**FINAL PROJECT REPORT**

**INFO 5502-PRINCIPLES AND TECHNIQUES OF DATA SCIENCE**

**Prof. Dr. Ting Xiao**

**Project Title:** Prediction of Medical Insurance Cost

**Participants and Individual Contribution :**

**Neha Reddy Rasapalle -** 11525110 - [nehareddyrasapalle@my.unt.edu](mailto:nehareddyrasapalle@my.unt.edu)

Involved in initial stage of the project for the data pre-processing and data-exploration.

**Sravani Akkiraju -**  11476837 - [sravaniakkiraju@my.unt.edu](mailto:sravaniakkiraju@my.unt.edu)

Involved in the coding part, training and testng the models.

We together worked on project presentation and report

**Abstract :**

Insurance is a policy that reduces or eliminates the expenses of losses caused by different risks. The cost of insurance is influenced by a number of things. These factors have a role in the creation of insurance policies. Machine learning (ML) can improve the efficiency of insurance policy language in the insurance business. This research shows how different regression models can anticipate insurance prices. We'll also compare the outcomes of several models, such as Linear Regression and the Decision Tree technique.

**Data Specification :**

Here, we are going to use the ‘insurance.csv’ data set which is downloaded from Kaggle.

Data Source Link - <https://www.kaggle.com/datasets/mirichoi0218/insurance>

Insurance.csv file is obtained from the Machine Learning course website (Spring 2017) from Professor Eric Suess at<http://www.sci.csueastbay.edu/~esuess/stat6620/#week-6>.

This dataset contains 1338 observations (rows) and 7 features (columns). The dataset contains 4 numerical features (age, BMI, children, and expenses) and 3 nominal features (sex, smoker, and region) that were converted into factors with numerical values designated for each level. As the target variable is continuous, we are going to use supervised learning algorithms like Linear Regression and Decision Tree Algorithms.

**Project Design :**

We are going to use Jupyter Notebook for the entire analysis and a few other libraries which are needed for the analysis like NumPy, pandas, Matplotlib, sklearn, etc..

We are going to use two supervised learning techniques for the predictions. They are Linear regression and Decision Tree Algorithms. Before applying the model to the data set. We need to make sure that the data is free from outliers or any other missing data without any errors to avoid the wrong predictions.

**Loading The Data:** Initially, we are going to read the data set into the Jupyter notebook using pandas. Using function info(), we can see that there are 1338 entries and 7 columns in the entire data set.

**Data Pre-Processing :**

**Checking Null Values:** To check whether we have any missing values in the data set. We are going to use ‘isnull’ function. We can see that there are no missing values, which is good to go further.

To get more insights of what are the variables, We have separated both Categorical and Numerical columns and generated the mean, median, standard deviation for the numerical variables using the ‘describe’ function

**Explanatory Data Analysis and Visualization :**

Import all the libraries used for the visualization like matplotlib and seaborn.

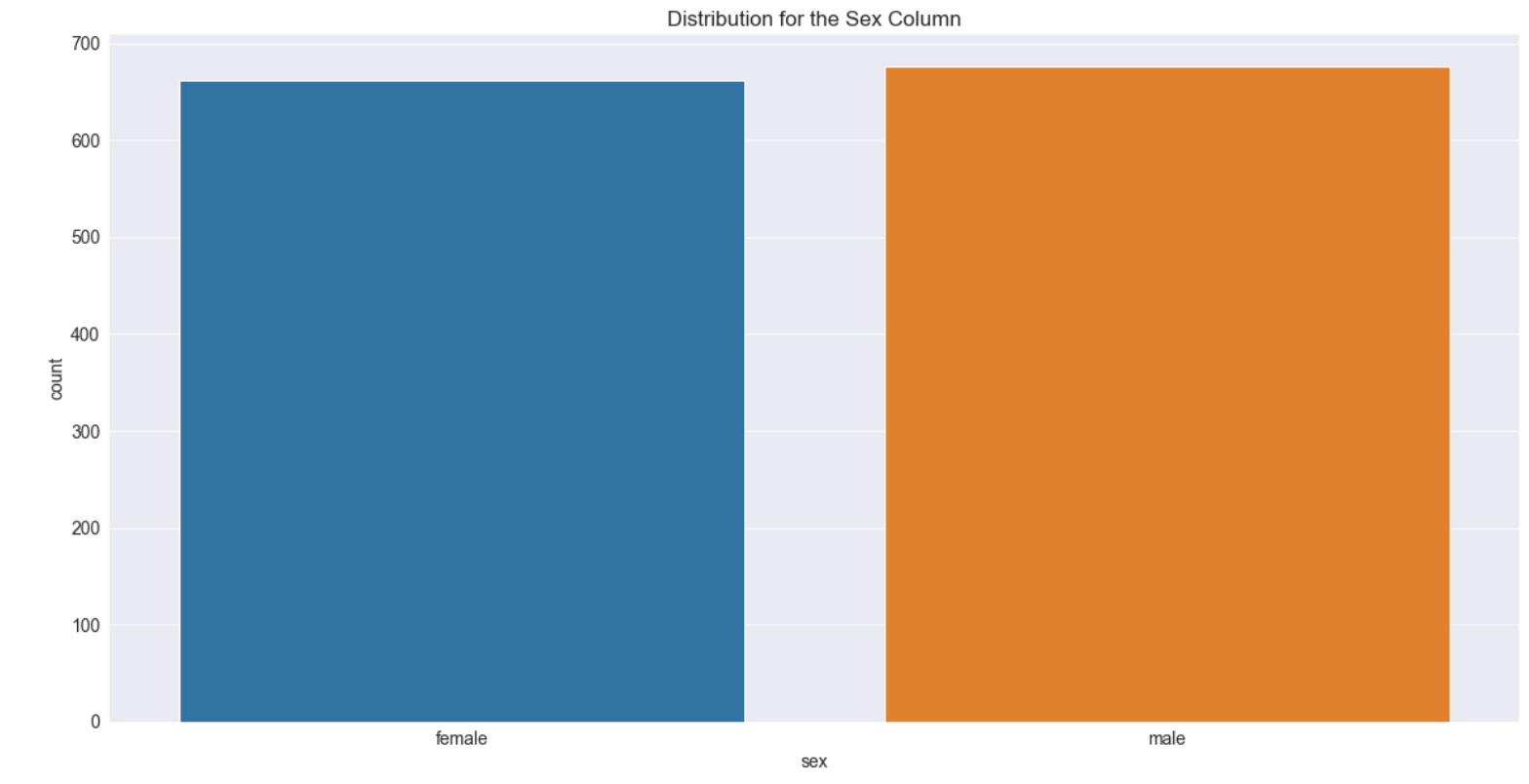
Matplotlib is a NumPy-based multi-platform data visualization framework. One of the most significant advantages of visualization is that it provides us with visual access to massive volumes of data in simply understandable graphics. Matplotlib has a variety of plots such as line, bar, scatter, histogram, and so on.

Seaborn is a data visualization tool, but the main distinction is that it focuses on the key elements of data exploration and interpretation, as well as providing attractive default styles and color palettes to make statistical plots more appealing. It is based on the matplotlib software and is tightly connected with pandas data structures.

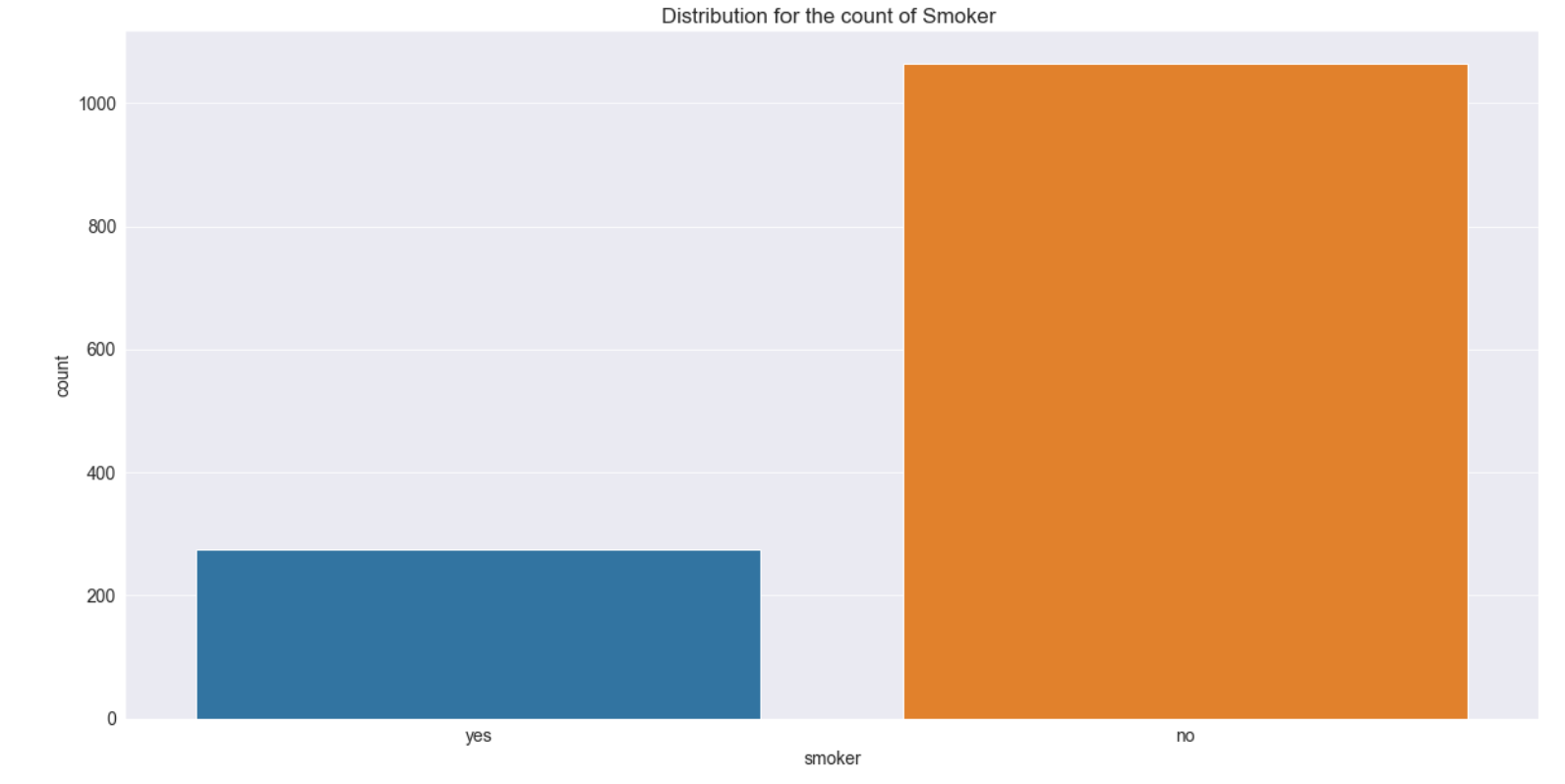
**Univariant Analysis :**

Univariant analysis is done only on one variable. We can use barplot, histogram, pie chart for the univariant analysis

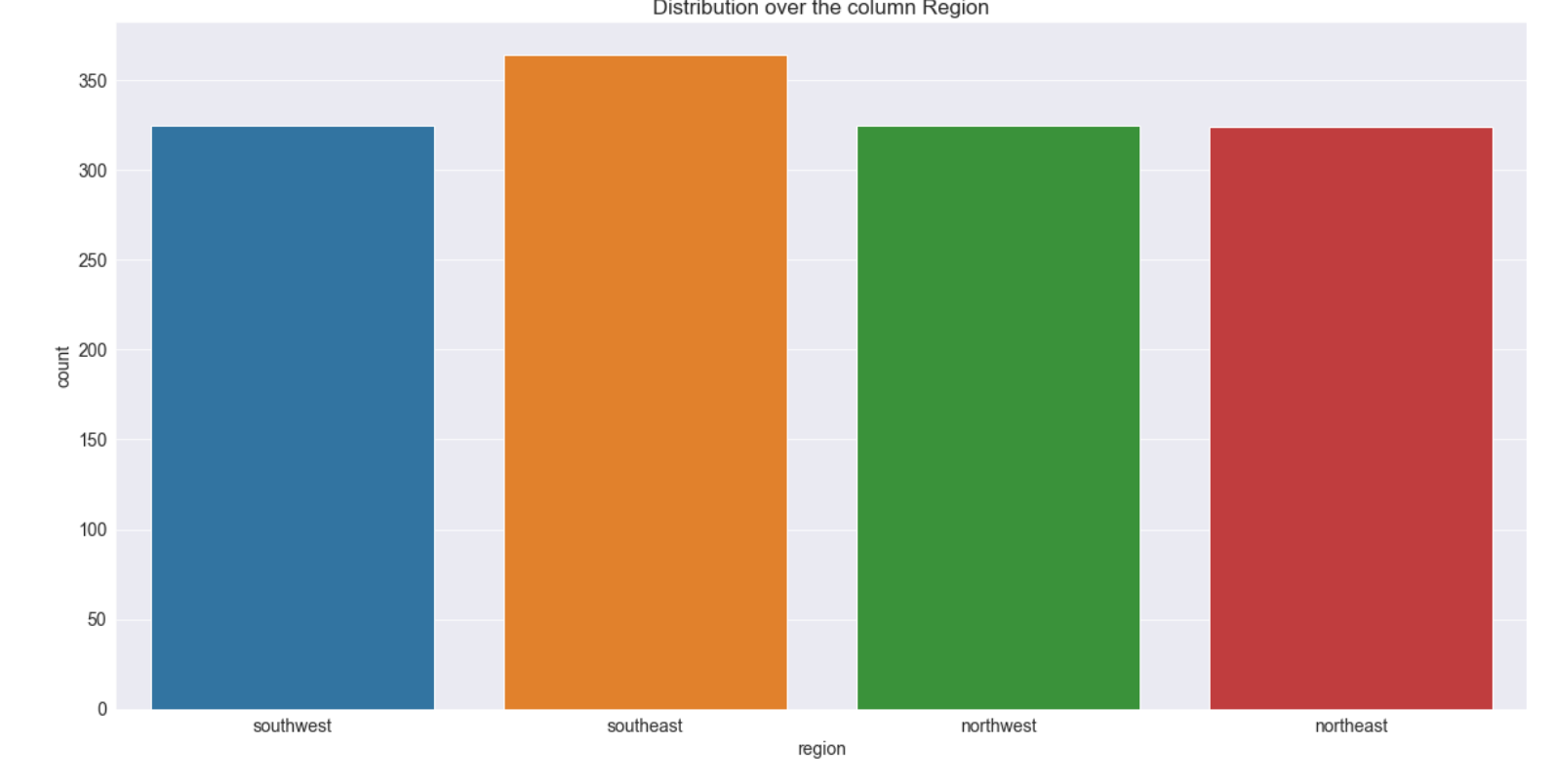
We used a count plot from the seaborn library to visualize the distribution of sex. From this plot, we can say that among the total children 49.4% are females, and the rest of the 50.5% of them are males.



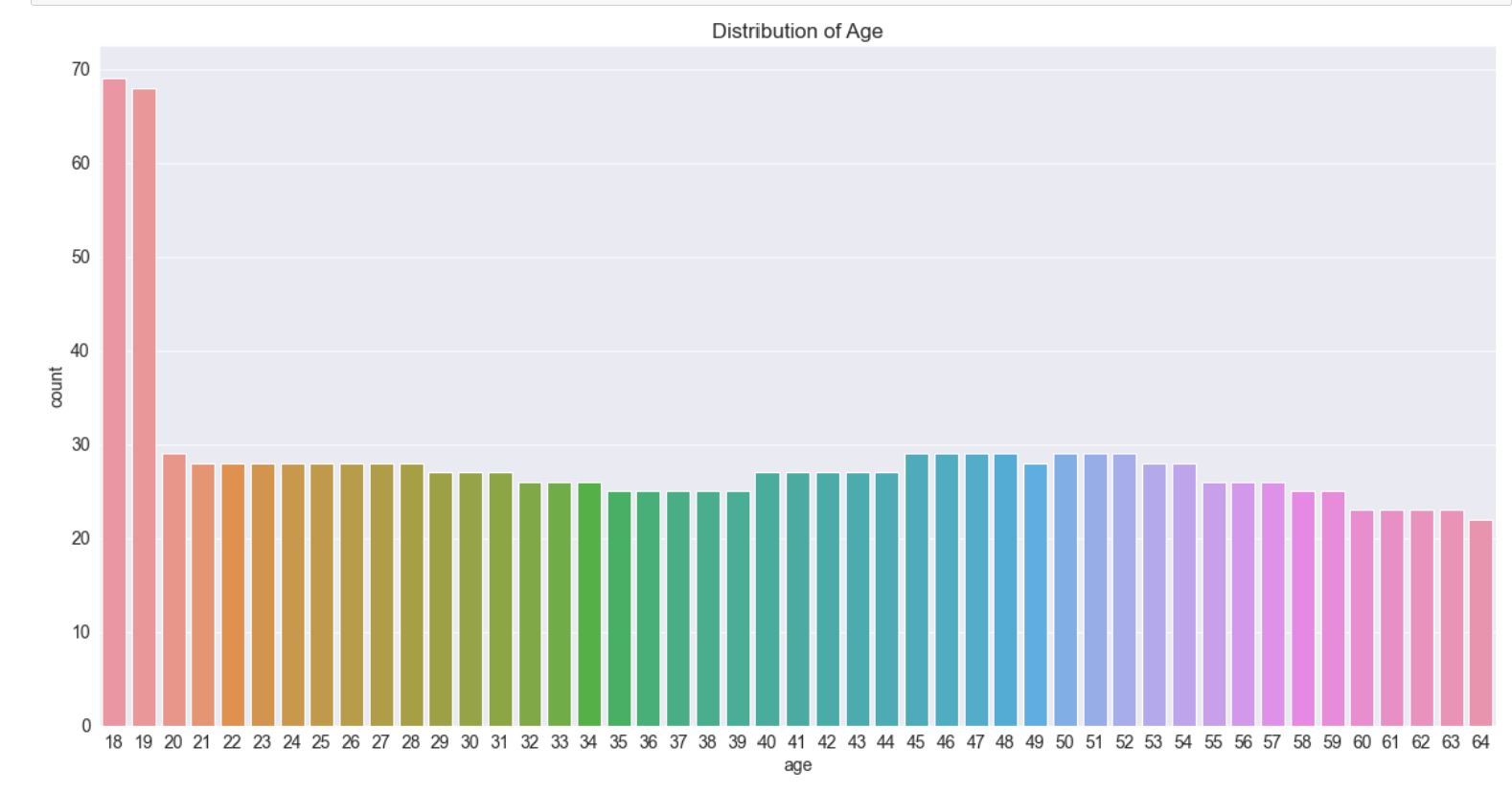
In the same way, 20.4% of the children have a habit of smoking, and 79.5% don't have a habit of smoking.



From the distribution of regions, we can observe that most of them are from the southeast region. Whereas for the southwest and northwest the count is the same and the last one is the northeast with a very minute difference.

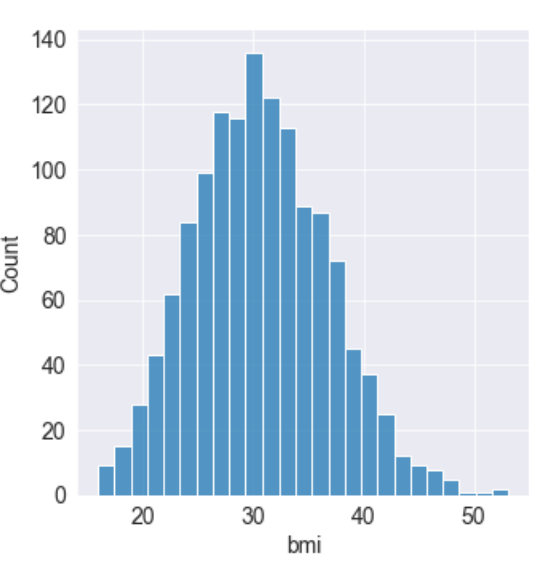


Using the unique function, we can say that there are 47 different ages in the data set. So, using bin size as 47 for the histogram to visualize the distribution of Age.

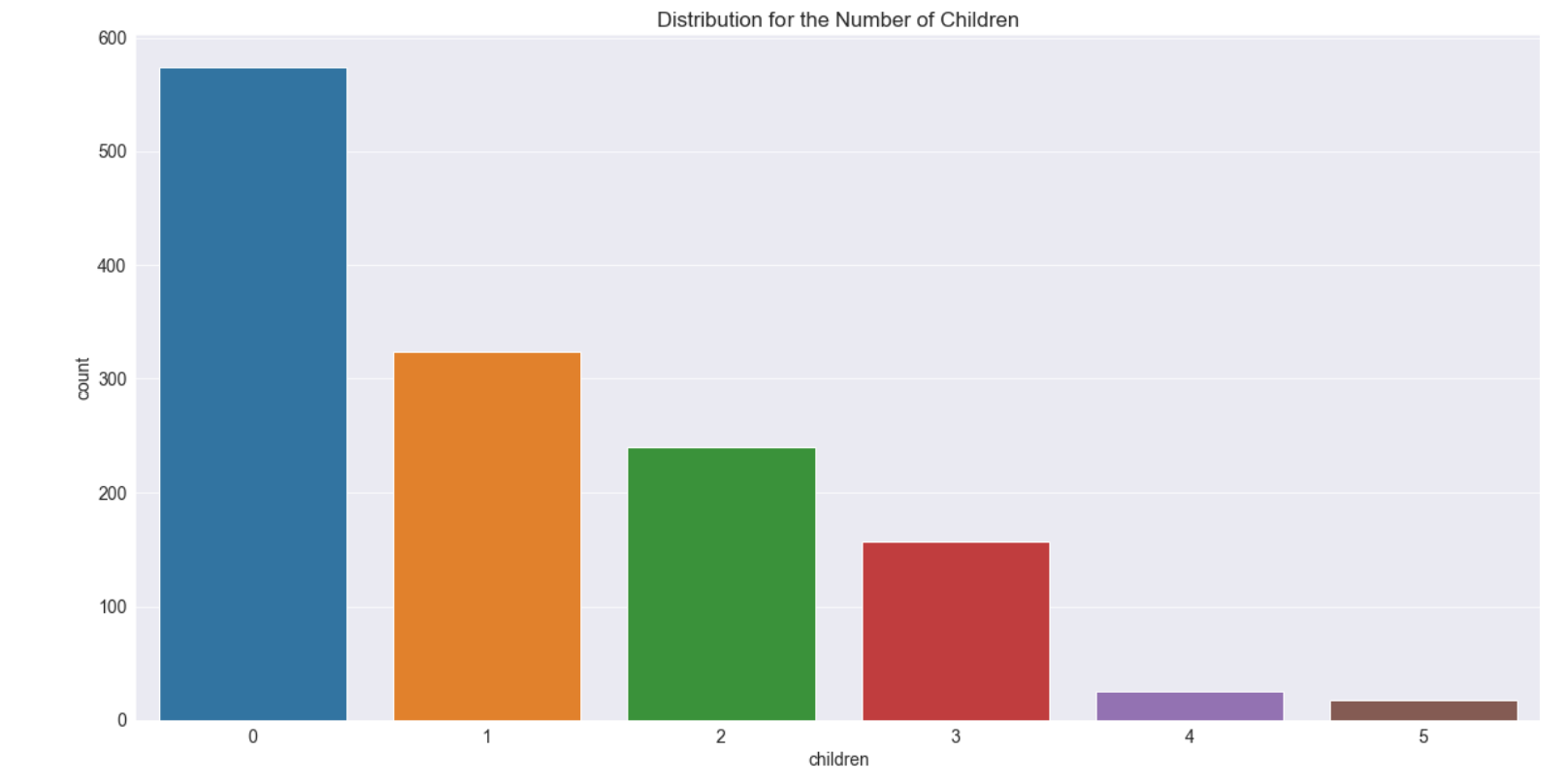


We can see that, ages above 19 has almost a uniform distribution.

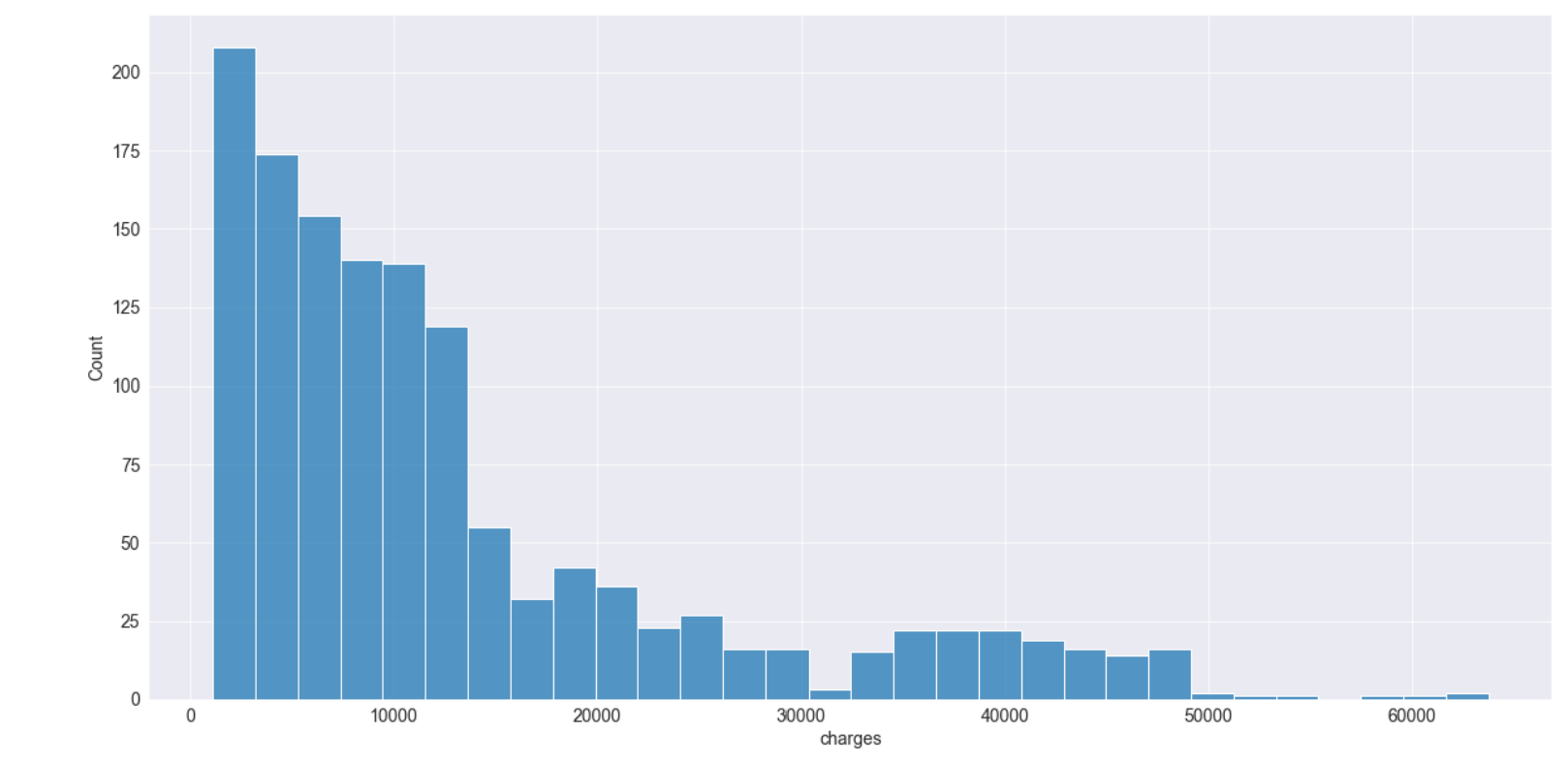
Using displot from the seaborn library, we can see the normal distribution of bmi.



From the distribution of children, we can observe that maximum count of children are 0 with 574 and the least are of 5 with 18 numbers.



And finally, distribution of the target variable which is charges. The max value of charges is 63770.



**Bi-Variant Analysis :**

Scatter Plot is used for the Bi-Variant Analysis. It helps to find relationships between two or more variables. Perform different Analysis on Age and bmi with charges with respect to smoker, sex, region and children and checking which variable impacts the target variable charges the most.

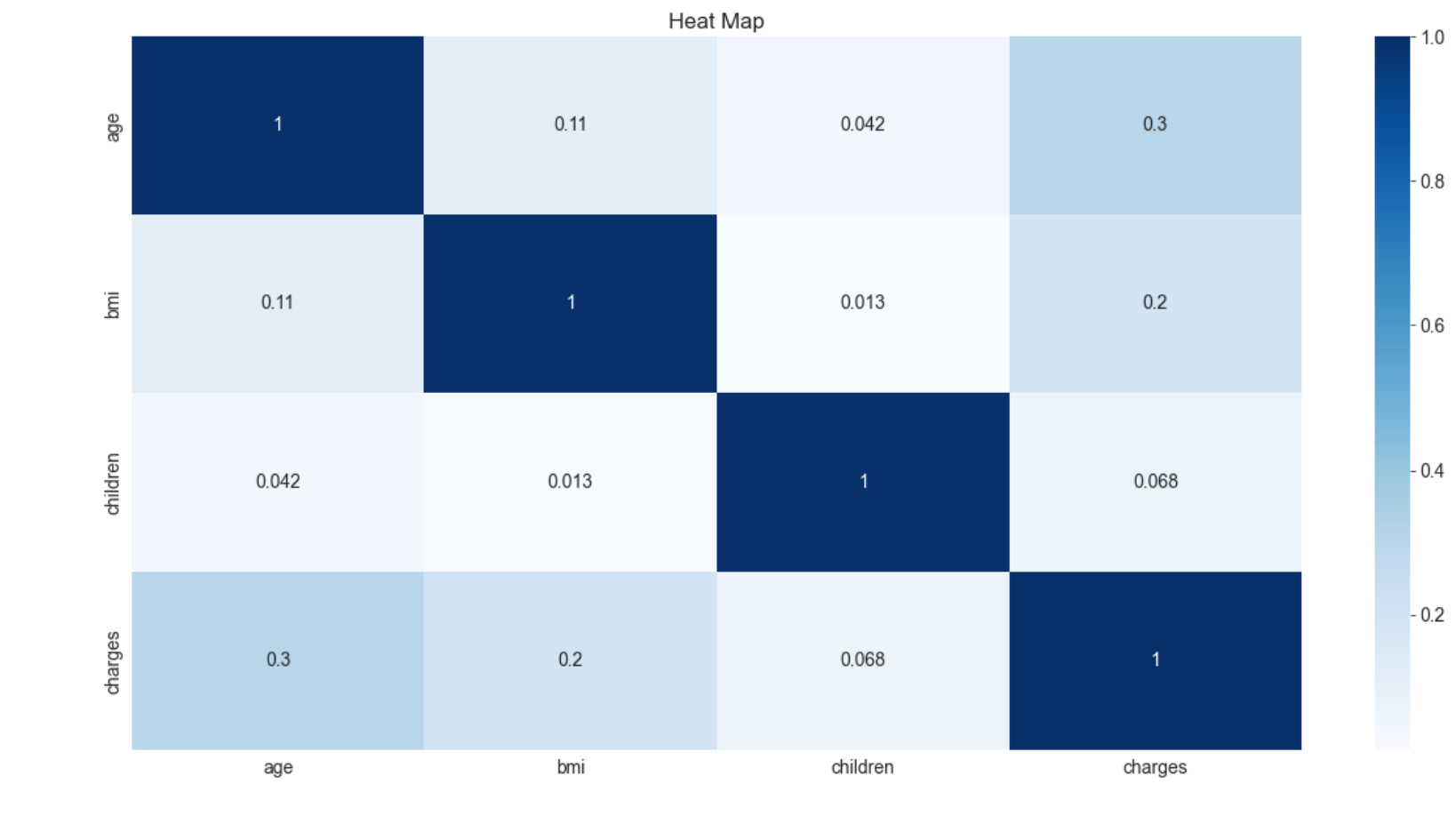
Correlation is used to check how one or more variables are related to reach other. In general, the values of correlation range from -1 to 1.

The Correlation Coefficient greater than zero indicates positive relationship which means if value of one variable increases, the other also tends to increase.

Correlation Coefficient less than zero indicates negative relationship which means if value of one variable increases, the other also tends to decrease in opposite direction.

If the correlation coefficient is equal to zero, then there is no relationship between the variables. They are simply randomly distributed without any trend of increase or decrease with respect to another variable.

A heatmap is a graphical representation of data that use a color-coding method to depict various values.



Using heatmap, we can clearly observe that age and charges are moderate positive relationship with eachother with the value of 0.3 which is highlighted in dark color compared to other values. Whereas, children is the least correlated one which is very lightly colored in heat map. We can conclude that, older people uses money from insurance compared to the younger ones.

**Performing One-Hot Encoding to Categorical Variables :**

One-hot encoding is a wonderful way to turn some of these category data into numerous binary features, which can then be fit into a linear regression using the presence or absence of each categorical unit. Sometimes, categorical variables may be good features to be considered. But, it may not fit due to its nature. So,to convert this categorical to binary values.One-Hot encoding is great technique.

After that, we will Remove the duplicate values if we have any in the data set.

After performing encoding, drop the categorical columns which are already encoded to avoid viewing multiple unwanted columns.

Separating Target Variable from rest of the variables. Here, target variable is charges.

**Preparation of Train, Test and Validation data :**

The Train-Valid-Test split is a method for evaluating your machine learning model's performance which is fast and easy.

Train Data Set : Data set utilized for learning (by the model), i.e. fitting the parameters to the machine learning model.

Validation Data Set : Set of data used to offer an unbiased evaluation of a model fitted on the training dataset when adjusting model hyperparameters, as well as other aspects of model preparation including feature selection and threshold cut-off selection.

Test Data Set : A set of data that can be utilized to offer an unbiased evaluation of a final model that has been fitted to the training dataset.

Using sklearn library with train\_test\_split function splitting the data set into train with 80% and test data with 20%. And then using the same function, split into train and validation set with 25%

**Linear Regression Model :**

It is a supervised learning technique used to find linear relationship between one dependent variable with several other independent variables.

Steps Involved in this model :

From sklearn library, import linear regression model and fit the model to the train data.

Loading the data set, calculating predicted values on train and validation data set.

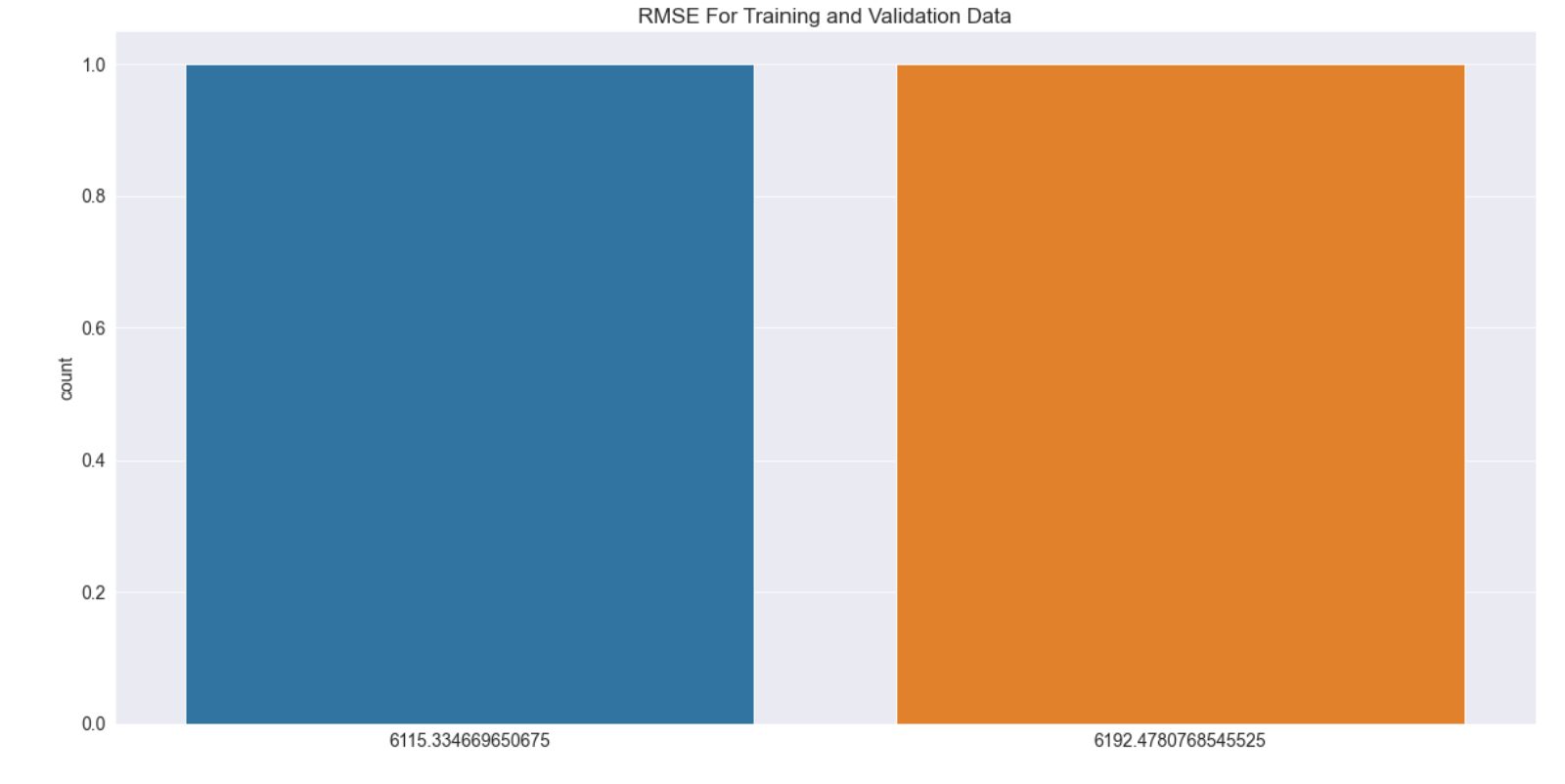
**Calculating Loss Function :**

When dealing with numeric variables, we must quantify the losses quantitatively, which means that simply knowing if the anticipated value is incorrect is insufficient; we must also determine the amount of variation between our predicted value and the actual value in order to train our model effectively.

So, to calculate loss function we have several techniques like Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) etc.

Here, we are using MSE to calculate loss function. For Validation set, the MSE value is 6115.334669650675 and for the Training set, the MSE value is 6192.4780768545525.

So, we have low MSE values. Which is good i.e less difference between actual and predicted values.



From the above plot, we can see that we have significantly less root mean squared error

**Hyper Parameter Tuning For Linear Regression Model :**

Hyper Parameter Tuning is used to increase the model performance by avoiding overfitting and to reduce the variance error. Variance error is the value that predicted value changes when different training data is used.

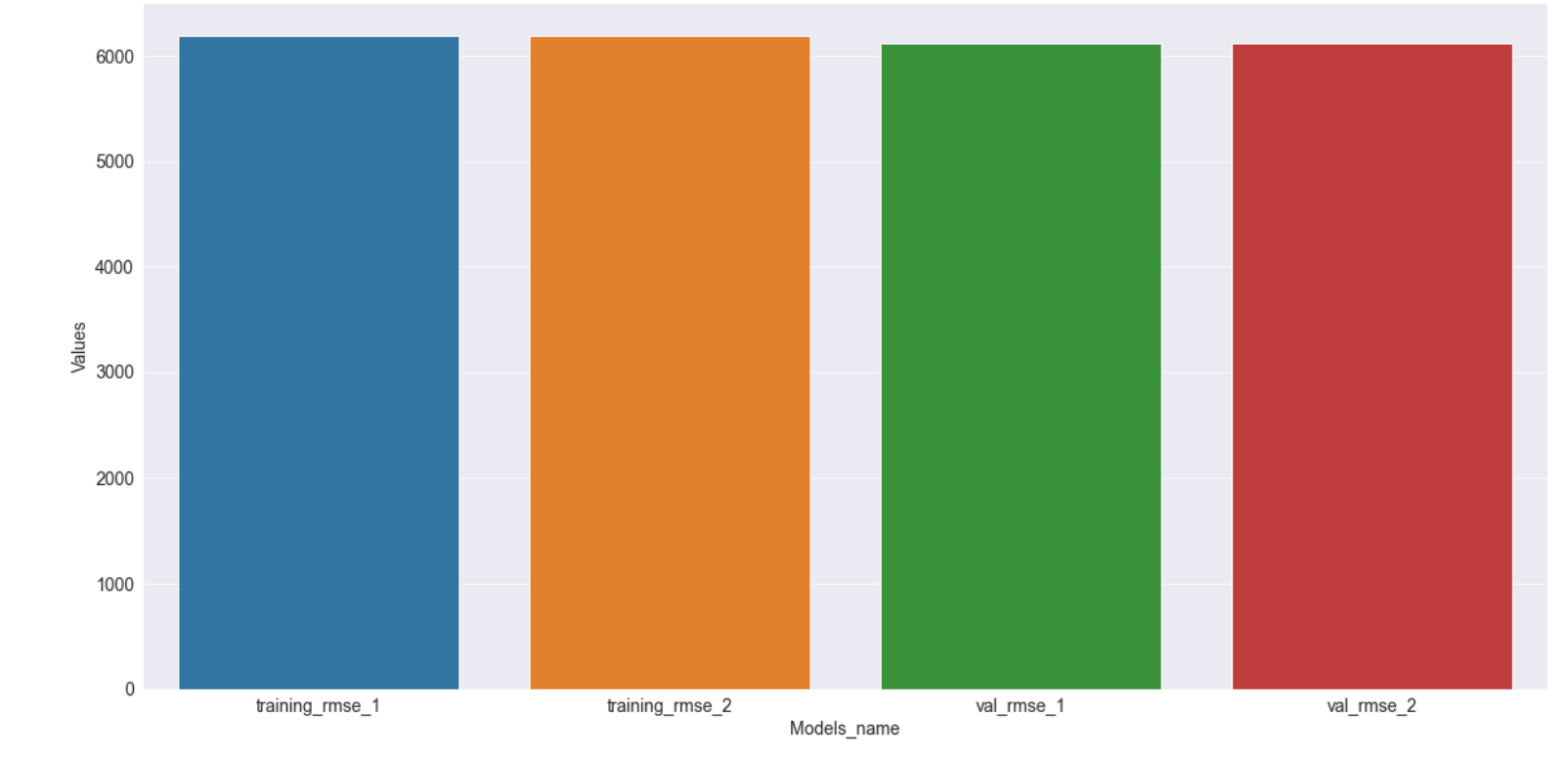
Hyper Parameters which needs to be tuned

* Normalize
* fit\_intercept

**Tuning the Hyper Parameter - Normalize :**

Tuning the Parameter when Normalize is true and when it is false on training data.

Visualizing the values after tuning using bar plot.

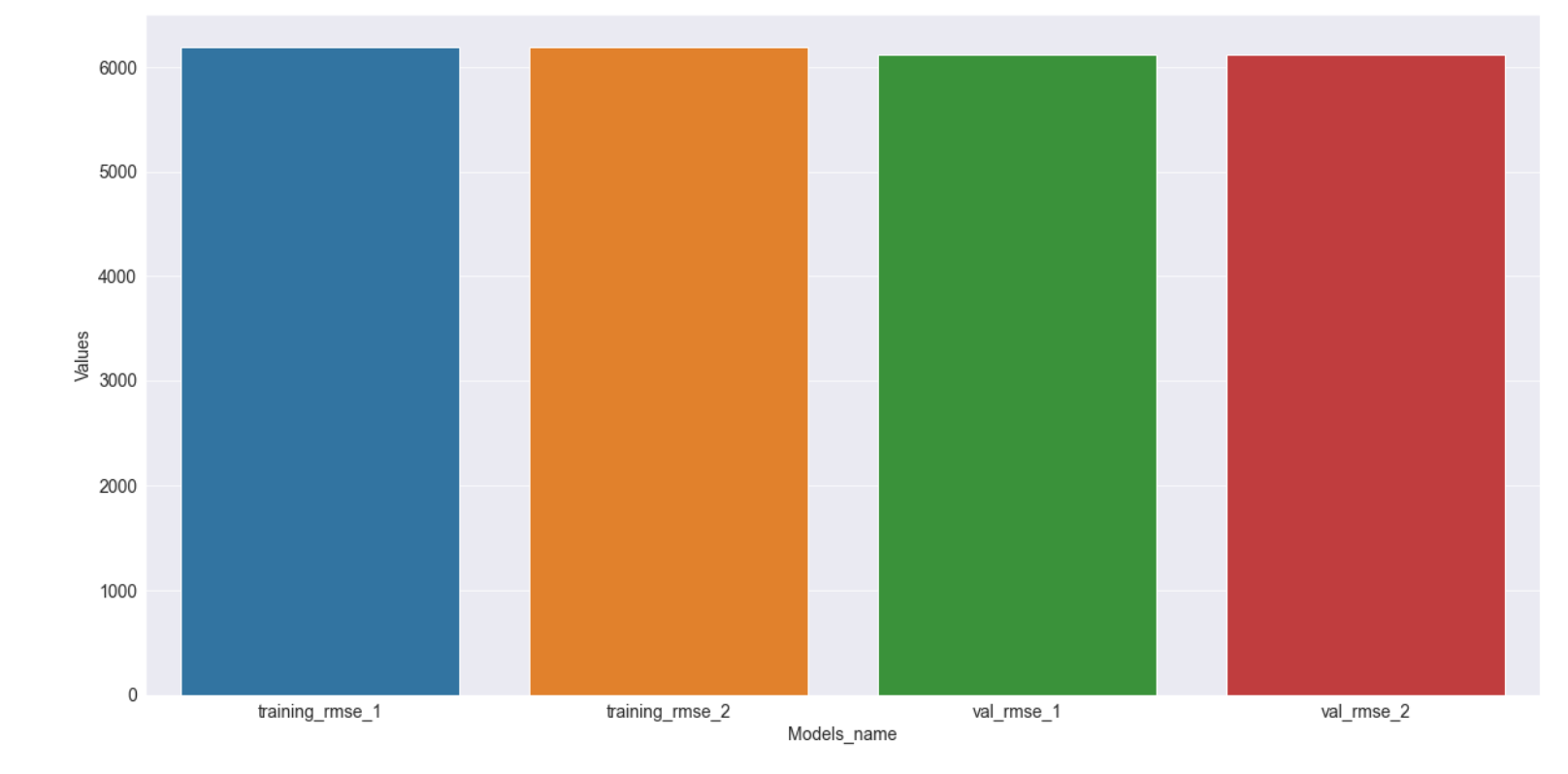


There is not much difference even after tuning the hyper parameter ‘Normalize’

**Tuning the Hyper Parameter - fit\_intercept :**

Tuning the Parameter when fit\_intercept is true and when it is false on training data.

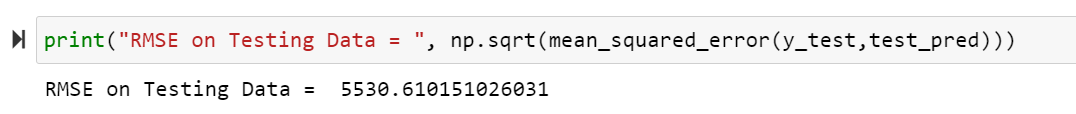
Visualizing the values after tuning using bar plot.



We can see that, there is not much difference even after tuning the hyper parameters. So, let’s go with the default model for the prediction.

**Final Prediction of Linear Regression Model on Test Data set.**

Using this linear regression model for the prediction on the test data.



The root mean squared error we got on testing data is 5530.610151026031

We are now predicting with a single model. Let's look for another model and see which one has the best performance.

**Decision Tree Model :**

The most powerful and widely used tool for categorization and prediction is the decision tree. A decision tree is a flowchart-like tree structure in which each internal node represents an attribute test, each branch reflects the test's conclusion, and each leaf node (terminal node) stores a class label.

Steps Involved in this model :

From sklearn library, import Decision Tree Model and fit the model to the train data.

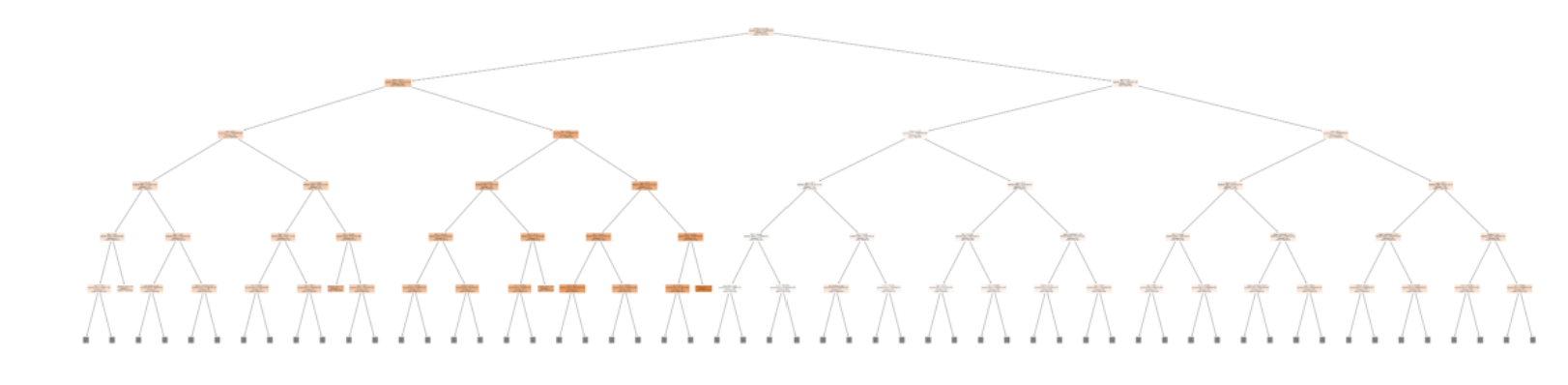
By Calculating the error between train data and validation data before and after prediction.

We got RMSE values of 571 for train data and 6332 forvalidation set.

We can say from the value of RMSE that the decision tree model is overfitted to the training data. So, we will tune the hyperparameters to reduce the overfitting

**Visualizing the Decision Tree :**

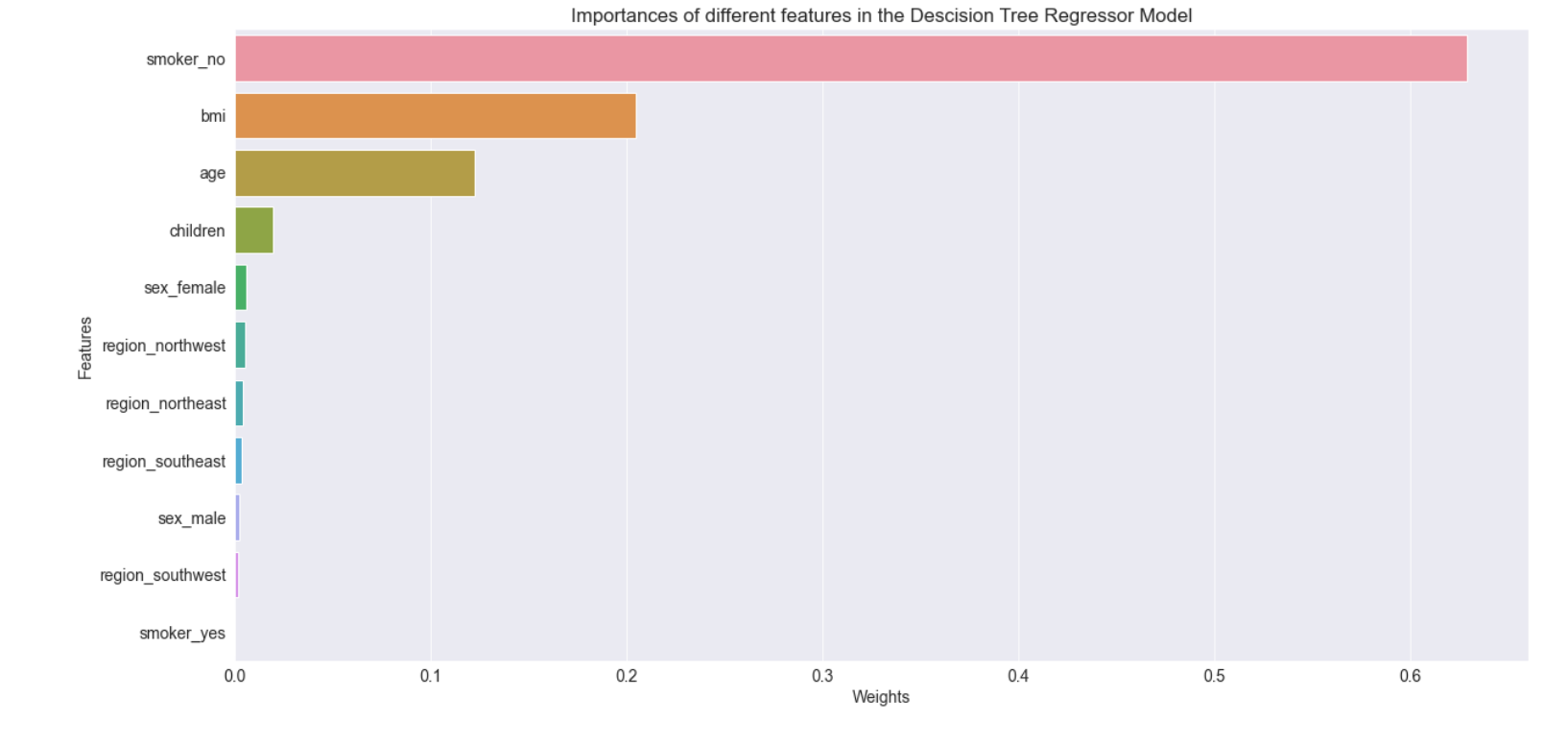
From sklearn library, import plot\_tree and export\_text for visualizing tree.



**Feature importances :**

Feature importance contributes to data and model insight, as does feature selection, which can improve the efficiency and effectiveness of a predictive model on the problem.

Visualizing the feature importances graphically using seaborns library



**Hyper Parameter Tuning For Decison Tree Model :**

Hyper Parameters which need to be tuned

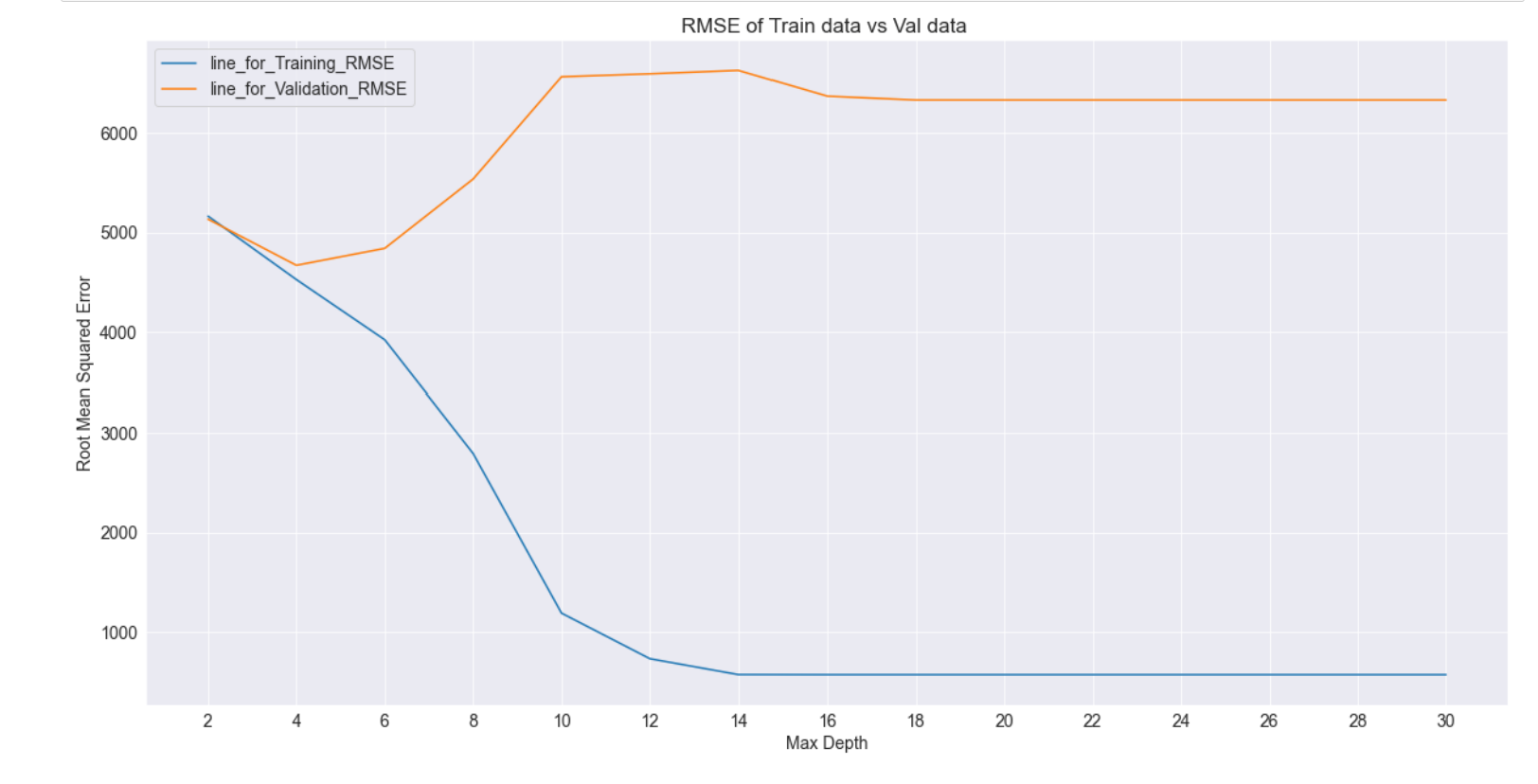
* max\_depth
* max\_leaf\_nodes

**Tuning the Hyper Parameter - Normalize :**

A tree's max depth is defined as the longest path between the root and leaf nodes. Using this parameter, I can specify how deep I want the tree to grow.

Here, we are using maximum depth to be 31.

Getting values of RMSE for both training, validation set and visualizing it graphically.



When max depth = 4, we can see that our model performs best on the validation set. It can also be seen that when the value of max depth increases, our model begins to overfit the training data. As a result, we'll fine-tune our model based on its efficiency in validation data.

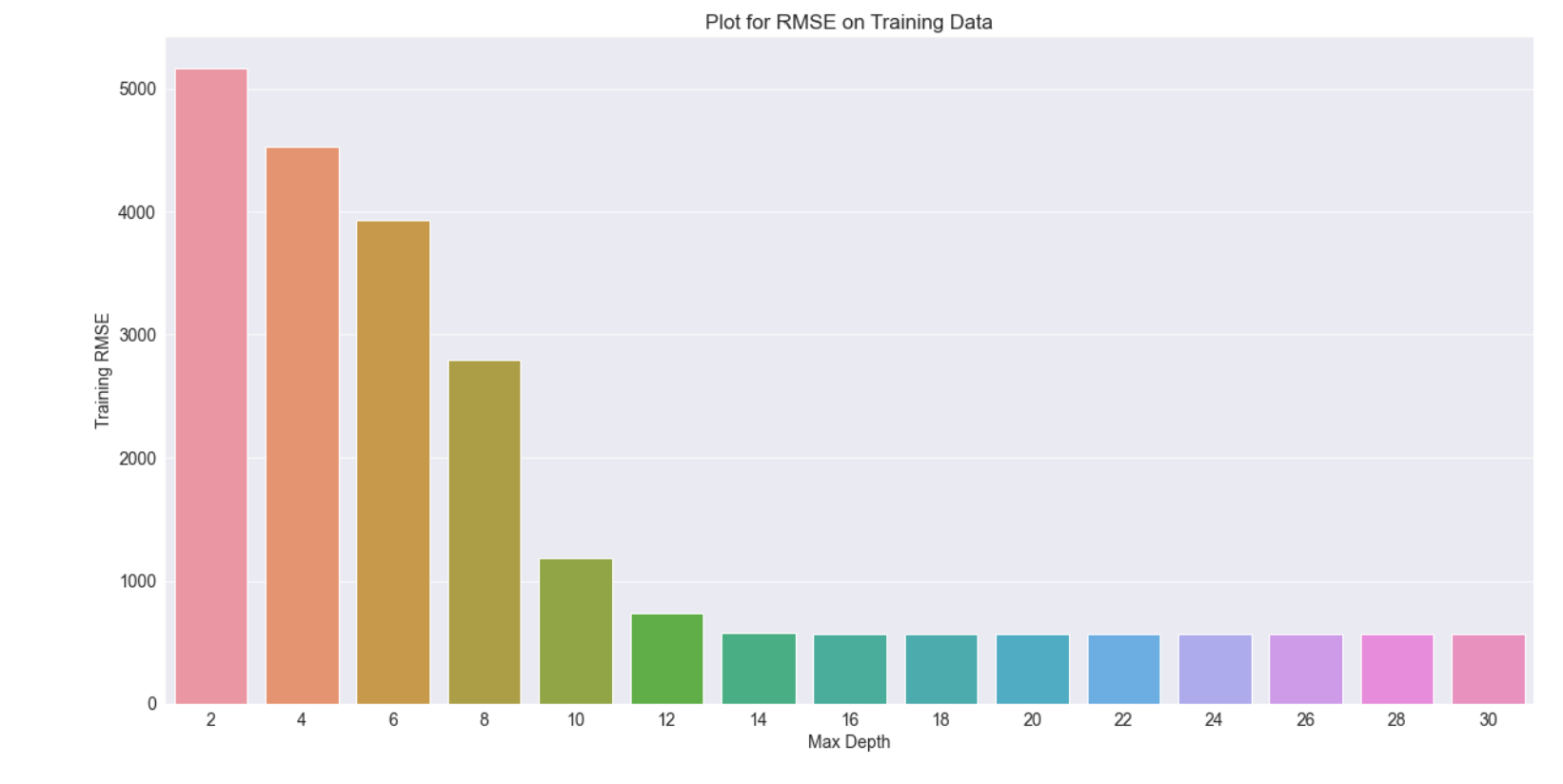
Max depth is one of the decision tree parameters that works greatly by reducing the growth of the Decision Tree.

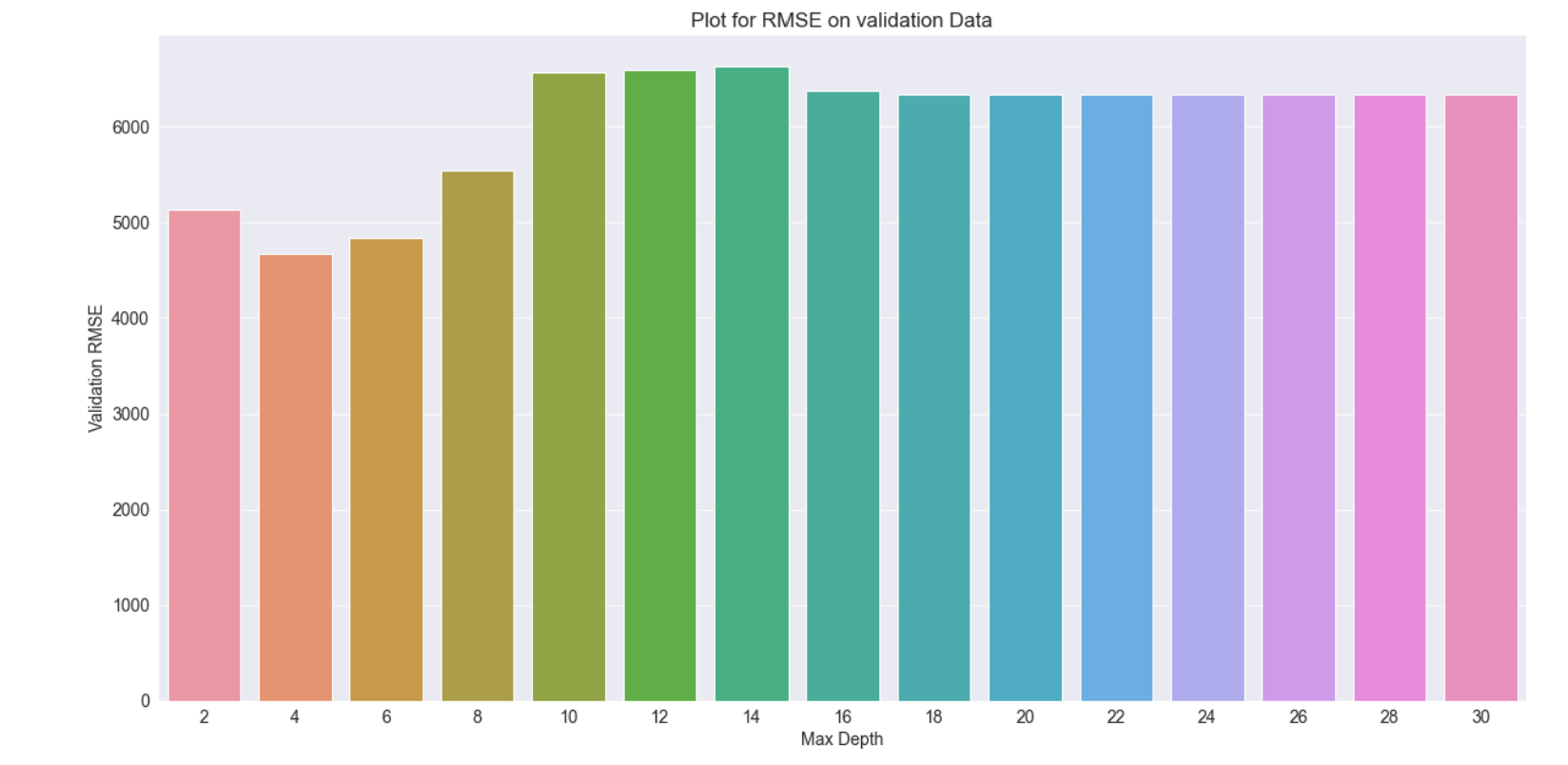
**Tuning the Hyper Parameter - max\_leaf\_nodes :**

This hyperparameter restricts the growth of the tree by imposing a condition on the splitting of nodes in the tree. If we have more terminal nodes than the specified number of terminal nodes after splitting, the splitting will be stopped and the tree will not grow further.

Here, we are using maximum leaf nodes to be 10

Getting values of RMSE for both training, validation set and visualizing it graphically.



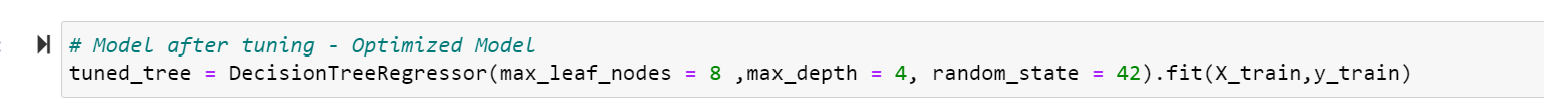


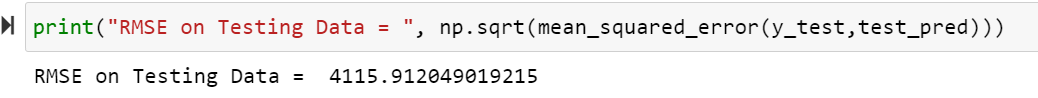
From the above barplots, When max leaf nodes = 8, our model performs best on both the training and validation sets. As a result, we'll adjust our model accordingly.

So, After tuning the parameters. We set the maximum depth to be 4 and maximum leaf nodes to be 8

**Final Prediction of Decision Tree Model on Test Data set.**

Now, let’s use the final model after tuning the hyper parameters for the prediction on the testing data.

The root mean squared error we got on testing data is 4115.912049019215



**Final Conclusion :**

We can clearly see that, the rmse value for Linear Regression Model is 5530.610151026031 and for Decision Tree Model is 4115.912049019215. Root mean squared error (RMSE) is less for Decison Tree Model compared to Linear Regression Model. So, we can select decision tree algorithm for this data set.

**Repository / Archive:**

[**https://github.com/rasapallenehareddy/Prediction-of-Medical-Insurance-Cost/blob/main/5502-FinalProject\_Notebook%20(1)%20(2).ipynb**](https://github.com/rasapallenehareddy/Prediction-of-Medical-Insurance-Cost/blob/main/5502-FinalProject_Notebook%20(1)%20(2).ipynb)

**Appendix:**

# Importing all the necessary libraries

import numpy as np

import pandas as pd

import os

# Reading the File

df = pd.read\_csv('insurance.csv')

df.shape

df.head()

df.describe()

df.info()

df.isnull().sum()

df.columns

# Categorical Columns

categorical\_cols = df.select\_dtypes(include = 'object').columns

df[categorical\_cols] = df[categorical\_cols].astype('string')

categorical\_cols

# Numerical Columns

numerical\_cols = df.select\_dtypes(include = np.number).columns

numerical\_cols

# # Data Visualization

# Importing the Visualization Libraries

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

get\_ipython().run\_line\_magic('matplotlib', 'inline')

sns.set\_style('darkgrid')

sns.set\_style('darkgrid')

plt.rcParams['font.size'] = 14

plt.rcParams['figure.figsize'] = (20, 10)

plt.rcParams['figure.facecolor'] = '#00000000'

df.sex.value\_counts()/len(df.sex)

# From the below distribution, we can clearly say that the ratio of female and male is almost the sam

# For Sex Column, plotting a countplot

sns.countplot(x = 'sex', data = df).set(title = 'Distribution for the Sex Column');

# Smoker's count

yes = df[df.smoker == 'yes'].smoker.value\_counts()/len(df.smoker)

yes

# Non-Smoker's count

no = df[df.smoker == 'no'].smoker.value\_counts()/len(df.smoker)

no

# 79.52% of children are not smoking and rest of the 20.47% are smoking

# For Smoker Column, plotting a countplot

sns.countplot(x = df.smoker).set(title = 'Distribution for the Smoker')

# For Region Column

df.region.value\_counts()

sns.countplot(x = df.region).set(title = 'Distribution of Region');

# From the above graph, we can clearly say that majority of the people are from southeast region

# For Age Column

sns.countplot(x = df.age).set(title = 'Distribution of Age');

# From the above graph, we can say that the age is uniformly distributed

# For BMI Column

sns.displot(x = df.bmi)

# For Children Column

sns.countplot(x = df.children).set(title = 'Distribution of Number of Children');

# For Charges

sns.histplot(x = df.charges, bins = 30);

# To check which variable impacts the target variable most. We can check the Relation of different variables with the target variable charges to find that

# # Age vs Charges

sns.scatterplot(x = 'age', y = 'charges', data = df).set(title = "Age vs Charges Scatterplot")

# # Age vs Charges with respect to Smoker

sns.scatterplot(x = 'age', y = 'charges', hue = 'smoker', data = df).set(title = "Age vs Charges with respect to Smoker");

# # Age vs Charges with respect to Sex

sns.scatterplot(x = 'age', y = 'charges', hue = 'sex', data = df).set(title = "Age vs Charges with respect to Sex")

# # Age vs Charges with respect to Region

sns.scatterplot(x = 'age', y = 'charges', hue = 'region', data = df).set(title = "Age vs Charges with respect to Region");

# # Age vs Charges with respect to Number of Children

sns.scatterplot(x = 'age', y = 'charges', hue = 'children', data = df).set(title = "Age vs Charges with respect to number of children");

# # Bmi vs Charges

sns.scatterplot(x = 'bmi', y = 'charges', data = df).set(title = 'Scatter Plot for BMI vs Charges')

# # Bmi vs Charges with respect to Smoker

sns.scatterplot(x = 'bmi', y = 'charges', hue = 'smoker', data = df).set(title = 'BMI vs Charges with respect to Smoker');

# # Bmi vs Charges with respect to Sex

sns.scatterplot(x = 'bmi', y = 'charges', hue = 'sex', data = df).set(title = 'BMI vs Charges with respect to Sex');

# # Bmi vs Charges with respect to the Number of Children

sns.scatterplot(x = 'bmi', y = 'charges', hue = 'children', data = df).set(title = 'BMI vs Charges with respect to the number of Children');

# # Bmi vs Charges with respect to Region

sns.scatterplot(x = 'bmi', y = 'charges', hue = 'region', data = df).set(title = 'BMI vs Charges with respect to the Region');

# # Correlation

# To check how all the variables are correlated with each other

df.corr()

# We can see that age is highly correlated with the variable charges.

sns.heatmap(df.corr(), cmap='Reds', annot=True)

plt.title('Heat Map');

# # Encoding Categorical Variables using OneHotEncoder

# These are the following categorical columns

categorical\_cols

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False, handle\_unknown='ignore').fit(df[categorical\_cols])

# To list the columns which are encoded

encoded\_cols = list(encoder.get\_feature\_names(categorical\_cols))

encoded\_cols

df[encoded\_cols] = encoder.transform(df[categorical\_cols])

df

df.drop(columns=categorical\_cols,axis = 1, inplace = True)

df

# Let's remove duplicate values

df.drop\_duplicates(inplace = True)

df

# Feature Matrix

X = df.drop('charges',axis = 1)

X.shape

# Response/Target Vector

y = df.charges

y.shape

# Splitting the Data Set into test data and train data set

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,random\_state = 911, test\_size = 0.20)

X\_train.shape,y\_train.shape

X\_test.shape,y\_test.shape

X\_train,X\_val,y\_train,y\_val = train\_test\_split(X\_train, y\_train, random\_state = 42, test\_size = 0.25)

X\_val.shape,y\_val.shape

# # Linear Regression Model

# Importing linear regression model from sklearn library

# Fitting the model to the train data

from sklearn.linear\_model import LinearRegression

log\_reg = LinearRegression(n\_jobs = -1, normalize = False, fit\_intercept = True).fit(X\_train,y\_train)

# Prediction on training data

train\_pred = log\_reg.predict(X\_train)

train\_pred

# To check y-intercept of the model

validation\_pred = log\_reg.predict(X\_val)

validation\_pred

# Co-efficients of the feautures which are selected

log\_reg.coef\_

log\_reg.normalize

log\_reg.fit\_intercept

# # Calculating the Loss Function for this Linear Regression Model

from sklearn.metrics import mean\_squared\_error

# Let's check the rmse for the training data

rmse\_train = np.sqrt(mean\_squared\_error(y\_train,train\_pred))

rmse\_train

# Let's check the rmse for the Validation data

rmse\_val = np.sqrt(mean\_squared\_error(y\_val,validation\_pred))

rmse\_val

sns.countplot(x = [rmse\_train,rmse\_val]).set(title = 'RMSE For Training and Validation Data');

# From the above plot, we can see that we have significantly less root mean squared error

# # Hyper parameter Tuning for the linear Regression model

# Our Linear Regression Model

log\_reg = LinearRegression(n\_jobs = -1, normalize = False, fit\_intercept = True).fit(X\_train,y\_train)

# # Tuning the Hyperparameter 'Normalize'

# when normalize = True

log\_reg2 = LinearRegression(n\_jobs = -1, normalize = True, fit\_intercept = True).fit(X\_train,y\_train)

rmse\_train2 = np.sqrt(mean\_squared\_error(y\_train, log\_reg2.predict(X\_train)))

rmse\_val2 = np.sqrt(mean\_squared\_error(y\_val, log\_reg2.predict(X\_val)))

# when normalize = False (default)

log\_reg1 = LinearRegression(n\_jobs = -1, normalize = False, fit\_intercept = True).fit(X\_train,y\_train)

rmse\_train1 = np.sqrt(mean\_squared\_error(y\_train, log\_reg1.predict(X\_train)))

rmse\_val1 = np.sqrt(mean\_squared\_error(y\_val, log\_reg1.predict(X\_val)))

# Creating Dictionary

dictionary1 = {

'Models':['rmse\_train1','rmse\_train2','rmse\_val1','rmse\_val2'],

'Values':[rmse\_train1,rmse\_train2,rmse\_val1,rmse\_val2],

'Bool':[0,0,1,1]

}

tuned\_df = pd.DataFrame(dictionary1)

tuned\_df

sns.barplot(data = tuned\_df, x = 'Models',y = 'Values')

# We can see that, there is no difference even after tuning the hyperparameters

# # Tuning the Hyperparameter 'fit\_intercept'

# when fit\_intercept = True (default)

log\_reg1 = LinearRegression(n\_jobs = -1, normalize = False, fit\_intercept = True).fit(X\_train,y\_train)

rmse\_train1 = np.sqrt(mean\_squared\_error(y\_train, log\_reg1.predict(X\_train)))

rmse\_val1 = np.sqrt(mean\_squared\_error(y\_val, log\_reg1.predict(X\_val)))

# when fit\_intercept = False

log\_reg2 = LinearRegression(n\_jobs = -1, normalize = True, fit\_intercept = False).fit(X\_train,y\_train)

rmse\_train2 = np.sqrt(mean\_squared\_error(y\_train, log\_reg2.predict(X\_train)))

rmse\_val2 = np.sqrt(mean\_squared\_error(y\_val, log\_reg2.predict(X\_val)))

# creating dictionary for the above created rmse values

dictionary2 = {

'Models':['rmse\_train1','rmse\_train2','rmse\_val1','rmse\_val2'],

'Values':[rmse\_train1,rmse\_train2,rmse\_val1,rmse\_val2]

}

tuned\_df = pd.DataFrame(dictionary2)

tuned\_df

sns.barplot(data = tuned\_df, x = 'Models',y = 'Values');

# Well, we can see that the model is not affected even after tuning the hyperparameters normalize and fit\_intercept.So, let's use our default model for the prediction.

# # Predicting the Model on test data set

test\_pred = log\_reg.predict(X\_test)

print("RMSE on Testing Data = ", np.sqrt(mean\_squared\_error(y\_test,test\_pred)))

# We can see that, the root mean squared value for the testing data is 5530.610151026031

# # Decision Tree Model

# Initially importing Decision Tree Model from sklearn library

from sklearn.tree import DecisionTreeRegressor

decision\_tree\_model = DecisionTreeRegressor(random\_state = 911, max\_depth = None, max\_leaf\_nodes = None).fit(X\_train,y\_train)

train\_pred = decision\_tree\_model.predict(X\_train)

validation\_pred = decision\_tree\_model.predict(X\_val)

# Calculating RMSE for validation data

rmse\_val = np.sqrt(mean\_squared\_error(y\_val,validation\_pred))

rmse\_val

# Calculating RMSE for training data

rmse\_train = np.sqrt(mean\_squared\_error(y\_train, train\_pred))

rmse\_train

# We can say from the value of RMSE that the decision tree model is overfitted to the training data.

# So, we will tune the hyperparameters to reduce the overfitting

from sklearn.tree import plot\_tree, export\_text

plt.figure(figsize=(80,20))

plot\_tree(tree, feature\_names = X\_train.columns,filled = True, max\_depth =2);

tree.tree\_.max\_depth

tree\_text = export\_text(tree, max\_depth=10, feature\_names=list(X\_train.columns))

print(tree\_text[:1000])

# generating feature imoortances for the tree

tree.feature\_importances\_

# creating dictionary for the feauture importances

importances = {

'Features':X\_train.columns,

'Weights':tree.feature\_importances\_

}

importances = pd.DataFrame(importances).sort\_values('Weights', ascending = False)

importances

plt.title("Importances/Weights asserted by different features in the Descision Tree Regressor Model")

sns.barplot(x = 'Weights', y ='Features', data = importances);

# From the above barplot, wecan clearly that 'smoker\_no' is impacting the target variable charges the most compared to the other features

# # Tuning the Hyperparamters for Decison Tree Model

# # Tuning the Hyperparameter 'max\_depth'

max\_depth\_values = list(range(2,31,2))

training\_rmse = []

validation\_rmse = []

for i in max\_depth\_values:

tree = DecisionTreeRegressor(random\_state = 911, max\_depth = i).fit(X\_train,y\_train)

training\_rmse.append(np.sqrt(mean\_squared\_error(y\_train,tree.predict(X\_train))))

validation\_rmse.append(np.sqrt(mean\_squared\_error(y\_val,tree.predict(X\_val))))

# Creating Dictionary for the RMSE values on training and validation

errors\_df = pd.DataFrame({

'Max Depth':max\_depth\_values,

'Training RMSE':training\_rmse,

'Validation RMSE':validation\_rmse

})

errors\_df

# plotting in graph

plt.title('Training vs Validation RMSE')

plt.xlabel('Max Depth')

plt.ylabel('Root Mean Squared Error')

plt.plot(errors\_df['Max Depth'], errors\_df["Training RMSE"])

plt.plot(errors\_df['Max Depth'], errors\_df["Validation RMSE"])

plt.xticks(max\_depth\_values)

plt.legend(['Training RMSE','Validation RMSE']);

# plotting in graph

plt.title('Training RMSE')

sns.barplot(x = 'Max Depth', y = 'Training RMSE', data = errors\_df);

# We can see that when Max Depth increases the RMSE values are going down. So, the Model is overfitting on the Training Data.

plt.title('Validation RMSE')

sns.barplot(x = 'Max Depth', y = 'Validation RMSE', data = errors\_df);

# When max depth = 4, we can see that our model performs best on the validation set. It can also be seen that when the value of max depth increases, our model begins to overfit the training data. As a result, we'll fine-tune our model based on its efficiency in validation data.

# # Tuning the Hyperparameter 'max\_leaf\_nodes'

max\_leaf\_nodes\_values = [2 \*\* i for i in range(1,11)]

training\_rmse = []

validation\_rmse = []

for i in max\_leaf\_nodes\_values:

tree = DecisionTreeRegressor(random\_state = 42, max\_depth = 4, max\_leaf\_nodes = i).fit(X\_train,y\_train)

training\_rmse.append(np.sqrt(mean\_squared\_error(y\_train,tree.predict(X\_train))))

validation\_rmse.append(np.sqrt(mean\_squared\_error(y\_val,tree.predict(X\_val))))

# Creating Dictionary for the RMSE values on training and validation

errors\_df = pd.DataFrame({

'Max\_leaf\_nodes':max\_leaf\_nodes\_values,

'Training RMSE':training\_rmse,

'Validation RMSE':validation\_rmse

})

errors\_df

# plotting in graph

plt.title('Training RMSE')

sns.barplot(x = 'Max\_leaf\_nodes', y = 'Training RMSE', data = errors\_df);

plt.title('Validation RMSE')

sns.barplot(x = 'Max\_leaf\_nodes', y = 'Validation RMSE', data = errors\_df);

# From the above barplots, When max leaf nodes = 8, our model performs best on both the training and validation sets. As a result, we'll adjust our model accordingly.

# Model after tuning - Optimized Model

tree = DecisionTreeRegressor(max\_leaf\_nodes = 8 ,max\_depth = 4, random\_state = 42).fit(X\_train,y\_train)

# # Predicting the Model on Testing Data

test\_pred = tree.predict(X\_test)

print("RMSE on Testing Data = ", np.sqrt(mean\_squared\_error(y\_test,test\_pred)))

# We can see that, the root mean squared error value for the testing data is 4115.912049019215.

# # Conclusion

# We can clearly see that, the rmse value for Linear Regression Model is 5530.610151026031 and for Decision Tree Model is 4115.912049019215. RMSE is less for Decison Tree Model than Linear Regression Model. So, Decison Tree Algorithm performed better for this data set

**Reference Material:**

[**https://datagy.io/python-sklearn-linear-regression/**](https://datagy.io/python-sklearn-linear-regression/)

[**https://www.geeksforgeeks.org/splitting-data-for-machine-learning-models/**](https://www.geeksforgeeks.org/splitting-data-for-machine-learning-models/)

**https://datagy.io/sklearn-one-hot-encode/**