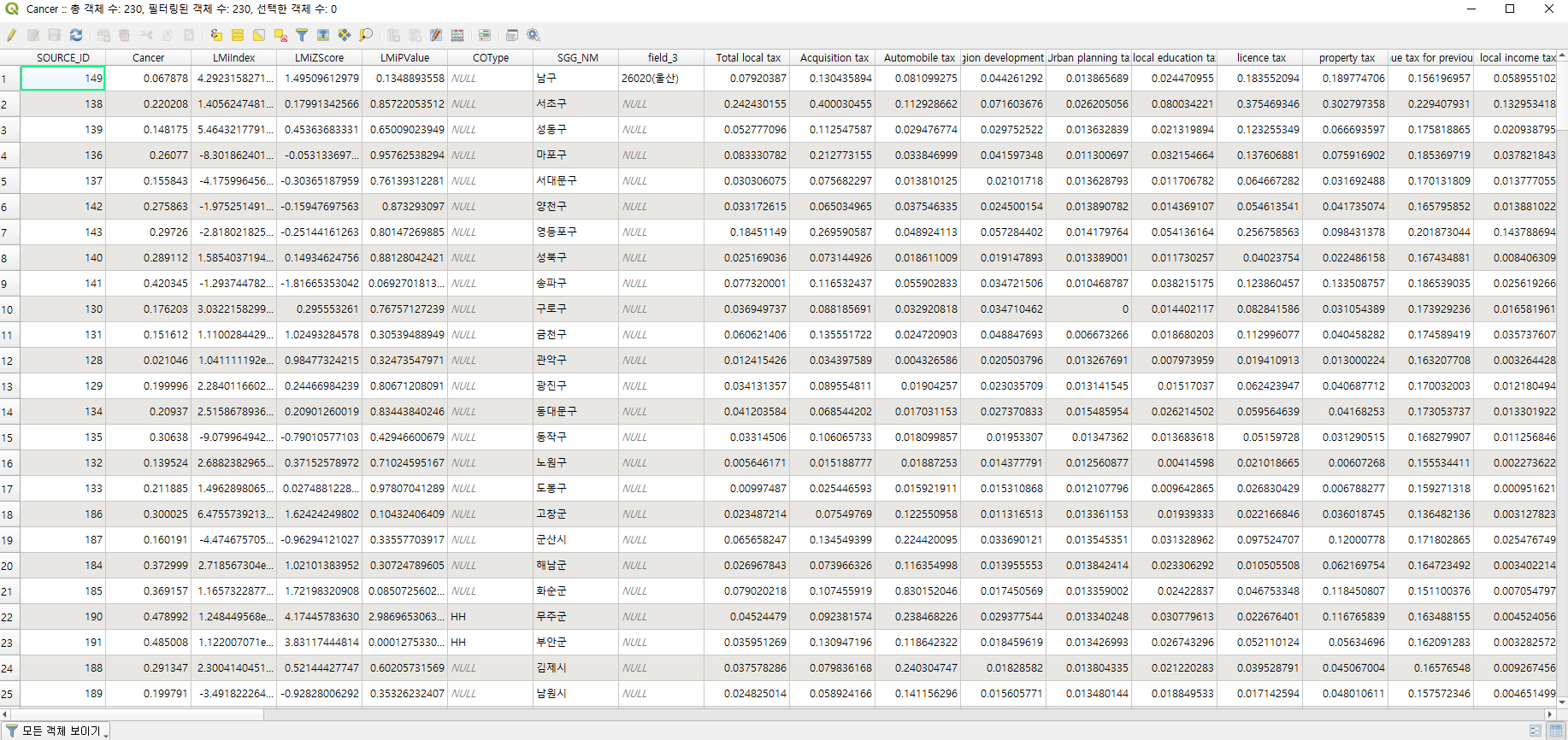
**Table 1**

**Step 2. Data preprocessing**

To see the relationship between disease prevalence rate and variables, 1) Click [Add Layer] to add a CSV file containing the values of the local variables, 2) Click [Add layer] to add a CSV file containing district name. Then join the disease shapefile and the district name CSV and join with the variable CSV. If you open the attribute table of final files, you can see that you can join files based on the "district name”. (Or you can join the files using the Join function of R.)

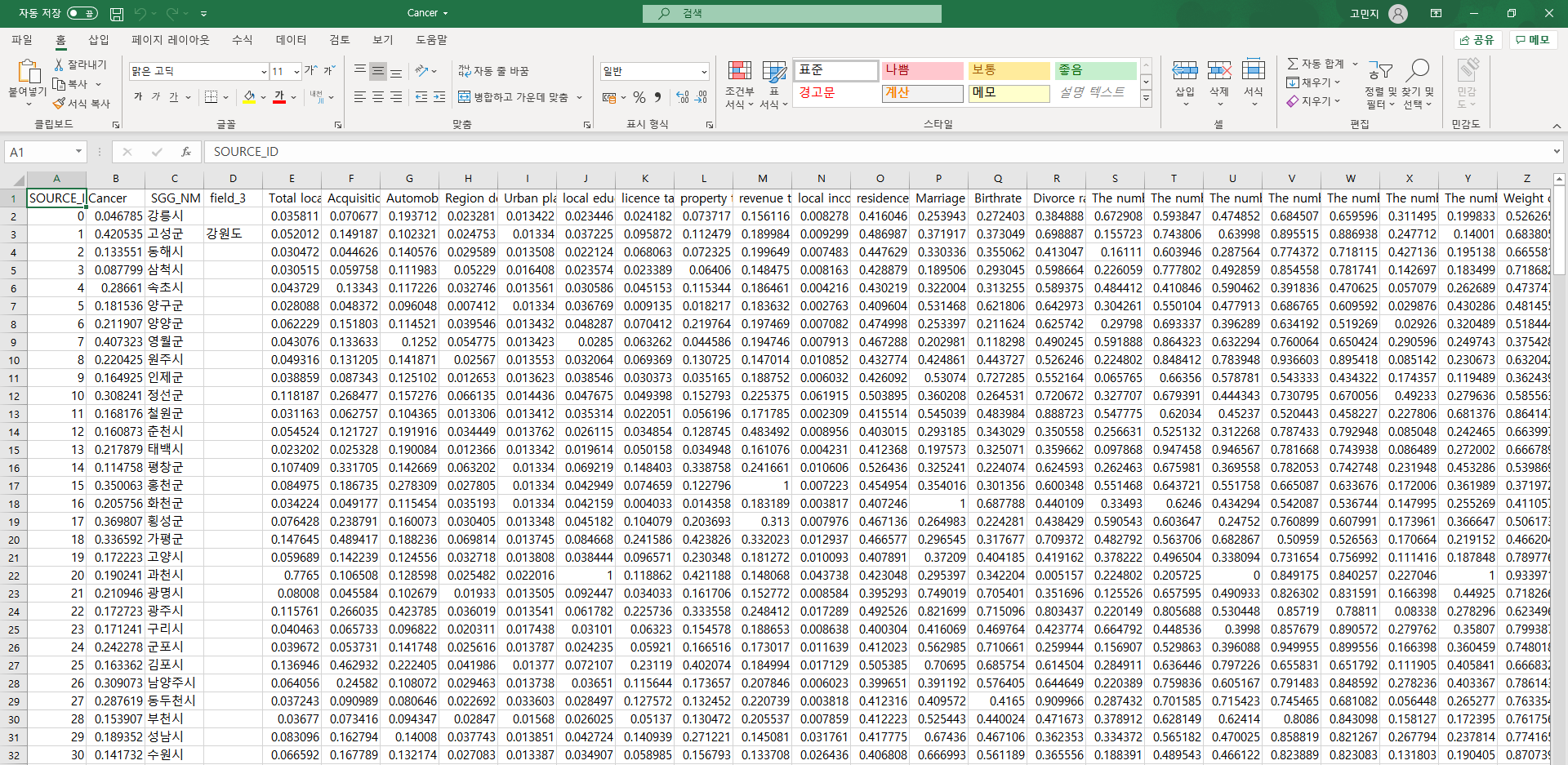


**Figure 3**



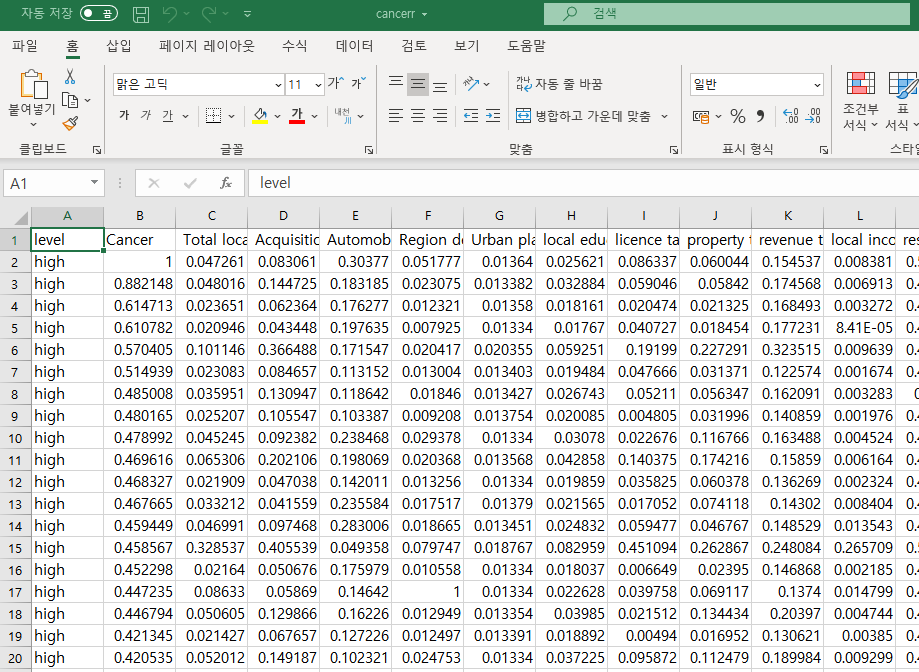
**Figure 4**

Finally, extract the file into a CSV file using the [Export] function. (Click final layer with right mouse button and click export) And because what is needed for the Decision Tree Analysis is the disease prevalence rate and the value of the local variables, it is okay to extract only the fields that you need. When exporting, select the field you want in [Select fields to export and export options]. Figure 8 is exported CSV file.



**Figure 5**

In Step 1, the values of each range were known by dividing the prevalence rate by low, mid, and high (as seen in the layer window), and reference this value to create a level column for low, mid, and high in Excel like figure 9. You can delete columns (SOURCE\_ID, directory name etc..) that are not required for the decision tree, or you can process them later in R.



**Figure 6**

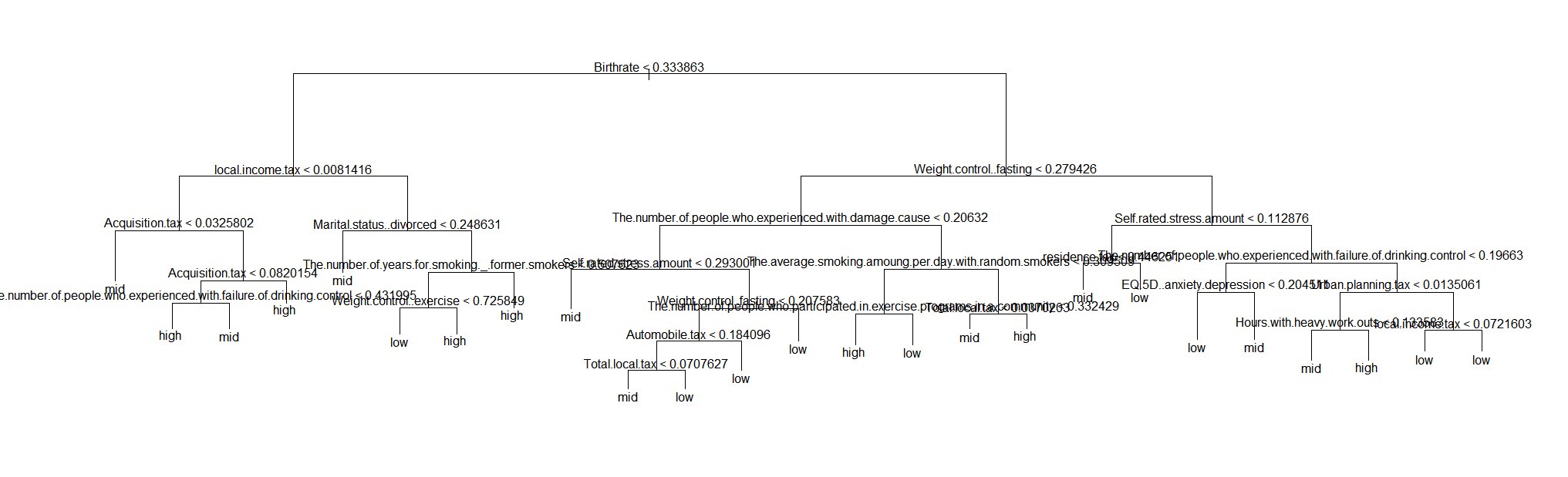
And then save the CSV file.

**Step 3. Decision tree**

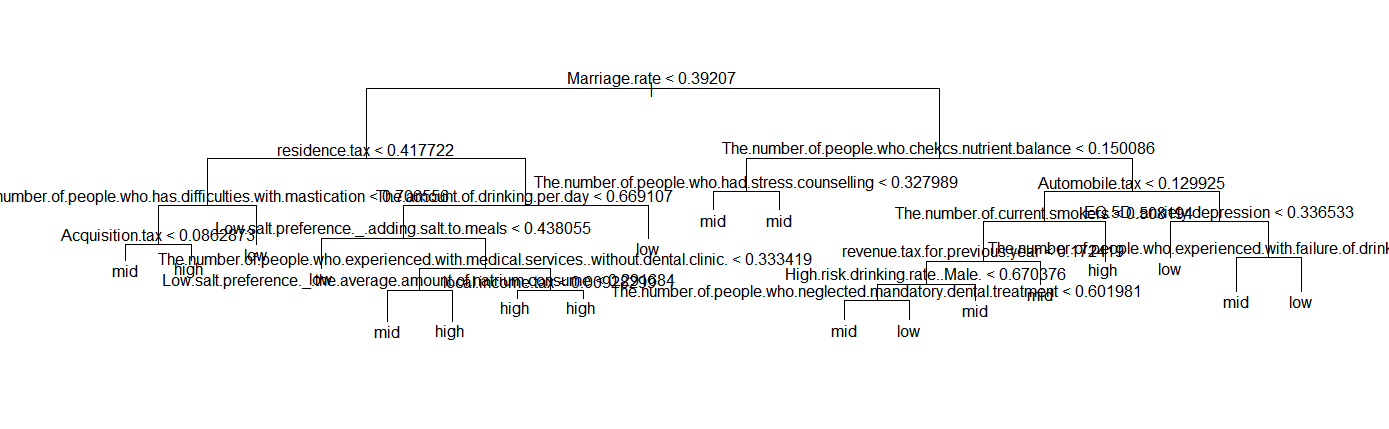
Finally, use R to perform a Decision Tree.

|  |
| --- |
| data = read.csv("E://유병률데이터//cancer.csv")  #in this material, use cancer data for example  data$level = as.factor(data$level)  str(data)  **#tree()**  library(tree)  data = data[-c(2)] #delete needless column  treeversion<-tree(level~.,data=data)  plot(treeversion)  text(treeversion)  **#caret()**  library(caret)  set.seed(1000)  intrain = createDataPartition(y = data$level, p=0.7, list=FALSE)  data\_train = data[intrain,]  data\_test = data[-intrain,]  treeversion2 = tree(level~.,data = data\_train)  plot(treeversion2)  text(treeversion2)  **#k-fold cross-validation**  treeversion2\_cv = cv.tree(treeversion2, FUN = prune.misclass)  plot(treemod2\_cv)  treeversion2\_prune = prune.misclass(treeversion2, best=8)  plot(treeversion2\_prune)  text(treeversion2\_prune, pretty=0)  **#test evaluate**  library(e1071)  treepred = predict(treeversion2\_prune, data\_test, type='class')  confusionMatrix(treepred, data\_test$level)  **#rpart()**  library(rpart)  rpart\_train <- rpart(level~.,data=data\_train, method="class")  plot(rpart\_train)  text(rpart\_train,pretty=0)  printcp(rpart\_train)  plotcp(rpart\_train)  rpart2\_prune <- prune(rpart\_train, cp= rpart\_train$cptable[which.min(rpart\_train$cptable[,"xerror"]),"CP"])  plot(rpart2\_prune)  text(rpart2\_prune,pretty=0)  **#ctree()**  library(party)  partyversion = ctree(level~., data = data\_train)  plot(partyversion)  A = ctree\_control(maxdepth=3)  partyversion2\_a = ctree(level~., data= data\_train, control=A)  plot(partyversion2\_a) |

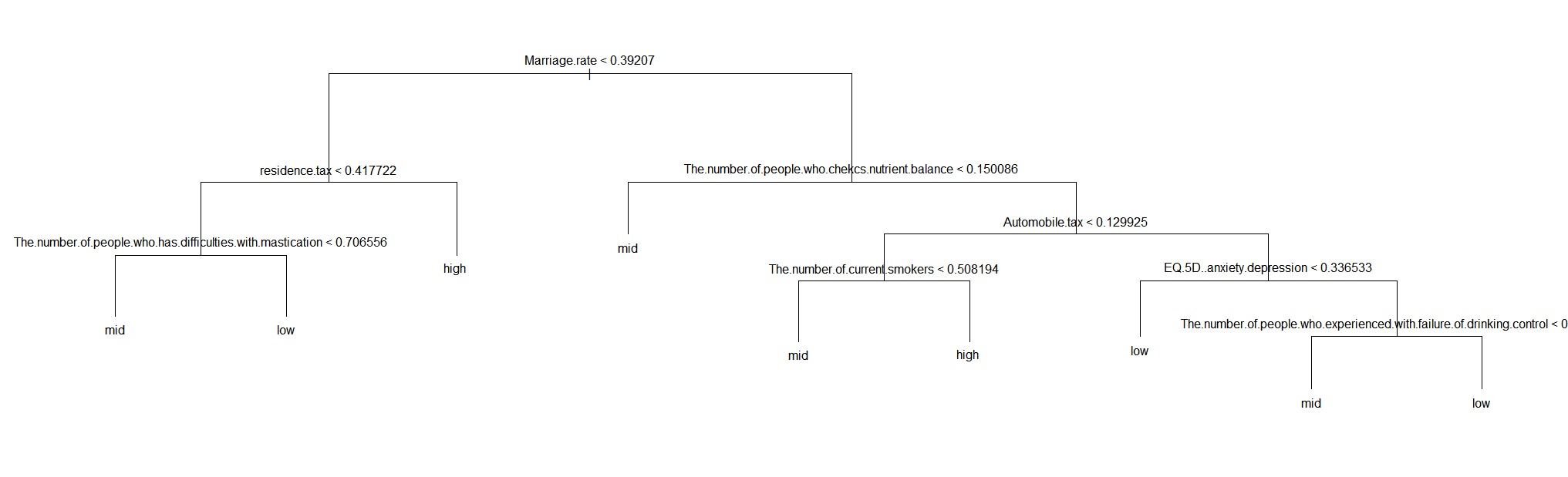
#tree()

****

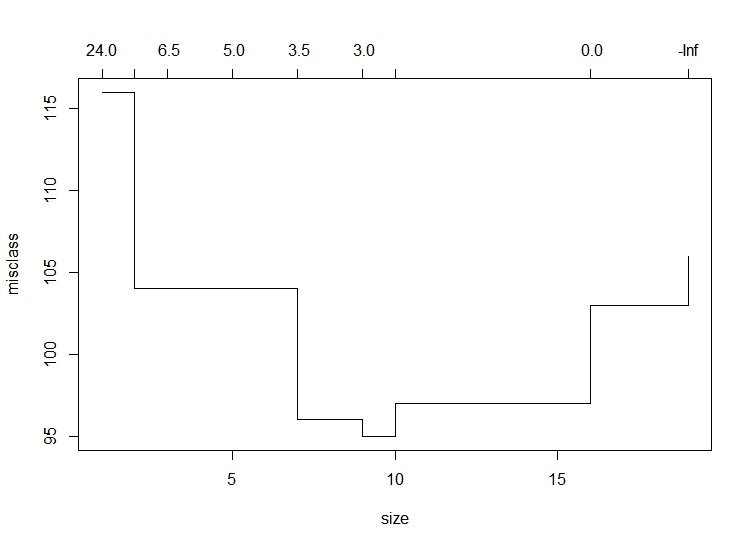
#caret()



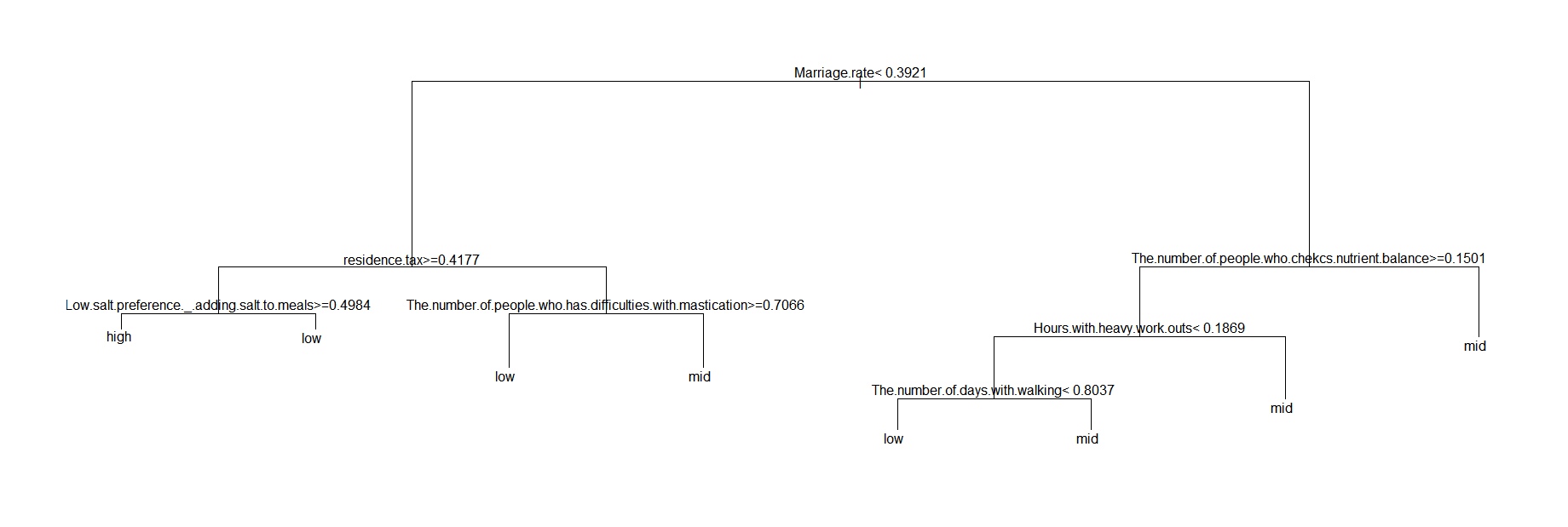
#caret() after prune



#k-fold cross validation ex

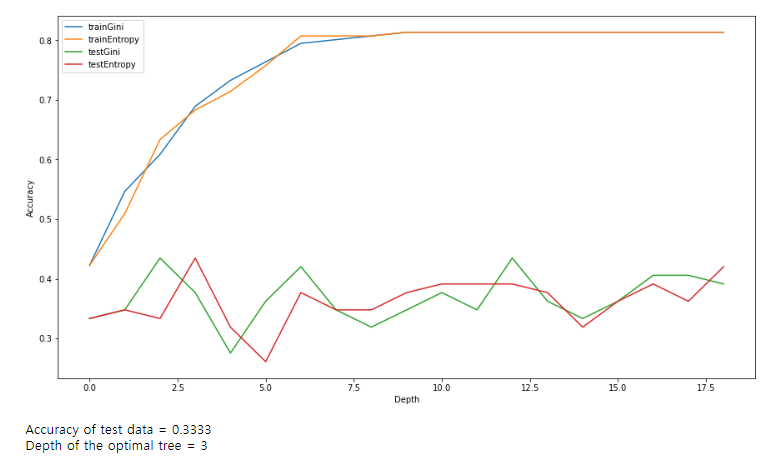


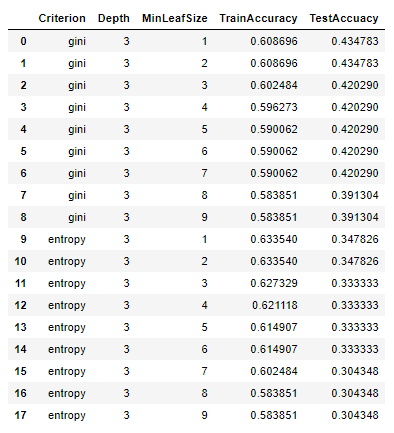
#rpart()

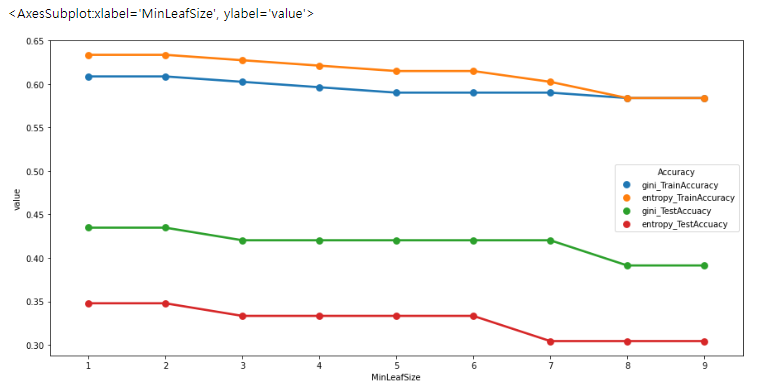


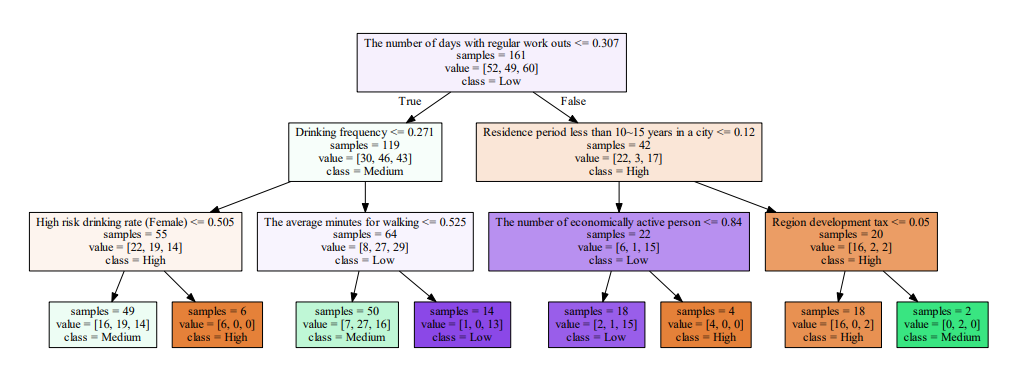
**Step 3. Python**

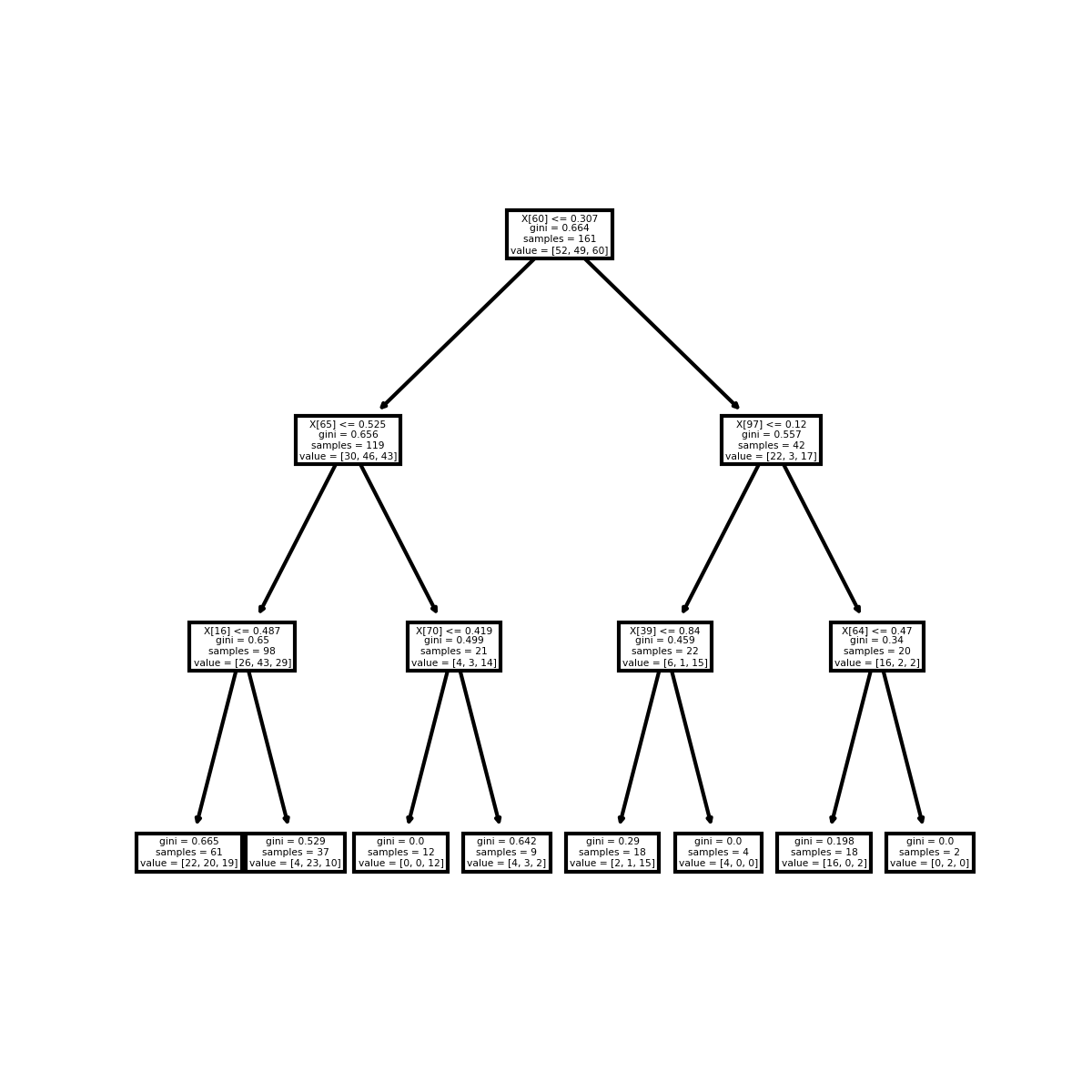
|  |
| --- |
| **#Load the library required for tree analysis.**  **#conda install python-graphviz : Installation using the command**  **import pandas as pd**  **from sklearn.tree import DecisionTreeClassifier**  **from sklearn.tree import export\_graphviz**  **from sklearn.model\_selection import train\_test\_split**  **import os**  **import graphviz**  **import matplotlib.pyplot as plt**  **from tqdm import tqdm\_notebook**  **import numpy as np**  **import seaborn as sns**  **#Call up Asthma data.**  **table1 = pd.read\_csv("C:\\Users\\scsi\_public\\Desktop\\intern\\Asthma\_variable.csv", encoding="cp949")**  **table1 = table1.fillna(0)**  **#Set Asthma above 0.24 (top 33.3%)high, 0.17 above (top 66.6%)medium, and 017 below (bottom 33%) low.**  **table1.loc[table1.Asthma >= 0.24, "Asthma"] = 100**  **table1.loc[(table1.Asthma >= 0.17) & (table1.Asthma <= 0.24), "Asthma"] = 50**  **table1.loc[table1.Asthma <= 0.17, "Asthma"] = 0**  **table1.loc[table1.Asthma == 100, "Asthma"] = "High"**  **table1.loc[table1.Asthma == 50, "Asthma"] = "Medium"**  **table1.loc[table1.Asthma == 0, "Asthma"] = "Low"**  **table1 = table1.astype('string')**  **#The variable (target variable) to use as the target value is Asthma.**  **table1["Asthma"].value\_counts()**  **#Let's first extract the variables to use for the target value.**  **#Let's select all the continuous variables and enter them as explanatory variables.**  **X = table1.iloc[ :, 1: ]**  **Y = table1.iloc[ : , 0]**  **#Using the train\_test\_split function, randomly distribute the training set and test set.**  **X\_train , X\_test ,Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.30,random\_state=1234)**  **model = DecisionTreeClassifier(random\_state=1234)**  **model.fit(X\_train, Y\_train)**  **print("Training Set Accuracy {:.3f}".format(model.score(X\_train,Y\_train)))**  **print("Test Set Accuracy: {: .3f}".format(model.score(X\_test, Y\_test)))**  **model.get\_params**  **# Use the Decision Tree algorithm to post - pruning**  **#We can know The accuracy of the test data and the depth of the optimal tree**  **trainGini = []**  **testGini = []**  **trainEntropy = []**  **testEntropy = []**  **depth = []**  **for k in tqdm\_notebook(range(1, 20)):**  **model = DecisionTreeClassifier(criterion='gini', max\_depth=k)**  **model.fit(X\_train, Y\_train)**  **trainGini.append(model.score(X\_train, Y\_train))**  **testGini.append(model.score(X\_test, Y\_test))**  **model = DecisionTreeClassifier(criterion='entropy', max\_depth=k)**  **model.fit(X\_train, Y\_train)**  **trainEntropy.append(model.score(X\_train, Y\_train))**  **testEntropy.append(model.score(X\_test, Y\_test))**  **depth.append(k)**  **print('depth = %d done.' % k)**  **plt.figure(figsize=(15, 8))**  **plt.plot(trainGini, label='trainGini')**  **plt.plot(trainEntropy, label='trainEntropy')**  **plt.plot(testGini, label='testGini')**  **plt.plot(testEntropy, label='testEntropy')**  **plt.legend()**  **plt.xlabel('Depth')**  **plt.ylabel('Accuracy')**  **plt.show()**  **# Find the best depth with the greatest accuracy**  **nDepth = depth[np.argmax(testGini)]**  **#Use the tree with opt\_alpha applied.**  **model = DecisionTreeClassifier(max\_depth = nDepth)**  **model.fit(X\_train, Y\_train)**  **print(' Accuracy of test data = %.4f' % model.score(X\_test, Y\_test))**  **print(' Depth of the optimal tree = %d' % nDepth)**  **#Set max\_depth to the maximum depth obtained above.**  **#This time, we will compare the accuracy of various models and analyze them by applying model.**  **train\_result = []**  **test\_result = []**  **model\_criterion = []**  **model\_max\_depth = []**  **parameter\_min\_leaf = []**  **insert\_criterion = ['gini', 'entropy']**  **max\_depth = 3**  **list\_min\_leaf= [i for i in range(1,10)]**  **for i in insert\_criterion:**  **for n in list\_min\_leaf:**  **tree = DecisionTreeClassifier(criterion=i, max\_depth=max\_depth,min\_samples\_leaf=n, random\_state=1000)**  **tree.fit(X\_train, Y\_train)**  **train\_result.append(tree.score(X\_train, Y\_train))**  **test\_result.append(tree.score(X\_test, Y\_test))**  **model\_criterion.append(i)**  **model\_max\_depth.append(max\_depth)**  **parameter\_min\_leaf.append(n)**    **result = pd.DataFrame()**  **result["Criterion"] = model\_criterion**  **result["Depth"] = max\_depth**  **result["MinLeafSize"] = parameter\_min\_leaf**  **result["TrainAccuracy"] = train\_result**  **result["TestAccuacy"] = test\_result**  **result**  **#The most accurate training model and test model can be found easily using the ogmelt function and visualization.**  **plt.figure(figsize=(15,7))**  **result\_melt = pd.melt(result, id\_vars = ['Criterion', 'Depth', 'MinLeafSize'])**  **result\_melt['Accuracy'] = result\_melt['Criterion']+'\_'+result\_melt['variable']**  **sns.pointplot(data=result\_melt, x='MinLeafSize', y='value', hue='Accuracy')**  **#The tree model with a min leaf size of 2 of the entropy model appears to be the best fit.**  **model1 = DecisionTreeClassifier(criterion = 'entropy', max\_depth=3, min\_samples\_leaf=2, random\_state = 1000)**  **model1.fit(X\_train, Y\_train)**  **print("Training Set Accuracy {:.3f}".format(model.score(X\_train,Y\_train)))**  **print("Test Set Accuracy: {: .3f}".format(model.score(X\_test, Y\_test)))**  **# we used the graphviz library to graph the actual tree structure.**  **export\_graphviz(model1, out\_file='tree1.dot',**  **feature\_names=X.columns, impurity = False, filled = True, class\_names=['High', 'Medium', 'Low'])**  **os.environ["PATH"]+=os.pathsep+'C:/Program Files (x86)/Graphviz 2.38/bin/'**  **with open("tree1.dot", encoding = 'utf-8') as f:**  **dot\_graph = f.read()**  **display(graphviz.Source(dot\_graph))**  **graphviz.Source(dot\_graph) .render(filename='tree.png')**  **#Also visualize Decision Trees using Matplotlib**  **from sklearn import tree**  **tree.plot\_tree(model)** |









****