

**ANL 252**

**Supply Chain Information Management and Technology**

# **End-of-Course Assessment**

**July 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Ng Yi Jian** | **E2211293** |

**Submission Date: 03/11/2023**

(1)

Treating missing values

The first thing we would do in the medical cost dataset would be treating the missing values in the dataset.

We will first convert the medical cost excel into a dataset in Python.

# converting excel file into dataset

import pandas as pd

medical\_cost = pd.read\_csv('ECA.csv')

We will now check if there are any missing values in the dataset.

#locating missing values

medical\_cost.isnull().sum(axis=0)

The output would show that the missing values under the column as follows,

PersonID 0

age 123

sex 0

bmi 0

children 0

smoker 0

region 0

charges 0

dtype: int64

Before we treat the missing value, we would first need to standardize the “sex” column to only values that is female and male by replace F and M respectively.

#replacing data in sex

medical\_cost['sex'].replace('M','male',inplace = True)

medical\_cost['sex'].replace('F','female', inplace = True)

medical\_cost

Since we have located the missing data, we will now do a mean imputation on the missing values. We need to calculate the mean age of male and female.

#finding mean age of male

male = (medical\_cost['sex'] == 'male')

null\_male = (medical\_cost['age'].isnull())

condition1 = (male & ~ null\_male)

medical\_cost[condition1]

medical\_cost[condition1]['age'].mean()

**output:**

39.011382113821135

#finding mean age of female

female = (medical\_cost["sex"] == "female")

null\_female = (medical\_cost['age'].isnull())

condition = (female & ~ null\_female)

medical\_cost[condition]

medical\_cost[condition]['age'].mean()

**output:**

39.383720930232556

As both values give a round-down value of 39, we will impute 39 into all the missing values.

#filling empty values in mean age

medical\_cost['age'].fillna(value = 39,inplace= True)

medical\_cost

Finally, we will check again if all the missing values has been filled.

#checking if data has been replaced

medical\_cost['age'].isnull().sum(axis=0)

**output:**

0

Treating categories value

The second thing would be treating the categories value. We need first figure out the categorical column among the dataset using “d.types”

#Finding categorical columns

medical\_cost.dtypes

The output would show the type of column the dataset possesses,

PersonID int64

age float64

sex object

bmi float64

children int64

smoker object

region object

charges float64

dtype: object

Afterwards, the column “sex”, “smoker”, and “region” would be changed to a numerical value using ordinal encoding.

We have created a dummy dataset for the treatment for categorical value. For sex” and “smoker”, we will set a certain condition where if the condition is met, the number will be 1 and 0 if the condition is not met using the “astype” function.

updated\_file = medical\_cost.copy()

updated\_file ['smoker'] = (updated\_file ['smoker'] == 'yes').astype(int)

updated\_file ['sex'] = (updated\_file ['sex'] == 'male').astype(int)

updated\_file

For the “region” column, the “OrdinalEncoder” function from Scikit- learn has been used to help us transform the data, by encoding the category to their respective numerical value

# converting "region" category to a numberical value

from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder().set\_output(transform="pandas")

updated\_file[["region"]] = encoder.fit\_transform(updated\_file[["region"]])

updated\_file

We will change the “region” data type to an integer using “.astype” to standardise with the other categorical values

#changing it to a integer

updated\_file['region'] = updated\_file['region'].astype('int32')

updated\_file.dtypes

Standardising data

The final thing we would do is to standardise the values in the ‘age’ , ‘bmi’, ‘children’ columns so that the modal we use later would be more optimized instead of being biased towards one column with a higher value. We will use the standard scaler function in Scikit-learn for this.

#standardising using standardscaler

import sklearn

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

updated\_file[['age','bmi','children',]] = ss.fit\_transform(

updated\_file[['age','bmi','children',]])

updated\_file

(349 words)

(2)

**Figure 1.1: Charges Vs. Region**

A diagram of a graph

Description automatically generated with medium confidence

Figure 1.1 shows the charges incurred in medical insurance varying across 4 different region. From the figure 1.1, we can see that the median charges are similar at around $10,000 however, the length of the boxes in southeast is wider than the rest. This could suggest that the healthcare services in the southeast region vary among individuals which could be due to being an underdeveloped region. We can also see that there are some outliers of medical cost of above $50,000. This could be due to some individuals having unique healthcare circumstances or discrepancies during billing. The diamond marker on the boxplot refers to the mean charges and we can see that the mean charges is higher than the median charges of the medical bills which suggest that the distribution is more positively skewed, and this information could be used for a further analysis. The codes that create figure 1.1 is shown below:

plt.figure(figsize=(10, 5))

sns.boxplot(data = region\_charges, x = 'charges', y = 'region', showmeans = True)

plt.title(f'Charges vs Region')

plt.xlabel('Charges')

plt.ylabel('Region')

plt.show()

**Figure 1.2: A scatterplot of BMI vs Charges**

A diagram of blue dots and a red line

Description automatically generated

Figure 1.2 shows a scatterplot of BMI and Charges. The objective of this scatterplot is to show the correlation between BMI and charges. From the figure, we can see that there is positively linear relationship between BMI and the charges as the red regression is shifting upwards as BMI increases. This suggests that for each unit increases in the BMI, there is a corresponding increase in the medical charges. However, we can see that data points around BMI 35 have become very widespread and does not position exactly along the red regression line. This could mean although BMI contributes to the medical charges, the correlation between BMI and charges are weak, thus, suggesting that other factor that could influence the charges. In order to estimate insurance costs based on BMI, the graph is helpful for both insurance companies and individuals since it allows them to budget for potential increases in medical costs in the event that a policyholder has a high BMI. The codes that create figure 1.2 is shown below:

plt.figure(figsize=(10, 5))

# Create the scatter plot with a regression line

sns.regplot(data=medical\_cost, x='bmi', y='charges', line\_kws={'color': 'red'})

plt.title('BMI vs Charges')

plt.xlabel('BMI')

plt.ylabel('Charges')

plt.grid(True, linestyle='--', alpha=0.6)

plt.show()

**Figure 1.3: Mean Charges by Age groups**

**A graph of a number of people

Description automatically generated**

Figure 1.3 shows the mean charges of medical cost by different age groups. We have created a new column in our dataset to help us categorise different age groups. This is to help us visualise the graphs easier. From the figure 1.3, we can see that the highest mean charge is observed in the 60+ age group with a mean medical charges of $20670.80 while the lowest mean is 0-20 age group with a mean medical charges of $8343.11. From here, we can infer that as the age increases, the mean charges also increase as elderly people require more healthcare and services. This could be useful for those who are policy makers who are keen in finding potential targets for their health insurance scheme as from the figure, they can determine that age 60 and above have the highest medical charges. The codes to create a new column and for the graphs are shown below:

#creating a copy file for medical charges

medical\_charges = medical\_cost.copy()

medical\_charges

#create bin for age column

age\_bin = [0,20,30,40,50,60,100]

labels = ['0-20','21-30','31-40','41-50','51-60','60+']

medical\_charges['age.group'] = pd.cut(medical\_charges['age'], bins = age\_bin, labels = labels)

#plotting a bar graph of Mean charges over age

plt.figure(figsize=(10, 5))

ax = sns.barplot(data=medical\_charges, x='age.group', y='charges',ci=None,)

plt.title(f'Mean Charges by age')

plt.xlabel('Age')

plt.ylabel('Charges')

ax.bar\_label(ax.containers[0], fontsize =10);

plt.show()

(496 words)

(3)

Pre-processing task

To further explore the dataset using a decision tree, we will need to do some pre-processing task that is similar to question 1 but with some tweaks.

Firstly, we will firstly fill up the null value in the data in the “age” column with the mean age, similar to the question 1. Afterwards, we will convert only “sex” and “region” column to numerical. We will leave the “smoker” as a categorical value as this is our dependant variable.

Creating a Decision Tree

Our goal is to determine, based on the features given on the dataset, are we able to predict whether the individual is a smoker or non-smoker using a decision tree. We will first create our independent and dependant variable. The reason why we drop the ‘PersonID’ as it is a unique identifier in a dataset, thus, should not be included in our features.

# Feature Selection on data frame

Feature = new\_medical\_cost.drop( columns = ['PersonID', 'smoker'], axis = 1)

x = Feature

y = new\_medical\_cost['smoker']

We will then split the data frame into a training and testing set with an 80-20 split.

#Creating a train-test set (80-20 split)

from sklearn.model\_selection import train\_test\_split

random\_state = 0

test\_size = 0.2

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

x, y, test\_size = test\_size, random\_state = random\_state

)

At this stage, we will normalise the features to give it zero mean and unit variance.

#Normalising the datas to give zero mean and unit variance

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(x\_train)

X\_test = scaler.transform(x\_test)

Finally, we will create and fit a decision tree model. We will use a classifier since “smoker” is a categorical target value.

# Create and fit a decision tree model

from sklearn import tree

from sklearn.feature\_extraction import DictVectorizer

from sklearn.metrics import classification\_report

#We use a classifier for categorical target value

dtc = tree.DecisionTreeClassifier(criterion= 'gini', max\_depth=4 )

We will try to fit the decision tree on our training data.

#Fitting decision tree on our training data

dtc.fit(X\_train, y\_train)

Lastly, we will used the fitted decision tree to predict our test data. A classification report will be printed out to represent the model`s evalution.

# fit decision tree to predict on test data

Y\_predict = dtc.predict(X\_test)

#Print classification report

print(classification\_report(Y\_predict,y\_test))

The output will be shown below:

precision recall f1-score support

no 0.95 0.98 0.96 201

yes 0.92 0.85 0.88 67

accuracy 0.94 268

macro avg 0.94 0.91 0.92 268

weighted avg 0.94 0.94 0.94 268

(255 words)

(4)

A diagram of a diagram

Description automatically generated with medium confidence The plotted decision-tree will be shown below:

In this decision tree, we would be focusing on the root node where there are 1072 samples and among these 1072 samples 858 samples are classify as non-smokers while 214 are classify as smokers. We can see that there are 12 leave nodes in this decision tree. The reason for this amount of leave nodes is because we have set a pre-specified depth of 4 such that it would not be oversimplify or overcomplicate. We set it up at 4 as we can see that already 8 nodes that has a Gini index of 0, which suggests that there are no more impurities detected. Since the target variable is categorical, the predicted value would be the mode.

(124 words)

(5)

I think that decision trees can be effective used for exploratory data analysis. Exploratory data analysis is the initial process of doing preliminary investigations in order to identify patterns, identify anomalies, test hypotheses, and verify assumptions (Patil, 2018). This initial process is done in python by looking for missing data, understanding the variable importance, outlier detection etc.

In the decision tree, we can do a missing data analysis by adjusting the algorithm in the classification and regression tree (CART) diagram for example, deciding how the missing values from the root node will be passed to the child node (Inside Learning Machines, 2023). This allows the model to predict even with there is missing data as the missing data will be treated as another branch in the tree. We can also the mean of the target variable by looking at the leaf nodes if the decision tree is predicting a numeric variable (Wu & Zhu, 2023).

The decision tree is also able to help us determine the relationship between the variables which is useful for exploratory data analysis. By creating a classification report in our decision tree, we are able to make sense of how the dependent variable is affected by the independent variables by looking at the precision and weighted average.

(211 words)

Inside Learning Machines. (2023, May 29). *Can decision trees handle missing values?* https://insidelearningmachines.com/decision\_trees\_handle\_missing\_values/#Classification\_Example\_with\_Missing\_Values

Patil, P. (2018, March 24). *What is exploratory data analysis?*. Medium. <https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15>

Wu, K. Y., & Zhu, S. (2023). ANL252 Python for data analytics. Singapore University of Social Sciences.