TP4- Decision tree

Parcours Progis

Etudes, Medias, communication, Marketing

Bahareh Afshinpour.

02.12.2024

References

- https://www.datategy.net/2024/05/31/predict-customer-life-time-value-using-ai/

- https://www.kaggle.com/datasets/pankajjsh06/ibm-watson-marketing-customer-value-data/code

CLV (customer lifetime value)

- Customer lifetime value, in short CLV, is a prediction of how much an average customer will spend on company products or services over the entire relationship with company ‘s business.

- It is crucial to have a good understanding of how much value each customer is going to bring to the company.

- Calculating CLV is important because it allows businesses to see how much profit they're making from each customer during their relationship, which provides valuable data for the marketing and sales efforts.

CLV (customer lifetime value)

There are multiple ways to calculate CLV.

- One way is to find the customer's average purchase amount, purchase frequency, and lifetime span, and do a simple calculation to get the CLV.

100 per months *5 times*12*20years

120000

https://www.tidio.com/blog/customer-lifetime-value/

- we can build machine learning models that can predict customers' CLV over the certain period.

Understanding and improving CLV in a company can increase company profits, help plan budgets, and analyze customer satisfaction.

your task at hand is to build a high-performance and interpretable machine learning model to predict the CLTV based on user and policy data.

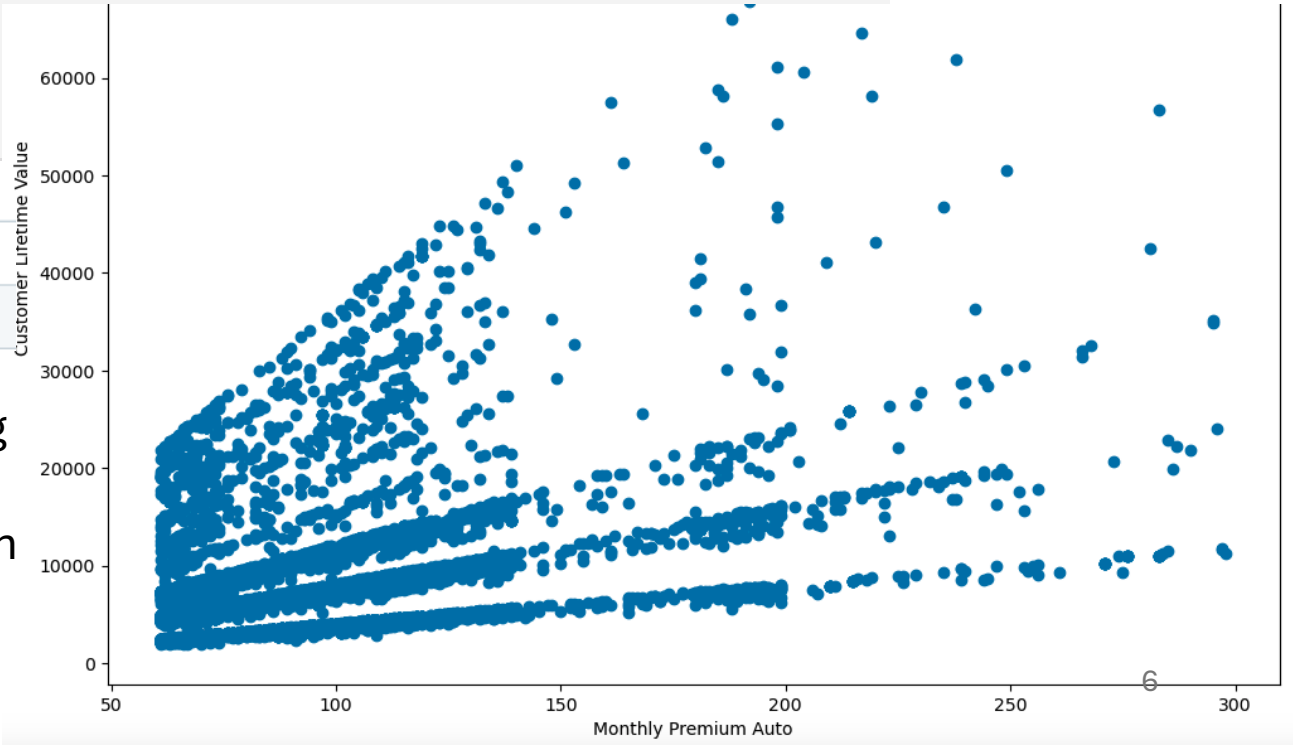
You are provided with the sample dataset of the company holding the information of customers and policies such as the highest qualification of the user, total income earned by a customer in a year …..

Customer liftetime value has positive correlation with monthly premium Auto feature

```
import matplotlib.pyplot as plt
plt.figure(figsize = (12, 9))
plt.scatter(data['Monthly Premium Auto'], data['Customer Lifetime Value'])
plt.ylabel('Customer Lifetime Value')
plt.xlabel('Monthly Premium Auto')
plt.show()
```

Number of Policies	Float	Number of policies owned by the customer
Monthly Premium Auto	Float	Monthly premium paid by the insured

In the **Monthly Premium Auto** feature, a spreading pattern is formed where
the higher the vehicle insurance premium paid each month, the more valuable the customer is.



```
X = data['Monthly Premium Auto'].values.reshape(-1,1)
y = data['Customer Lifetime Value'].values.reshape(-1,1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
```

▾ LinearRegression

LinearRegression()

```
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

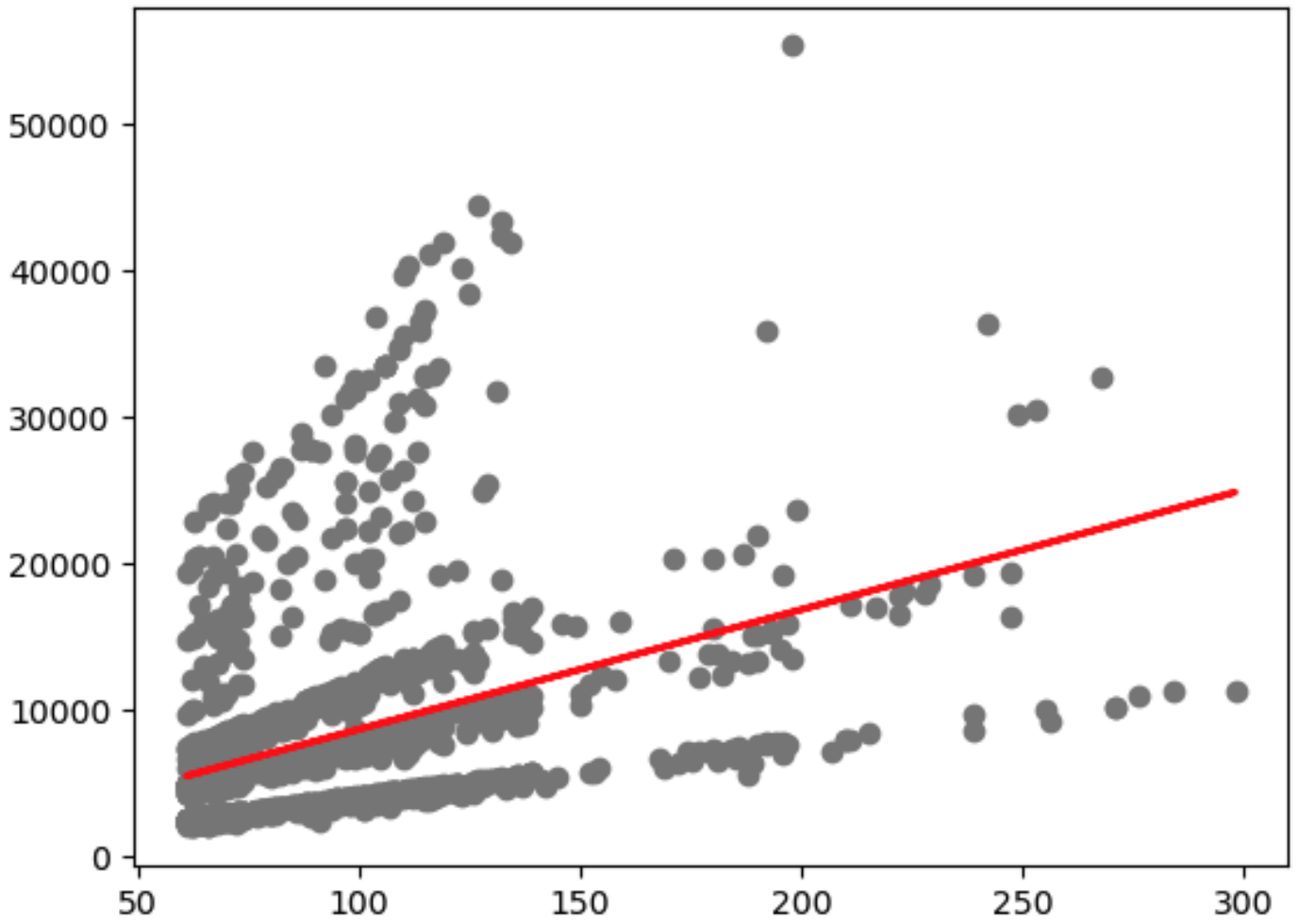
```
[432.67220054]
[[81.74976994]]
```

```
y_pred = regressor.predict(X_test)
```

```
import sklearn.metrics as metrics
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Root Mean Squared Error: 6123.7178203315325
```

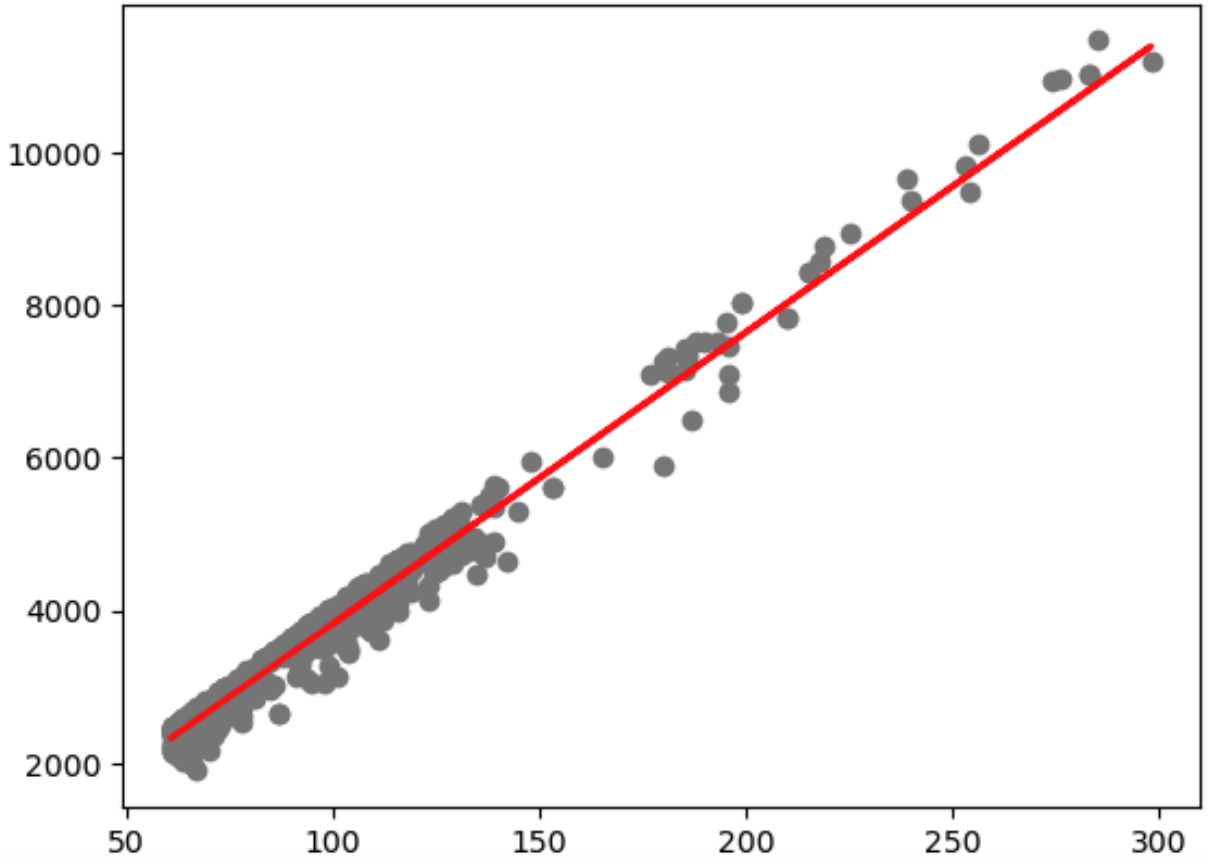
```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



It is not good
We can do
Linear Regression By filtering with 'Number of Policies'

```
data_filter=data[data['Number of Policies']==1]
```

```
X = data_filter['Monthly Premium Auto'].values.reshape(-1,1)
y = data_filter['Customer Lifetime Value'].values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
y_pred = regressor.predict(X_test)
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



In the **Number of Policies** feature, customers with 1 policies are more valuable than customers with other policies. Customers who have more than 1 policies can be assumed as 'low-value' customers.

- Factors positively influencing CLV include Monthly Premium and Number of Policies, while Open Complaints and Claim Amount have the potential to decrease CLV. These factors should be carefully managed to optimize customer value.

FUNCTION

There are two kinds of functions in Python.

-Built-in functions that are provided as part of Python

- print(), input(), type(), float(), int() ...

- Functions that we define ourselves and then use

FUNCTION

-In Python a function is some **reusable** code that takes arguments(s) as input, does some computation, and then returns a result or results

-We define a function using the **def** reserved word

-We **call/invoke** the function by using the function name, parentheses, and arguments in an expression

Function example

Creating a Function

```
def my_function():
  print("Hello from a function")
```

Calling a Function

```
.....
....
my_function()
```

Arguments

- An argument is a value we pass into the function as its input when we call the function

Parameters

A parameter is a variable which we use in the function definition. It is a “handle” that allows the code in the function to access the arguments for a particular function invocation.

```
>>> greet('en')
Hello
>>> greet('fr')
Bonjour
>>>
```

Argument

Parameter

```
>>> def greet(lang):
...     if lang == 'en':
...         print('Hello')
...     elif lang == 'fr':
...         print('Bonjour')
```

Return Values

```
def my_function(x):
  return 5 * x



print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Decision Tree example

Importing Libraries

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the Data for ML

[5]:
```
data = pd.read_csv('./Insurance.csv')
data.head()
```

[5]:

	id	age	sex	bmi	children	smoker	region	charges
0	C140	19	female	27.900	0	yes	southwest	16884.92400
1	C117	18	male	33.770	1	no	southeast	1725.55230
2	C128	28	male	33.000	3	no	southeast	4449.46200
3	C147	33	male	22.705	0	no	northwest	21984.47061
4	C116	32	male	28.880	0	no	northwest	3866.85520

We are building a model
to predict
the medical charges
encountered by
various patients.

Nous créons un modèle
pour prédire les frais médicaux des patients.

```
data.shape
```

```
(1338, 8)
```

Defining Independent and Target Feature

- We drop the charges feature.

```
X=data.drop(['charges'],axis=1)
Y=data[['charges']]
```

X

	id	age	sex	bmi	children	smoker	region
0	C140	19	female	27.900	0	yes	southwest
1	C117	18	male	33.770	1	no	southeast
2	C128	28	male	33.000	3	no	southeast
3	C147	33	male	22.705	0	no	northwest
4	C116	32	male	28.880	0	no	northwest

Y Target feature

charges
16884.92400
1725.55230
4449.46200
21984.47061
3866.85520

```
Y['charges'].mean()
```

```
13270.422265141257
```

Check for Missing Values

- isnull : the percentage of rows which are missing for each feature

```
X.isnull().mean()
```

```
id          0.0
age         0.0
sex         0.0
bmi         0.0
children    0.0
smoker      0.0
region      0.0
dtype: float64
```

```
Y.isnull().mean()
```

```
charges    0.0
dtype: float64
```

Non of the features have any missing value

It is always a good idea to keep checking

Split the data into Numeric and Categorical Features

- The next step is to split the X data frame, which is our independent data frame, into two parts:

```
num=X.select_dtypes(include="number")
char=X.select_dtypes(include="object")
```

```
char.head()
```

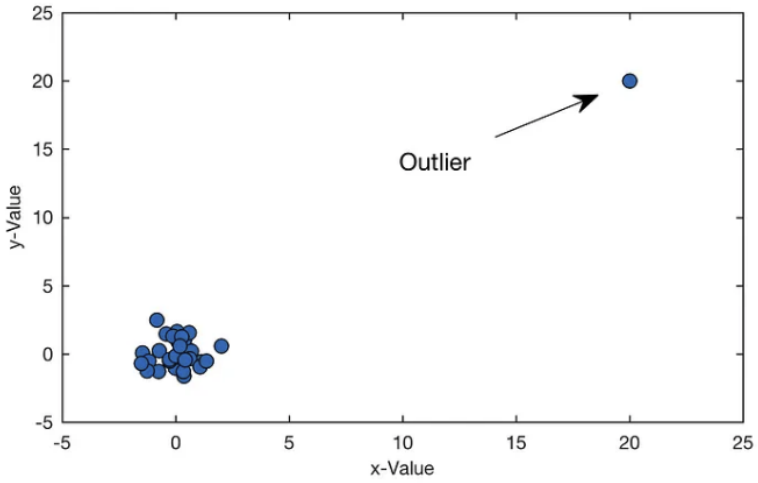
	id	sex	smoker	region
0	C140	female	yes	southwest
1	C117	male	no	southeast
2	C128	male	no	southeast
3	C147	male	no	northwest
4	C116	male	no	northwest

num ：
Select those features from X, which are numerics and put them together in to num dataframe

Remove extreme value and outlier from numerical features

- What is outlier?
  - an outlier is a data point that differs significantly from other observations.
  - An outlier is a single data point that goes far outside the average value of a group of statistics.

We need to take care of the outliers




```
num.describe()
```

	age	bmi	children
count	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918
std	14.049960	6.098187	1.205493

What to do with the undroppable outliers?

- **Impuation:** We can replace the outlier values with the mean, median or mode value based on the use case.
- **Quantile-based Flooring and Capping:** In this technique,we can do the flooring (e.g., replacing with the 10th percentile) for the lower values and capping (e.g.,replacing with the 90th percentile) for the higher values.

https://medium.com/analytics-vidhya/outlier-detection-in-machine-learning-382557c775aa

What is Flooring and capping technique

- Flooring: The lower bound threshold value (min) will take the place of any extreme values that are less than the min value (lower bound).

- Capping: Any extreme values higher than the max value (upper bound) will be replaced with the upper bound threshold value (max).

Remove extreme value and outlier from numerical features

```
num.describe(percentiles=[0.01,0.05,0.25,0.50,0.75,0.95,0.99])
```

	age	bmi	children
count	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918
std	14.049960	6.098187	1.205493
min	18.000000	15.960000	0.000000
1%	18.000000	17.895150	0.000000
5%	18.000000	21.256000	0.000000
25%	27.000000	26.296250	0.000000
50%	39.000000	30.400000	1.000000
75%	51.000000	34.693750	2.000000
95%	62.000000	41.106000	3.000000
99%	64.000000	46.407900	5.000000
max	64.000000	53.130000	5.000000

- By using this percentiles, they are aditional points present in the describe
- Now, we want to look at the 99th percentile value and the maximum percentile value

We should restrict the feature distribution within the 1% value and 99% value

The upper end of the distribution, Is composed of some extreme values

Flooring and caping

We have 1338 patient, 99% of them have BMI , less than 46.4 and only 1% of them have BMI more than 46.4

If there is any difference between these two, it means: there are few data are above the 99% of the data

Removal of Extreme Values and outliers

- Feature value:
  - Not be allowed to go beyond the upper threshold
  - Not be allowed to go below the lower threshold

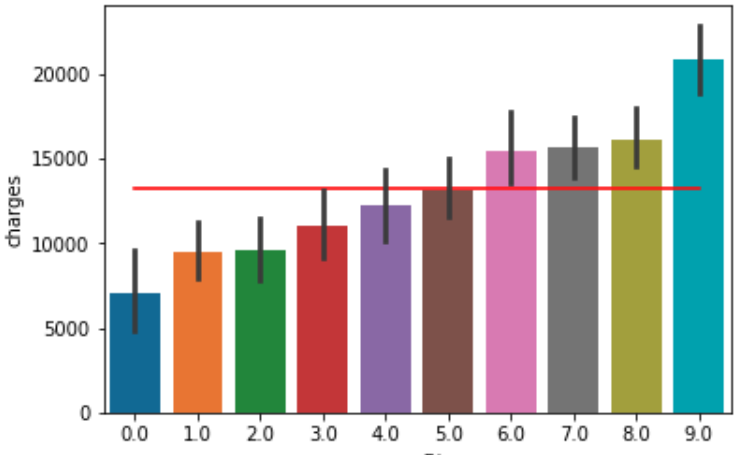
We will call the outlier cap function,
We use it as a lambda process across all the columns

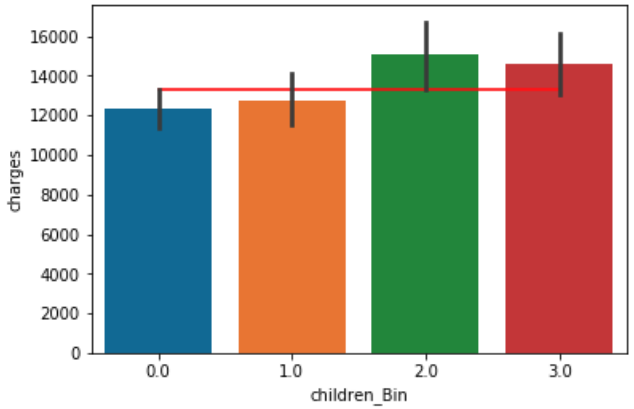
```
def outlier_cap(x):
    x=x.clip(upper=x.quantile(0.99))
    x=x.clip(lower=x.quantile(0.01))
    return(x)
```

```
num=num.apply(lambda x : outlier_cap(x))
num.describe(percentiles=[0.01,0.05,0.25,0.50,0.75,0.95,0.99
```

	age	bmi	children
count	1338.000000	1338.000000	1338.000000
mean	39.207025	30.649718	1.094918
std	14.049960	6.025065	1.205493
min	18.000000	17.895150	0.000000
1%	18.000000	17.917294	0.000000
5%	18.000000	21.256000	0.000000
25%	27.000000	26.296250	0.000000
50%	39.000000	30.400000	1.000000
75%	51.000000	34.693750	2.000000
95%	62.000000	41.106000	3.000000
99%	64.000000	46.330977	5.000000
max	64.000000	46.407900	5.000000

We are going to drop two unimportant columns. (Children and region)
Why?






```
num_good=num.drop(['children'],axis=1)
char_good=char.drop(['region','id'],axis=1)
```

```
num_good.head()
```

	age	bmi
0	19	27.900
1	18	33.770
2	28	33.000
3	33	22.705
4	32	28.880

```
char_good.head()
```

	sex	smoker	region
0	female	yes	southwest
1	male	no	southeast
2	male	no	southeast
3	male	no	northwest
4	male	no	northwest

Encode the Categorical Features

```
char_encode=pd.get_dummies(char_good,drop_first=True)
char_encode.head()
```

	sex_male	smoker_yes
0	False	True
1	True	False
2	True	False
3	True	False
4	True	False

Build the complete feature set

```
X_all_good_features=pd.concat([char_encode,num_good],axis=1,join="inner")
X_all_good_features.head()
```

	sex_male	smoker_yes	age	bmi
0	False	True	19	27.900
1	True	False	18	33.770
2	True	False	28	33.000
3	True	False	33	22.705
4	True	False	32	28.880

Splitting the data into Train and Test Sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X_all_good_features,Y,test_size=0.3,random_state=20)
```

```
print("Shape of Training Data",X_train.shape)
print("Shape of Testing Data",X_test.shape)
print("Average Salary in Training Data",y_train.mean())
print("Average Salary in Testing Data",y_test.mean())
```

```
Shape of Training Data (936, 4)
Shape of Testing Data (402, 4)
Average Salary in Training Data charges    13287.364425
dtype: float64
Average Salary in Testing Data charges    13230.974848
dtype: float64
```

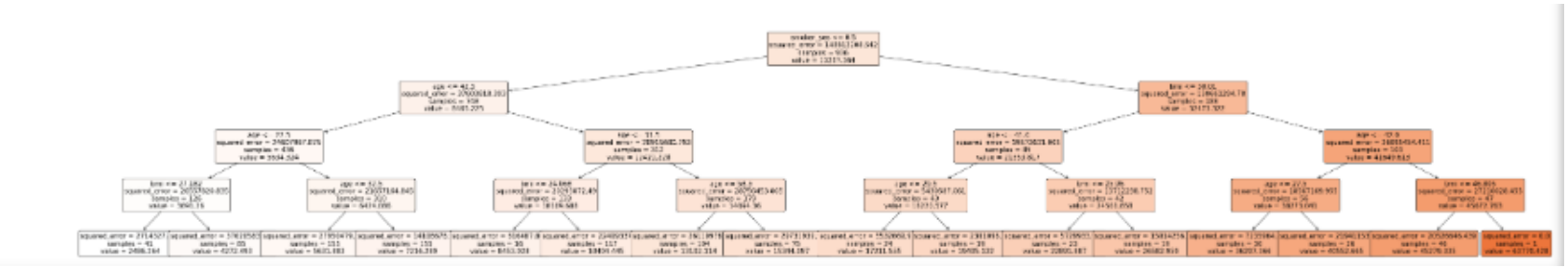
Fitting the Decision Tree Regression Model

```
from sklearn.tree import DecisionTreeRegressor
regrtree=DecisionTreeRegressor(max_depth=4,random_state=20)
regrtree.fit(X_train,y_train)
```

Visualizing the Decision Tree

```
from sklearn import tree
!pip install pydotplus
import pydotplus
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(60,10))
#_ = tree.plot_tree(regr, feature_names=X_all.columns, filled=True)
tree.plot_tree(regrtree,filled=True,rounded=True,feature_names=X_all.columns,fontsize=15)
plt.show
```



Evaluating the Model

```
regr_pred_train=regrtree.predict(X_train)
regr_pred_test=regrtree.predict(X_test)
regr_pred_all=regrtree.predict(X_all_good_features)
X_all_good_features['regr_pred_charges']=pd.DataFrame(regr_pred_all,index=X_all_good_features.index)
```

```
from sklearn.metrics import r2_score
r_sq_train=r2_score(regr_pred_train,y_train)
r_sq_train
```

```
0.8351484601848287
```

```
from sklearn.metrics import r2_score
r_sq_test=r2_score(regr_pred_test,y_test)
r_sq_test
```

```
0.8609041498681215
```

End