MEDICAL INSURANCE COST PREDICTION

Group Members:

Anuvab Chatterjee
Asansol Engineering College
10800120094

Rohan Pramanik
Asansol Engineering College
10800120100

Shinjon Mukherjee
Asansol Engineering College
10800120114

Manvi Bishnu
Asansol Engineering College
10800120058

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, Dr. Anindya Banerjee for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark. I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Anuvab Chatterjee

Rohan Pramanik

Shinjon Mukherjee

Manvi Bishnu

Contents

ABSTRACT

India's government spends 1.5 percent of its annual GDP on public healthcare, which is significantly less than that of other countries. Global public health spending, on the other hand, has almost doubled in line with inflation in the last two decades, reaching US$ 8.5 trillion in 2019, or 9.8% of global GDP. Multinational multi-private sectors provide around 60% of comprehensive medical treatments and 70% of out-patient care, which charge patients astronomically high fees. Because of the rising expense of quality healthcare, increased life expectancy, and the epidemiological shift toward non-communicable diseases, health insurance is becoming an essential commodity for everyone. Insurance data has increased dramatically in the last decade, and carriers now have access to it. The health insurance system explores predictive modeling to boost its business operations and services. Computer algorithms and Machine Learning (ML) is used to study and analyze the past insurance data and predict new output values based on trends in customer behavior, insurance policies, and data-driven business decisions, and support in formulating new schemes. Additionally, ML has found enormous and potential applications in the insurance industry.

Thus, this report develops a real-time insurance cost price prediction system named ML Health Insurance Prediction System using ML algorithms which will aid the insurance companies in the market for easy and rapid determination of values of premiums and thereby curb down health expenditure. The model incorporates and demonstrates different models of regression such as Linear Regression, Decision Tree Regression, Random Forest Regression, and Gradient Boost Regression to anticipate insurance costs and assess model outcomes. In the proposed model, the Gradient Boost Regression model has achieved better results with an RMSE value of 4449.357579 and R -squared value of 0.8681803006293177 compared to all the other models.

Keywords: Machine Learning, Data Collection, Data Analysis, Data Pre-Processing, Linear Regression Model, Decision Tree Regression, Random Forest Regression and Gradient Boost Regression.

INTRODUCTION

We live on a planet full of threats and uncertainty. Including People, households, durables, properties are exposed to different risks and the risk levels can vary. These risks range from risk of health diseases to death if not get protection, and loss in property or assets[1]. But, risks cannot usually be avoided, so the world of finance has developed numerous products to shield individuals and organizations from these risks by using financial capital to shield them. Therefore Insurance is one of the policies that either decreases or removes loss costs incurred by various risks. The value of insurance in the lives of individuals. That's why it becomes important for insurance companies to be sufficiently precise to measure the amount covered by this specific policy and the insurance charges which must be paid for it. Various parameters or factors play an important role in estimating the insurance charges and Each of these is important. If any factor is omitted or changed when the amounts are computed then, the overall policy cost changes. It is therefore very critical to carry out these tasks with high accuracy. So, the possibility of human mistakes are high so insurance agents also use different tools to calculate the insurance premium. And thus ML is beneficial here. ML may generalize the effort or method to formulate the policy. These ML models can be learned by themselves. The model is trained on insurance data from the past. The model can then accurately predict insurance policy costs by using the necessary elements to measure the payments as its inputs. This decreases human effort and resources and improves the company's profitability. Thus the accuracy can be improved with ML. Our goal is to predict insurance costs. The value of insurance fees is based on different variables. As a result, insurance fees are continuous. Regression is the best choice available to fulfill our needs. We use multiple linear regression in this analysis since there are many independent variables used to calculate the dependent(target) variable. For this study, the dataset for cost of health insurance is used. Pre-processing of the dataset done first. Then we trained regression models with training data and finally evaluated these models based on testing data. In this article, we used several models of regression, for example, Linear Regression, Decision Tree Regression and Gradient Boosting Regression. It is found that the gradient boosting provides the highest accuracy with an r-squared value of 0.8681803006293177. The inclusion of a novel method of insurance cost estimation is the main goal of this work.

LITERATURE SURVEY

Several studies on estimating medical prices have been published in the health field in different contexts. Machine learning has many probable assumptions, but its performance relies on picking a nearly precise algorithm for the specified problem domain and following the appropriate procedures to build, train, and deploy the model. The data used in the model is explicitly collected from Kaggle from a survey of more than 1300 patients with relevant attributes. The ages range from young to old i.e. anywhere from 10-70 age range, although most of the patients are in their 20's. There's about an equal ratio of men to women with slightly more men participating in the survey. We have then included different factors such as having children, smoking, BMI, region where they come from and ultimately a model was made to calculate and predict the charges for the unseen number of patients.

PROJECT OBJECTIVE

To make a machine learning system to predict the cost of medical insurance of a person using some machine learning algorithms. The model needs to learn from the existing data to make accurate predictions in terms of cost provided some given attributes.

First, we need to collect some data, which, here, in this case, we have gotten from Kaggle. The model learns from some parameters, eg: previous health issues, age, smoking habits, gender etc. Next step involves data analysis which includes plotting of graphs and charts to derive the meaning of the given data. This step helps us in inferring the data and its relationships. The data needs to be pre-processed next in order to be fed into the machine learning model. It then gets split into training data and test data and the training data is fed into the model. The model is then trained using different algorithms, here, we have used Linear Regression, Random Forest Regression, Decision Tree Regression and Gradient Boost Regression to find out the best accuracy. And, finally, the trained model gets fed the new data to predict the insurance cost of an individual.

POJECT SCOPE

Project Scope:- Medical Insurance Cost Prediction using Machine Learning with Python

1) The system will be available on an online platform accessible to all people.

2) The system will collect insurance data and check the information to give the prediction.

DATA DESCRIPTION

We had used a dataset from Kaggle Site for creating our prediction model. This dataset has 7 features and this data set has splitted into two-parts : training data and testing data. For training the model of this dataset, 80% of total data is used for training and the rest 20% is used for testing. To build a predictor model of medical insurance cost the training dataset is applied and to evaluate the regression model, test set is used.

Columns (continuous type)

- age: age of primary beneficiary

STATISTICAL MEASURES

count:- 1338.000000

max:- 64.000000

min:- 18.000000

std:- 14.049960

mean:- 39.207025

- bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height,
  objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9

STATISTICAL MEASURES

count:- 1338.000000

max:- 53.130000

min:- 15.960000

std:- 6.098187

mean:- 30.663397

- children: Number of children covered by health insurance / Number of dependents

STATISTICAL MEASURES

count:- 1338.000000

max:- 5.000000

min:- 0.000000

std:- 1.205493

mean:- 1.094918

- charges: Individual medical costs billed by health insurance.

STATISTICAL MEASURES

count:- 1338.000000

max:- 63770.428010

min:- 1121.873900

std:- 12110.011237

mean:- 13270.422265

Columns (categorical type)

- sex: insurance contractor gender, female, male
- smoker: Smoking
- region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.

GRAPHS USED:

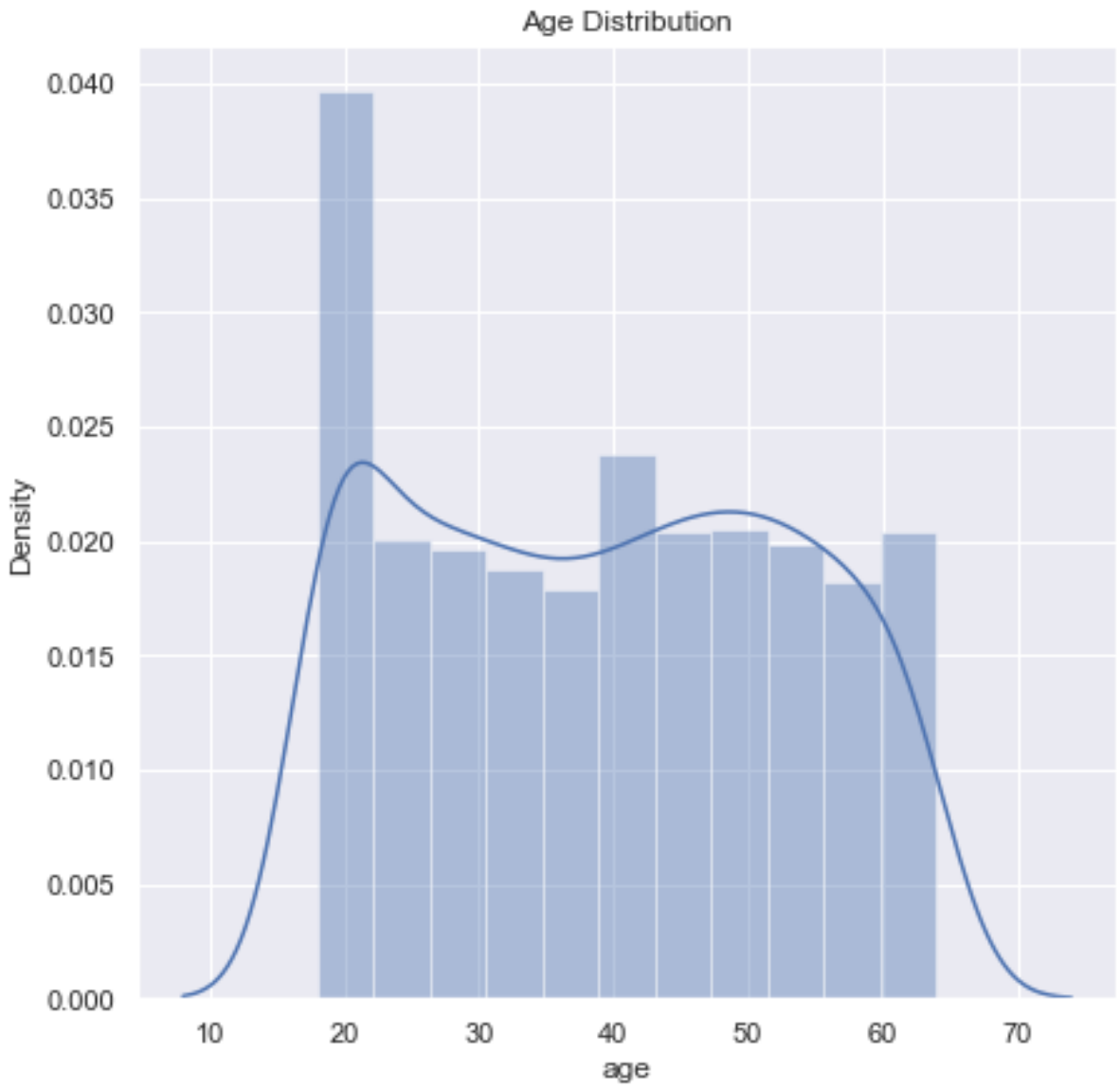
Graphs and charts are effective visual tools because they present information quickly and easily. It is not surprising then, that graphs are commonly used by print and electronic media. Sometimes, data can be better understood when presented by a graph than by a table because the graph can reveal a **trend or comparison**. They come under the Data Visualisation and Analysis part of making a machine learning model.

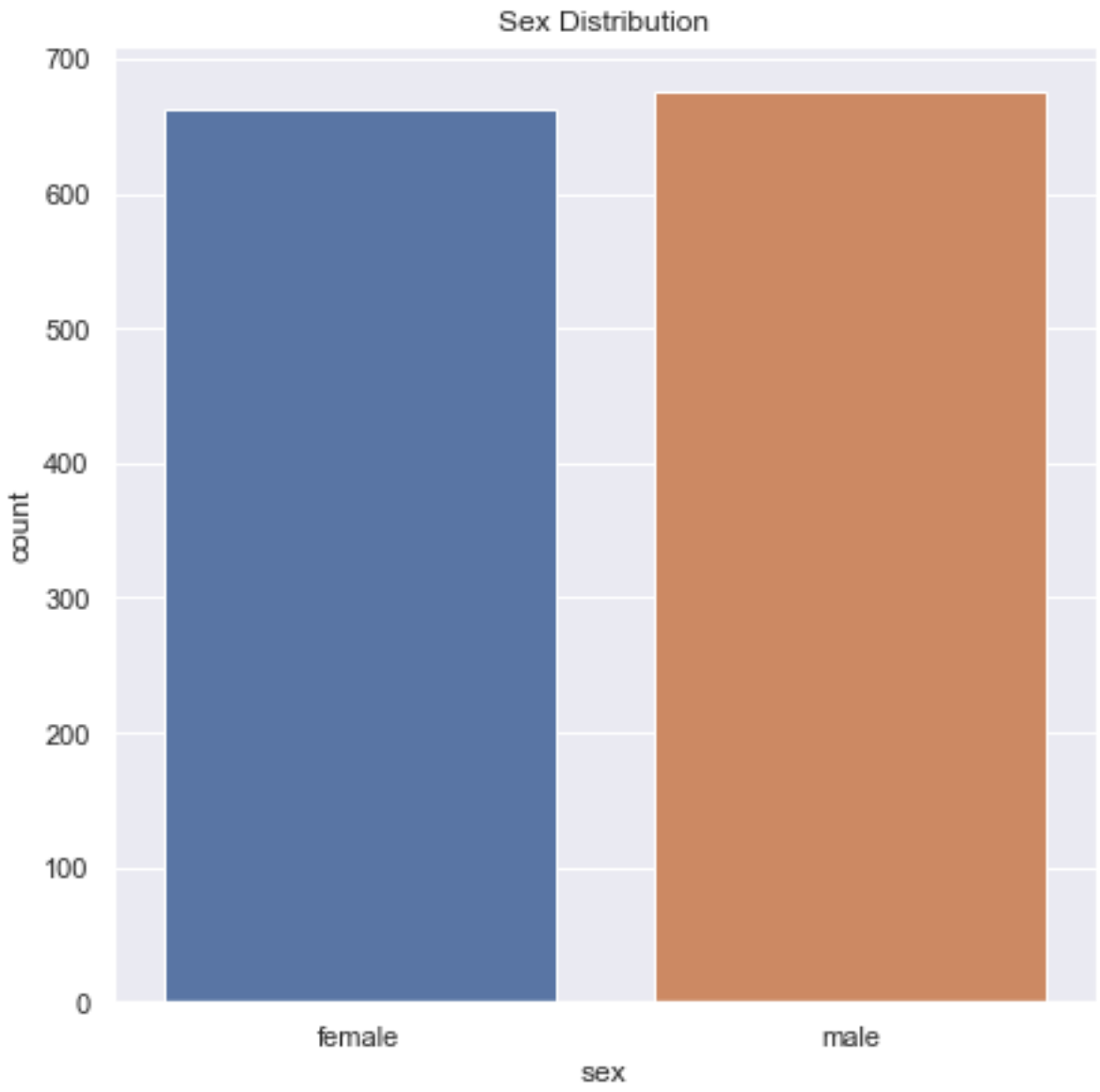
There are many different types of graphs that can be used to convey information, including:

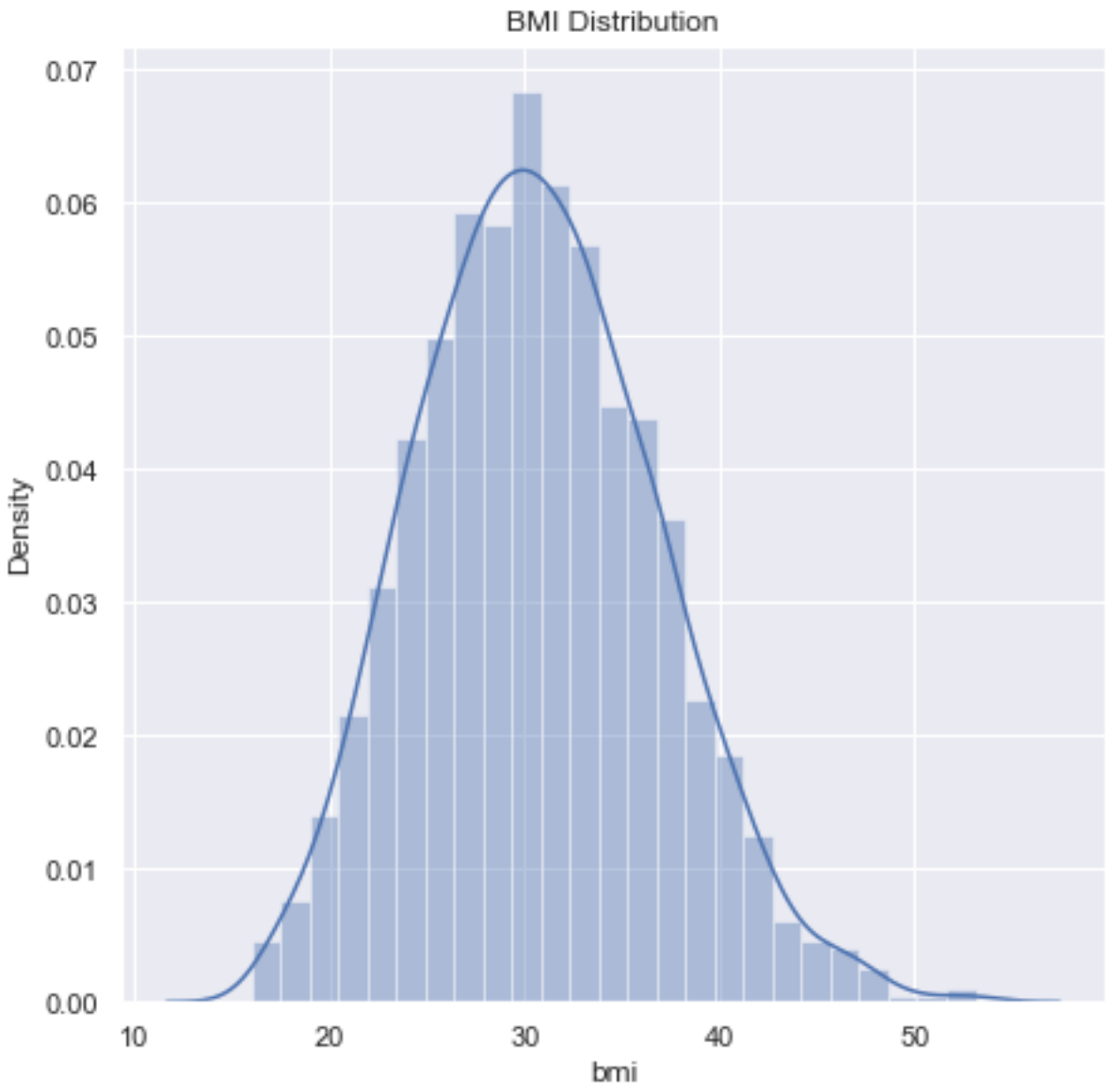
- bar charts
- pictographs
- pie charts
- line charts
- scatterplots
- histograms

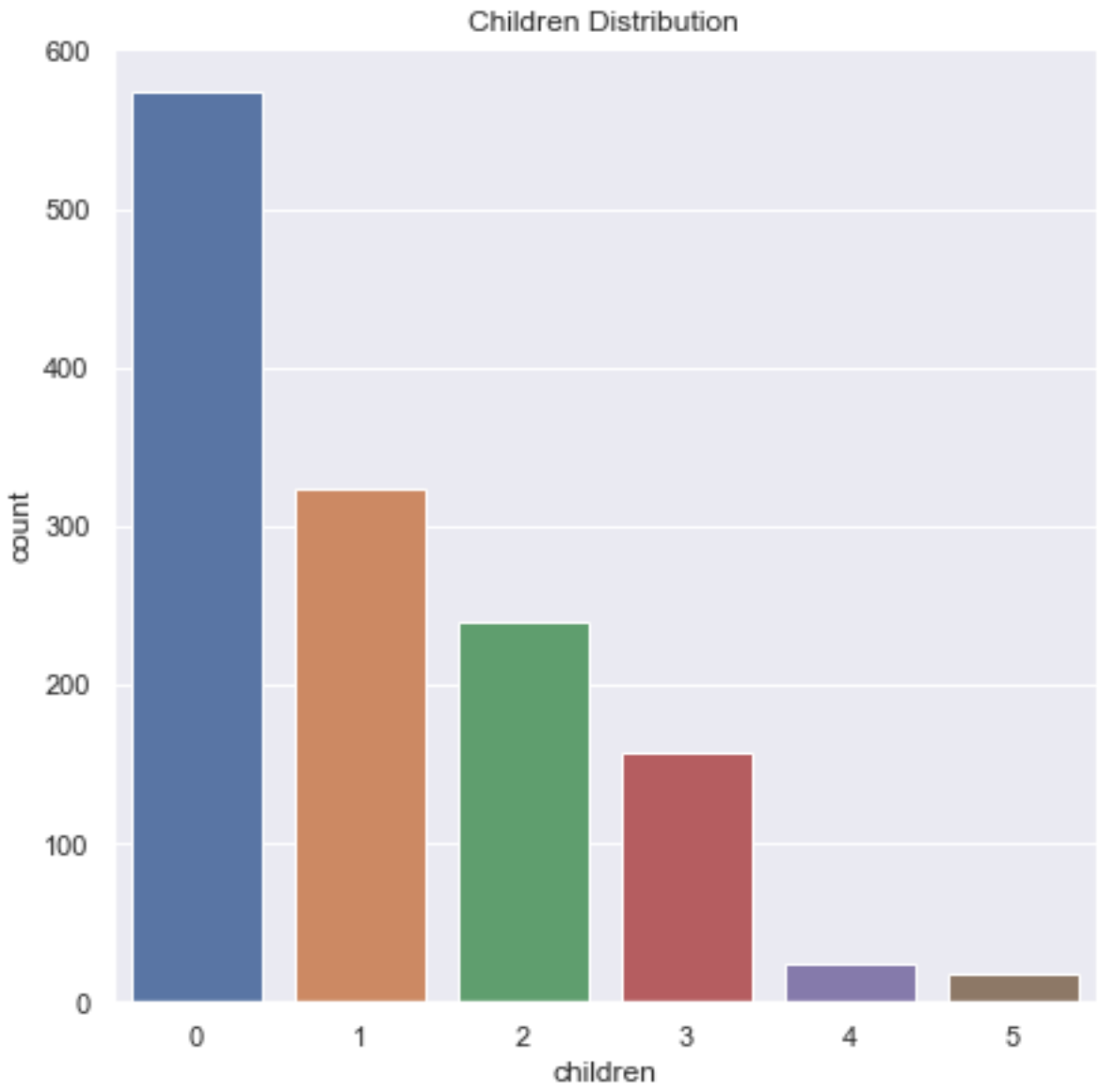
Knowing what type of graph or chart to use with what type of information is crucial. Depending on the nature of the variables, some types are more appropriate than others. For example, categorical variables are best displayed in a bar chart or pie chart while continuous variables are illustrated by a line chart or histogram.

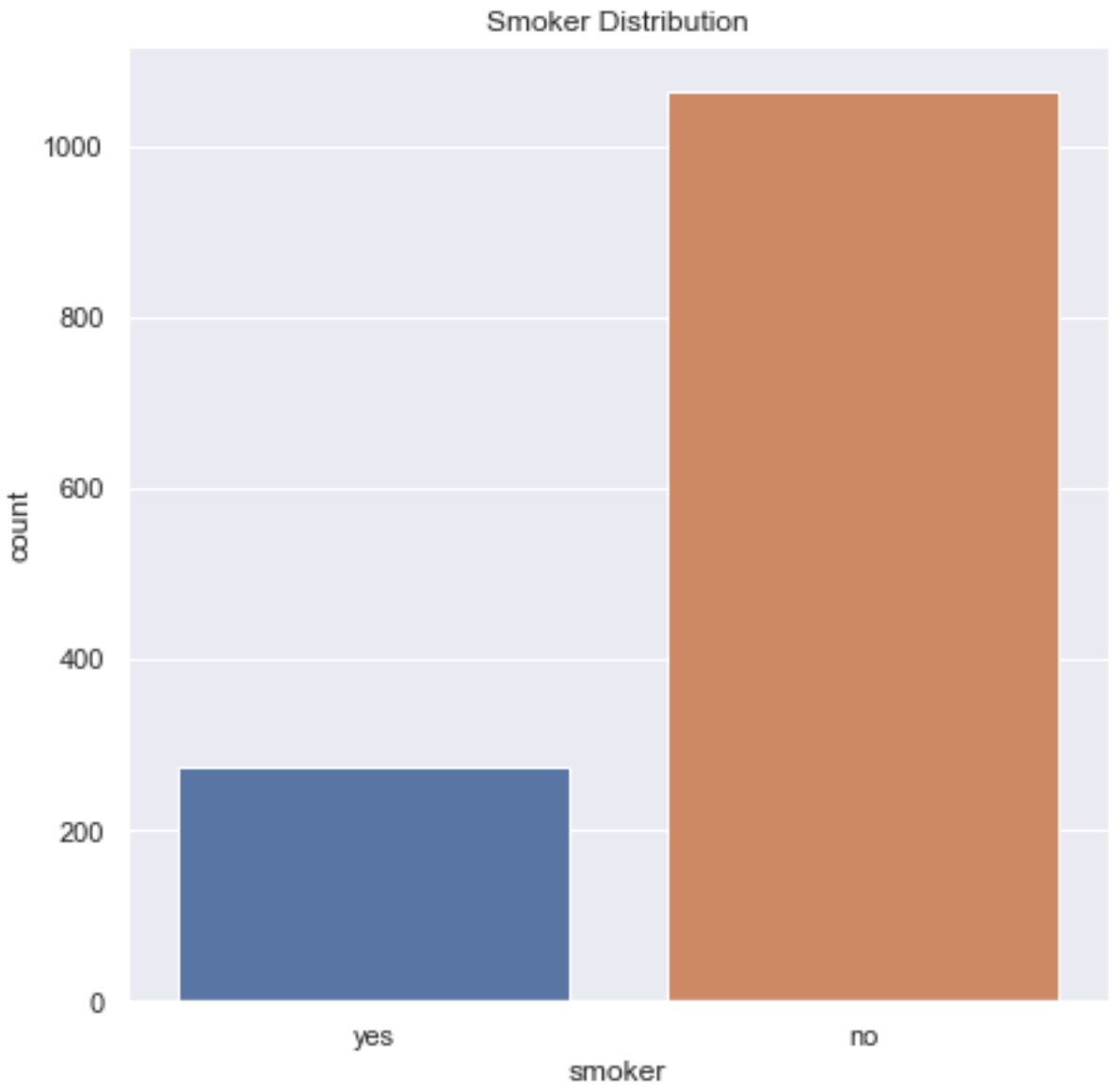
Here, in this project, we have made use of bargraphs, probability distribution plots/kde plots and scatterplots.

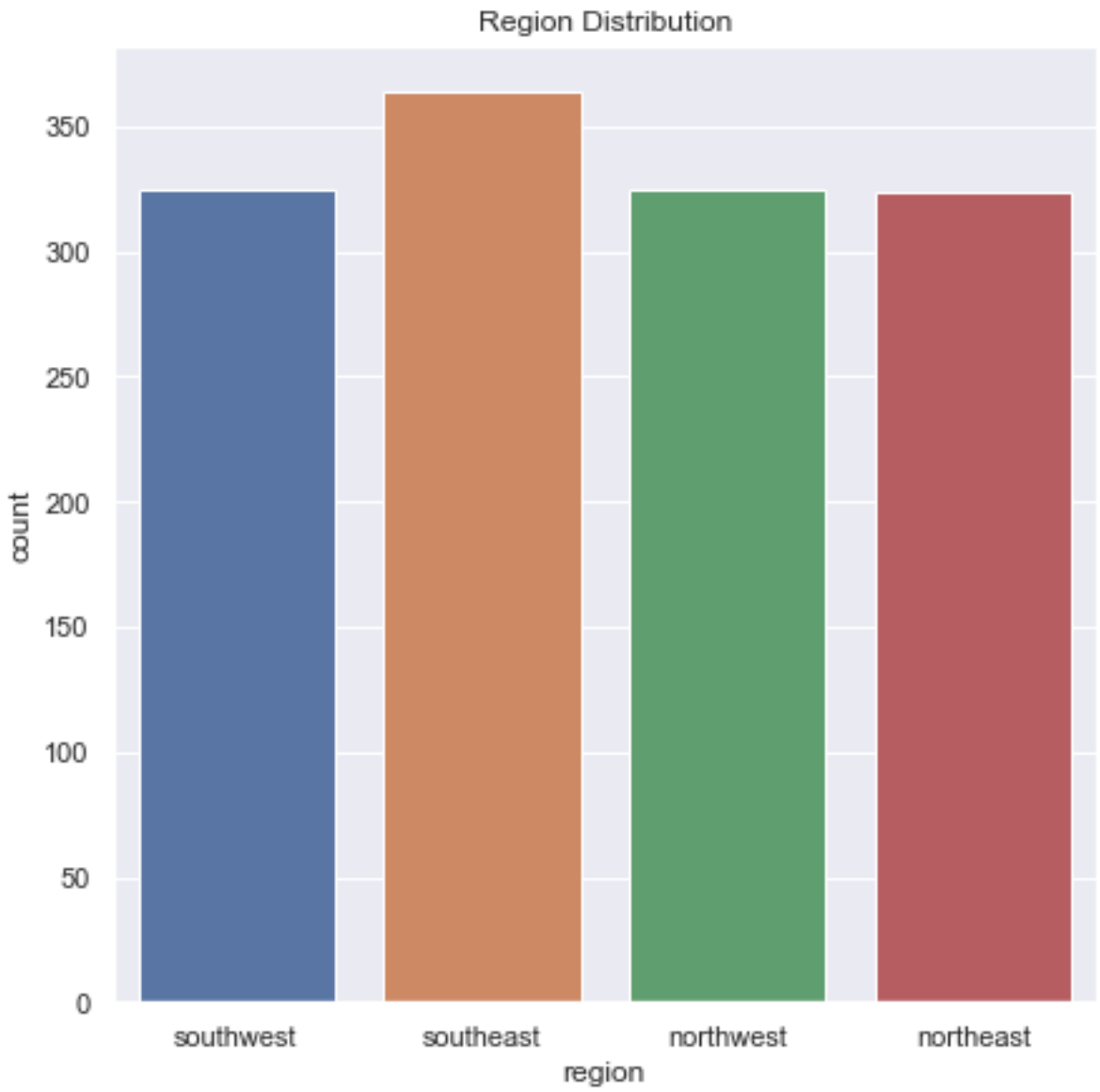

Age Distribution
Density
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
10
20
30
40
50
60
70
age

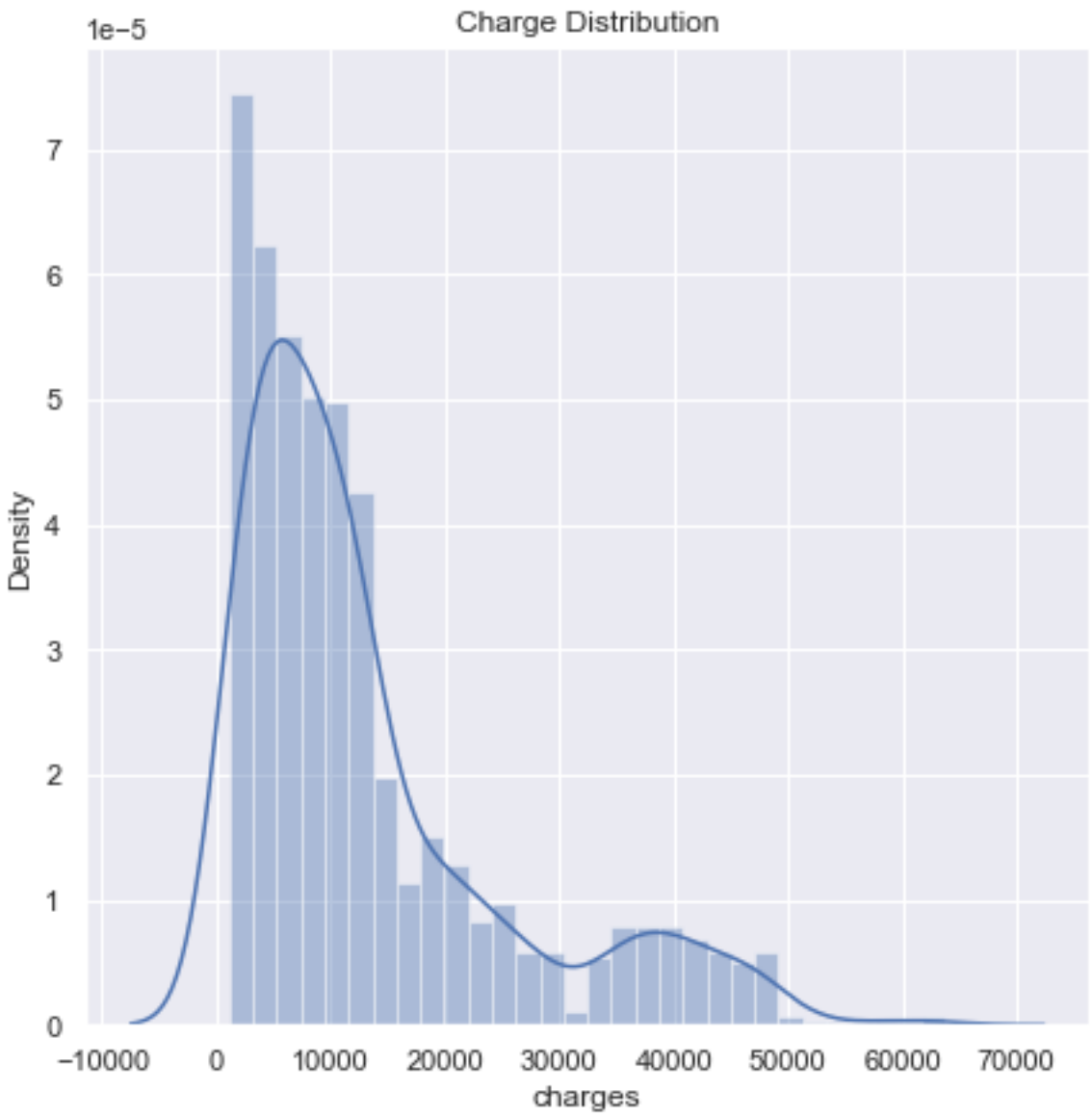

Sex Distribution
700
600
500
400
300
200
100
0
count
female
male
sex

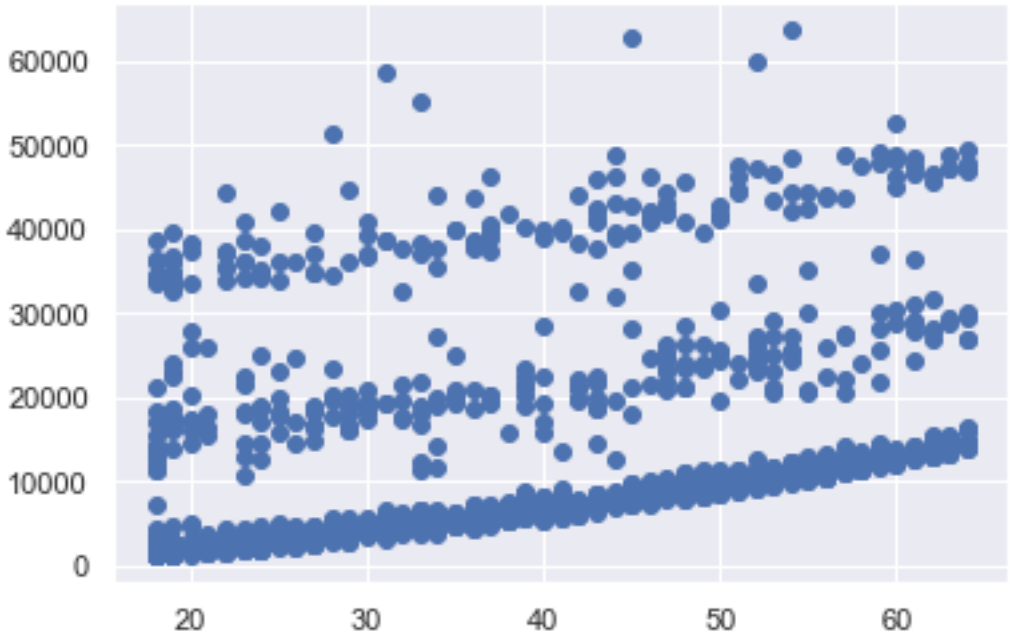

BMI Distribution
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
10
20
30
40
50
bmi

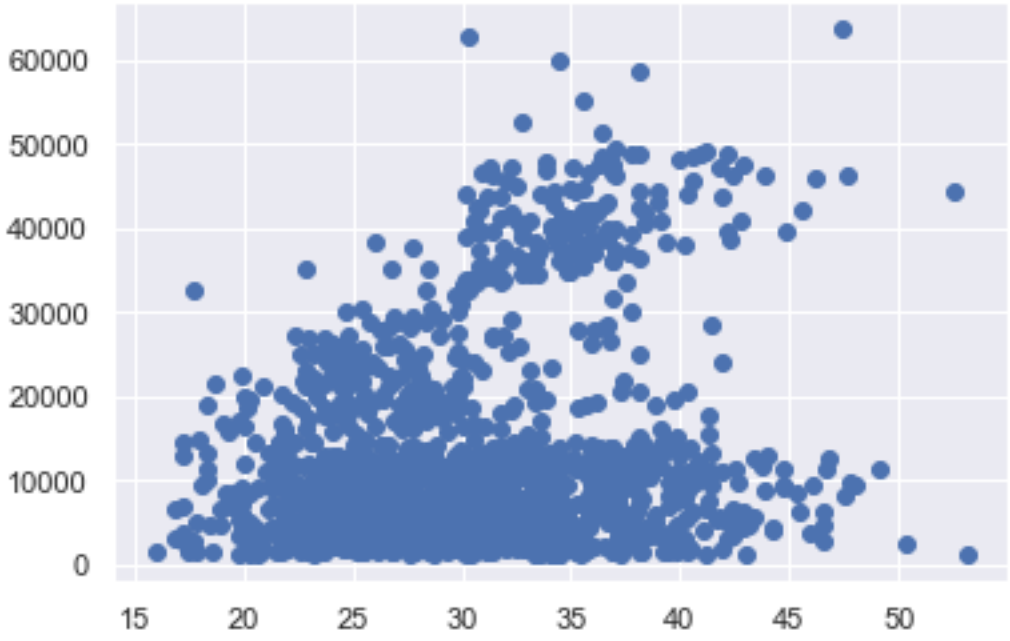

Children Distribution
600
500
400
300
200
100
0
count
0
1
2
3
4
5
children

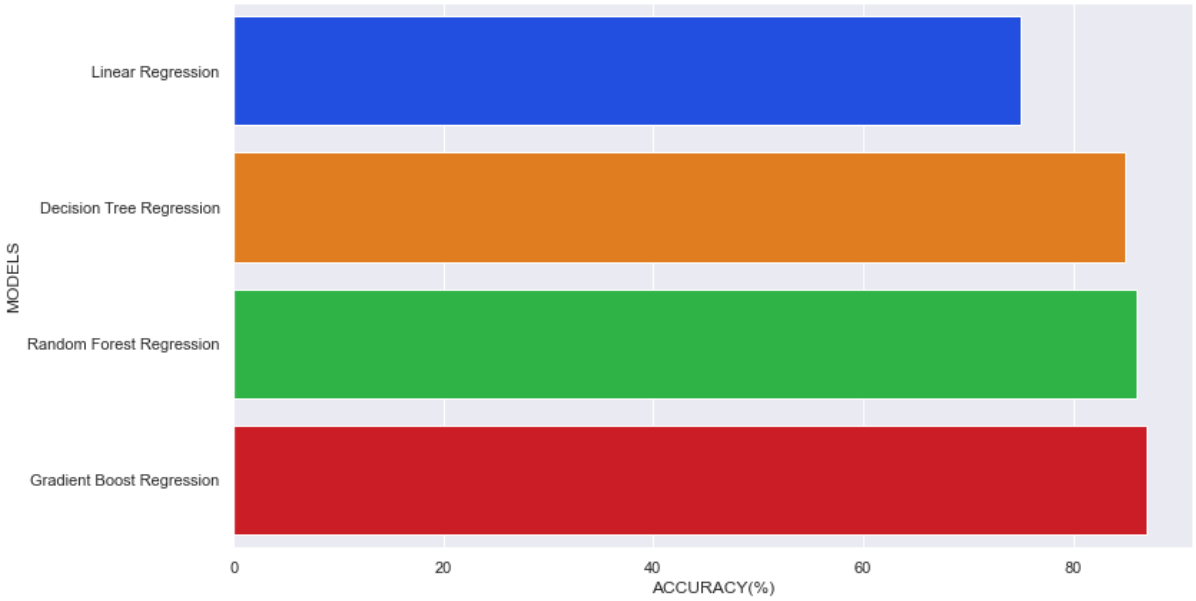

Smoker Distribution
1000
800
600
400
200
0
count
yes
no
smoker


Region Distribution
350
300
250
200
150
100
50
0
count
southwest
southeast
northwest
northeast
region


Charge Distribution
1e−5
7
6
5
4
3
2
1
0
Density
−10000
0
10000
20000
30000
40000
50000
60000
70000
charges


60000
50000
40000
30000
20000
10000
0
20
30
40
50
60


60000
50000
40000
30000
20000
10000
0
15
20
25
30
35
40
45
50


Linear Regression
Decision Tree Regression
Random Forest Regression
Gradient Boost Regression
MODELS
0
20
40
60
80
ACCURACY(%)

IMPORTED DEPENDENCIES

NUMPY

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

We are importing NumPy here as reference variable **np**.

PANDAS

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

We import pandas as **pd**.

Here, pd is referred to as an alias to the Pandas. However, it is not necessary to import the library using the alias, it just helps in writing less amount code every time a method or property is called.

Pandas generally provide two data structures for manipulating data, They are:

- **Series**
- **DataFrame**

Series: Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

DataFrame: Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

MATPLOTLIB.PYPLOT

Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits like Tkinter, awxPython, etc.

Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are **Line Plot**, **Histogram**, **Scatter**, **3D Plot**, **Image**, **Contour**, and **Polar**.

We import matplotlib.pyplot as **plt.**

SEABORN

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.
Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

lots are basically used for visualizing the relationship between variables. Those variables can be either be completely numerical or a category like a group, class or division. Seaborn divides plot into the below categories –

- **Relational plots:** This plot is used to understand the relation between two variables.
- **Categorical plots:** This plot deals with categorical variables and how they can be visualized.
- **Distribution plots:** This plot is used for examining univariate and bivariate distributions
- **Regression plots:** The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses.
- **Matrix plots:** A matrix plot is an array of scatterplots.
- **Multi-plot grids:** It is an useful approach is to draw multiple instances of the same plot on different subsets of the dataset.

We are importing seaborn here as **sns.**

SKLEARN

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon **NumPy, SciPy** and **Matplotlib**.

Model_selection is a method for setting a **blueprint** to analyze data and then using it to measure new data. Selecting a proper model allows us to generate **accurate results** when making a prediction.

To do that, you need to **train your model** by using a specific dataset. Then, we test the model against another dataset.

The **train_test_split** function of the **sklearn.model_selection** package in Python splits arrays or matrices into random subsets for train and test data, respectively.

LinearRegression fits a linear model with coefficients w=(w1,…,wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

The sklearn. metrics module **implements several loss, score, and utility functions to measure classification performance**. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

R2 (coefficient of determination) is regression score function.

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). In the general case when the true y is non-constant, a constant

model that always predicts the average y disregarding the input features would get a R2 score of 0.0.

In the particular case when y_true is constant, the R2 score is not finite: it is either NaN (perfect predictions) or -Inf (imperfect predictions). To prevent such non-finite numbers to pollute higher-level experiments such as a grid search cross-validation, by default these cases are replaced with 1.0 (perfect predictions) or 0.0 (imperfect predictions) respectively. We can set force_finite to False to prevent this fix from happening.

Note: when the prediction residuals have zero mean, the R2 score is identical to the **Explained Variance score**.

MODEL BUILDING

A. Linear Regression

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand. The independent variables can be called exogenous variables, predictor variables, or regressors.

Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings, or to predict the future value of a currency based on its past performance.

One of the most important supervised learning tanks is regression. In regression set of records are present with X and Y values and this values are used to learn a function, so that if you want to predict Y from an unknown X this learn function can be used. In regression we have to find value of Y, So, a function is required which predicts Y given XY is continuous in case of regression.

Here Y is called as criterion variable and X is called as predictor variable. There are many types of functions or modules which can be used for regression. Linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

Model Evaluation:-

RMSE :- 6182.955535

R2_Score(training):- 0.751820

R2_Score(test):- 0.745447

Accuracy(%):- 75.0%

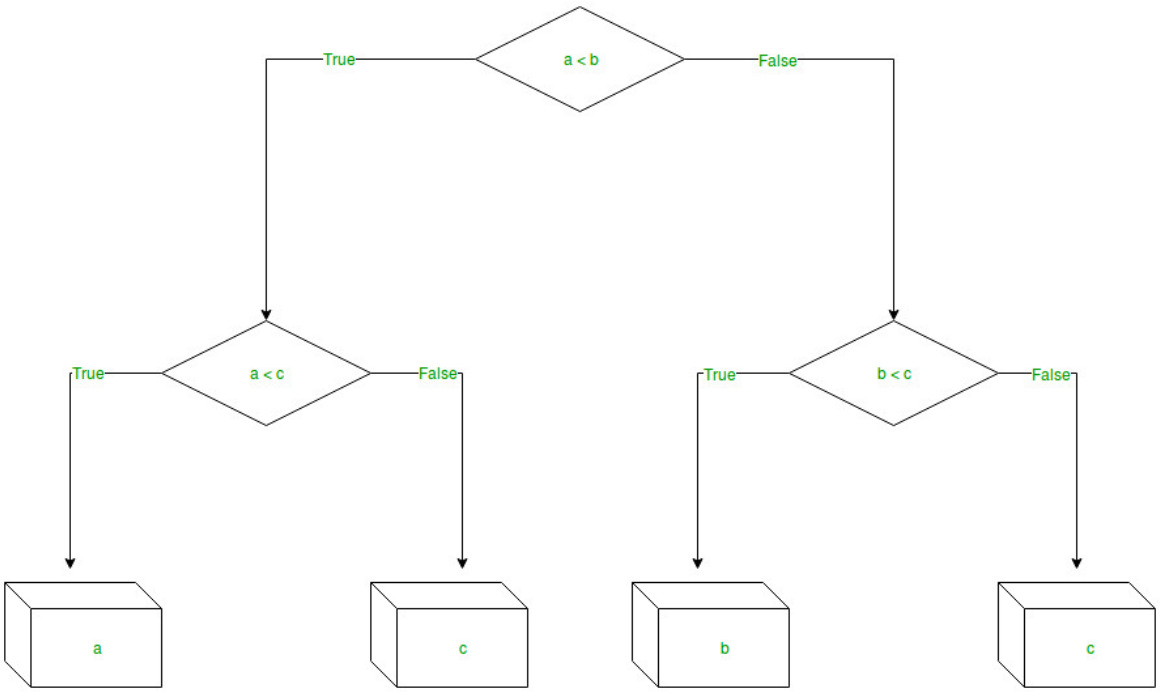
B. Decision trees Regression

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs, and utility.
Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

1. Conditions [Decision Nodes]
2. Result [End Nodes]

The branches/edges represent the truth/falsity of the statement and take makes a decision based on that in the example below which shows a decision tree that evaluates the smallest of three numbers:



Decision Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Model Evaluation:-

RMSE :- 4731.645479

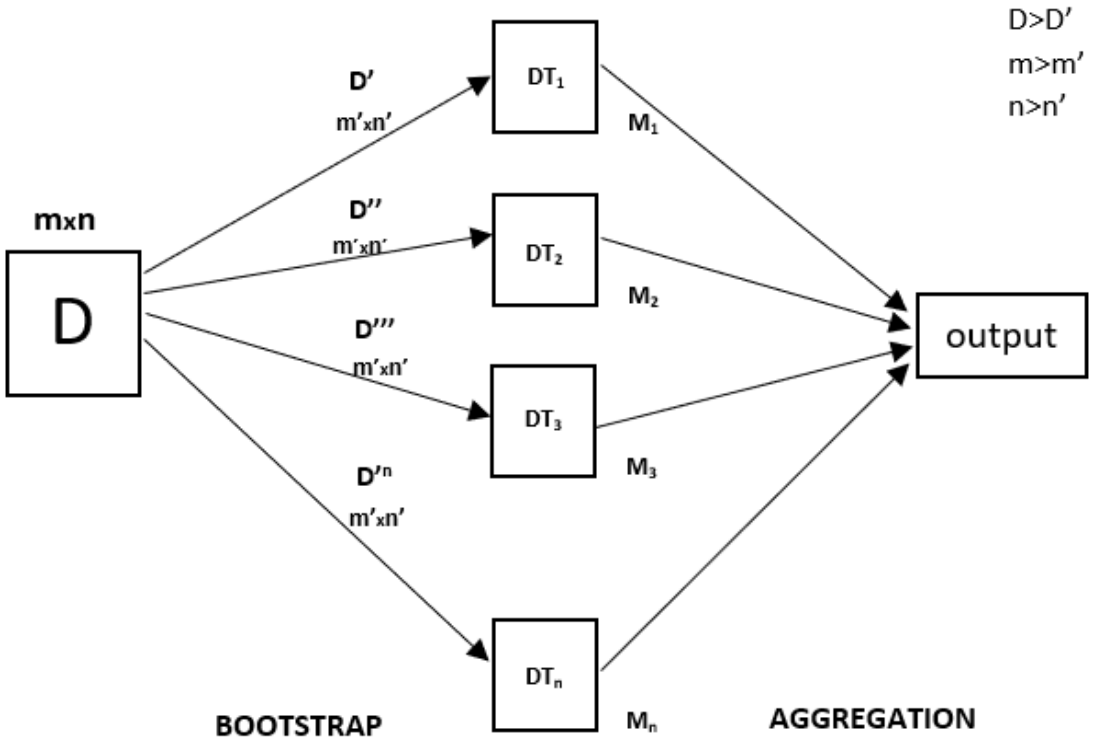
R2_Score(training):- 0.858204

R2_Score(test):- 0.850923

Accuracy(%):- 85.0%

B. Random Forest Regression

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data, and hence the output doesn't depend on one decision tree but on multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is called Aggregation.



Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

We need to approach the Random Forest regression technique like any other machine learning technique.

- Design a specific question or data and get the source to determine the required data.
- Make sure the data is in an accessible format else convert it to the required format.
- Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
- Create a machine learning model
- Set the baseline model that you want to achieve
- Train the data machine learning model.
- Provide an insight into the model with test data
- Now compare the performance metrics of both the test data and the predicted data from the model.
- If it doesn't satisfy your expectations, you can try improving your model accordingly or dating your data, or using another data modeling technique.
- At this stage, you interpret the data you have gained and report accordingly.

Model Evaluation:-

RMSE :- 4606.606669

R2_Score(training):- 0.863005

R2_Score(test):- 0.858698

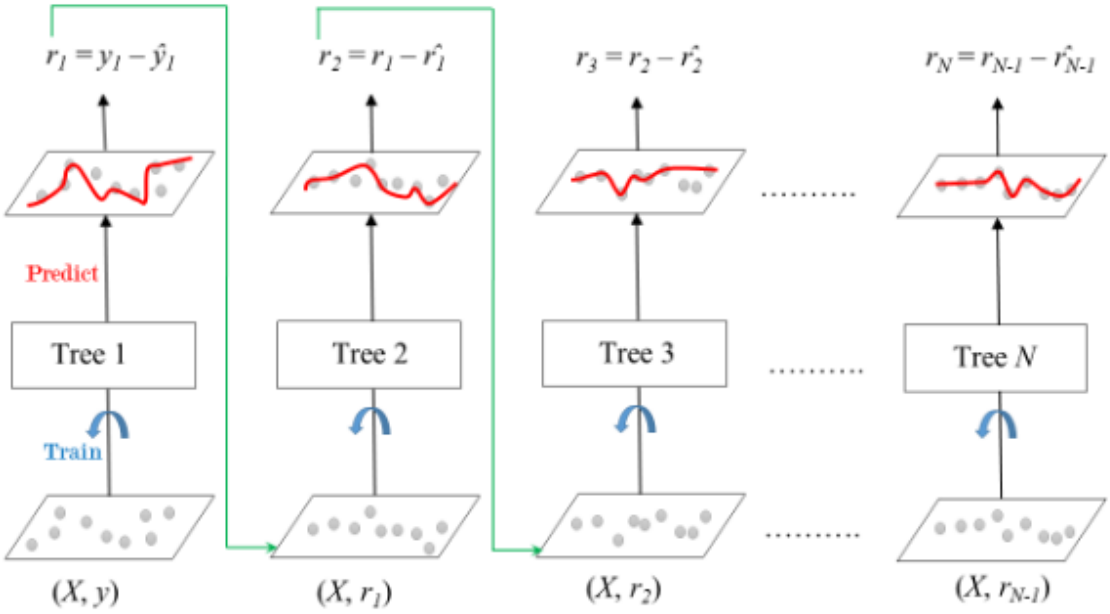
Accuracy(%):- 86.0%

C. Gradient Boosting Regression

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

The below diagram explains how gradient boosted trees are trained for regression problems.



The ensemble consists of *N* trees. Tree1 is trained using the feature matrix *X* and the labels *y*. The predictions labelled *y1(hat)* are used to determine the training set residual errors *r1*. Tree2 is then trained using the feature matrix *X* and the residual errors *r1* of Tree1 as labels. The predicted results *r1(hat)* are then used to determine the residual *r2*. The process is repeated until all the *N* trees forming the ensemble are trained.

There is an important parameter used in this technique known as **Shrinkage.**

Shrinkage refers to the fact that the prediction of each tree in the ensemble is shrunk after it is multiplied by the learning rate (eta) which ranges between 0 to 1. There is a trade-off between eta and number of estimators, decreasing learning rate needs to be compensated with increasing estimators in order to reach certain model performance. Since all trees are trained now, predictions can be made.

Each tree predicts a label and final prediction is given by the formula,

y(pred) = y1 + (eta * r1) + (eta * r2) + ... + (eta * rN)

The class of the gradient boosting regression in scikit-learn is **GradientBoostingRegressor.** A similar algorithm is used for classification known as **GradientBoostingClassifier.**

Model Evaluation:-

RMSE :- 4449.357579

R2_Score(training):- 0.905217

R2_Score(test):- 0.868180

Accuracy(%):- 87.0%

CODE

IMPORTING THE DEPENDENCIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import r2_score
```

DATA COLLECTION AND ANALYSIS

```
# Loading the data from csv file to a Pandas DataFram
e
insurance_dataset = pd.read_csv('C:/Users/Lenovo/OneD
rive/Desktop/AEC ML/insurance.csv')
# First 5 rows of DataFrame
insurance_dataset.head()
```

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

```
# Number of rows and columns
insurance_dataset.shape
```

```
(1338, 7)
```

```
# Getting Some Information about the dataset
insurance_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
```

```
memory usage: 73.3+ KB
```

```
# Checking for missing values in each column
insurance_dataset.isnull().sum()
```

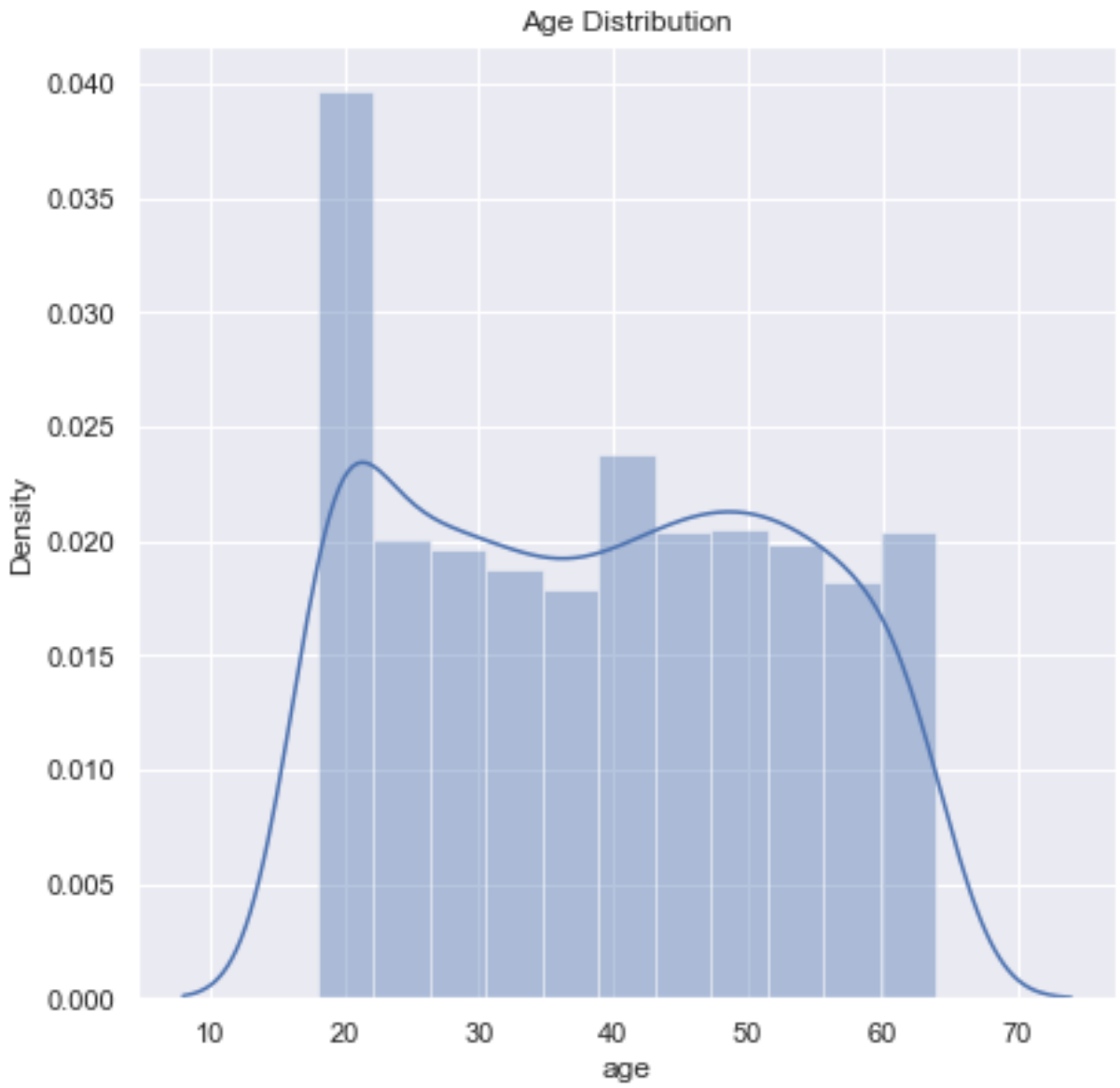
```
age       0
sex       0
bmi       0
children  0
smoker    0
region    0
charges   0
dtype: int64
```

DATA ANALYSIS

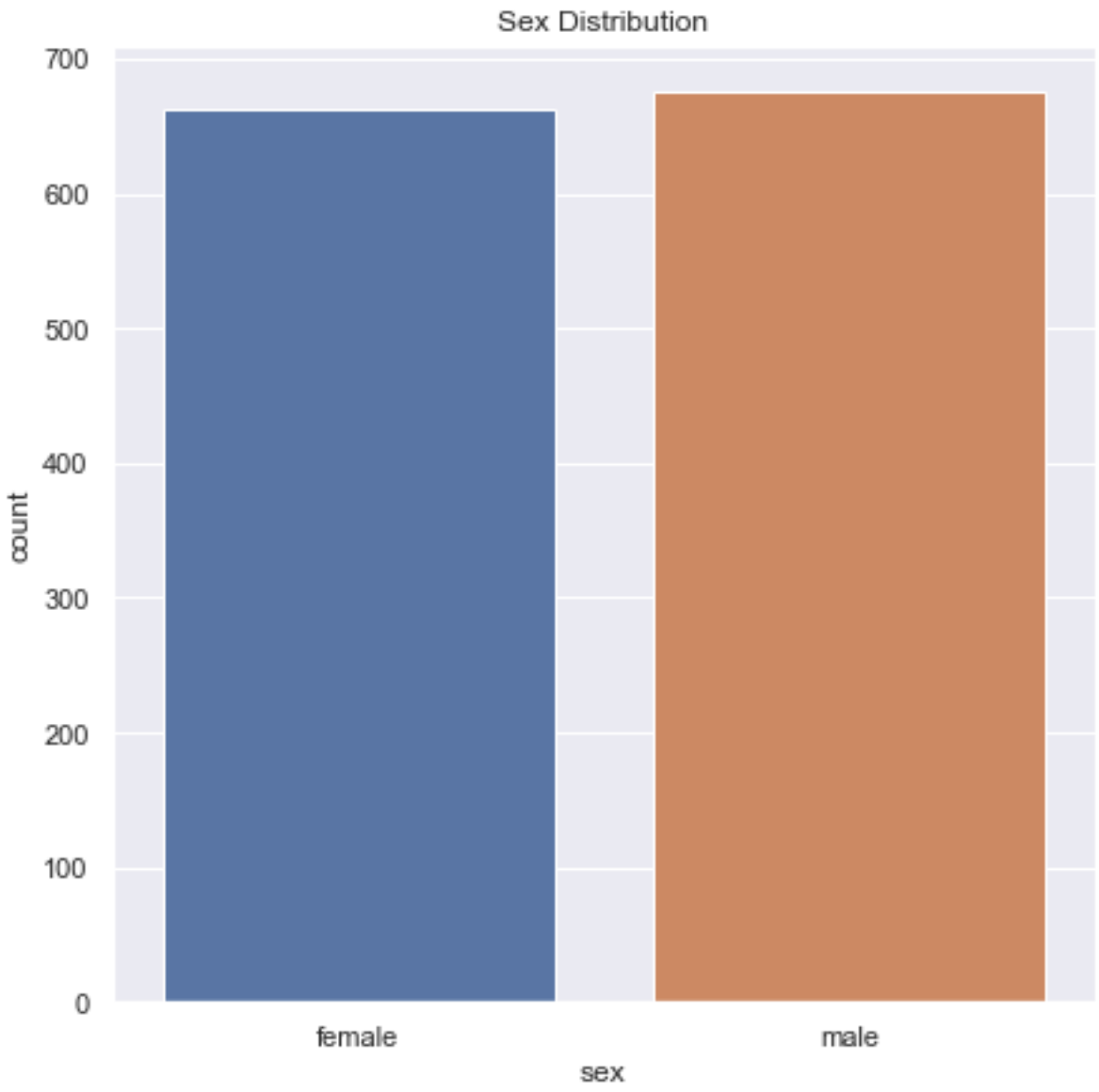
```
# Statistical measures of the dataset
insurance_dataset.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

```
# Distribution of 'age' value
sns.set()
plt.figure(figsize=(7,7))
sns.distplot(insurance_dataset['age'])
plt.title('Age Distribution')
plt.show()
```




```
# Distribution of 'sex' column
plt.figure(figsize=(7,7))
sns.countplot(x='sex',data=insurance_dataset)
plt.title("Sex Distribution")
plt.show()
```




```
# Number of Males and Females
insurance_dataset['sex'].value_counts()
```

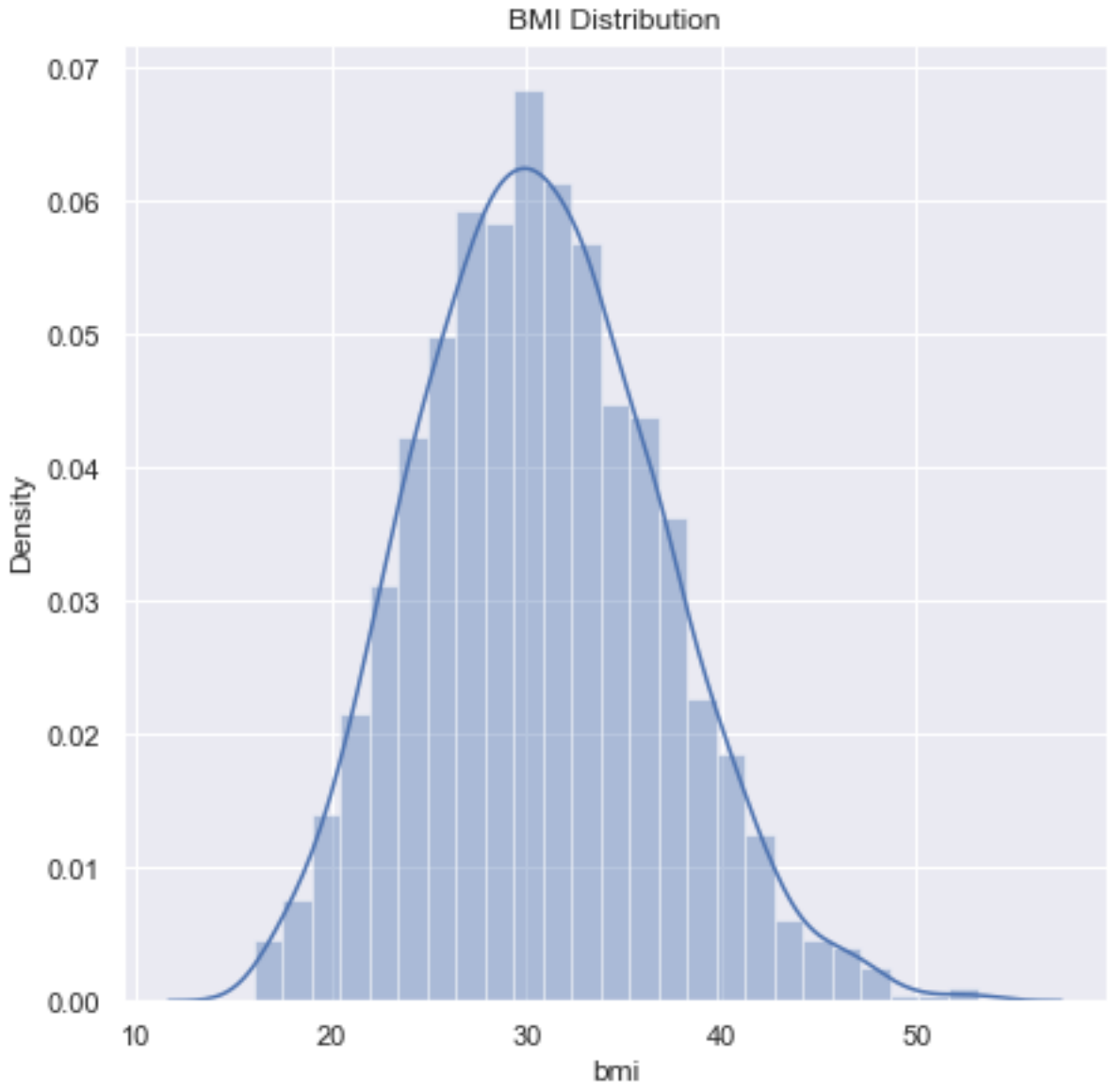
male 676

female 662

Name: sex, dtype: int64

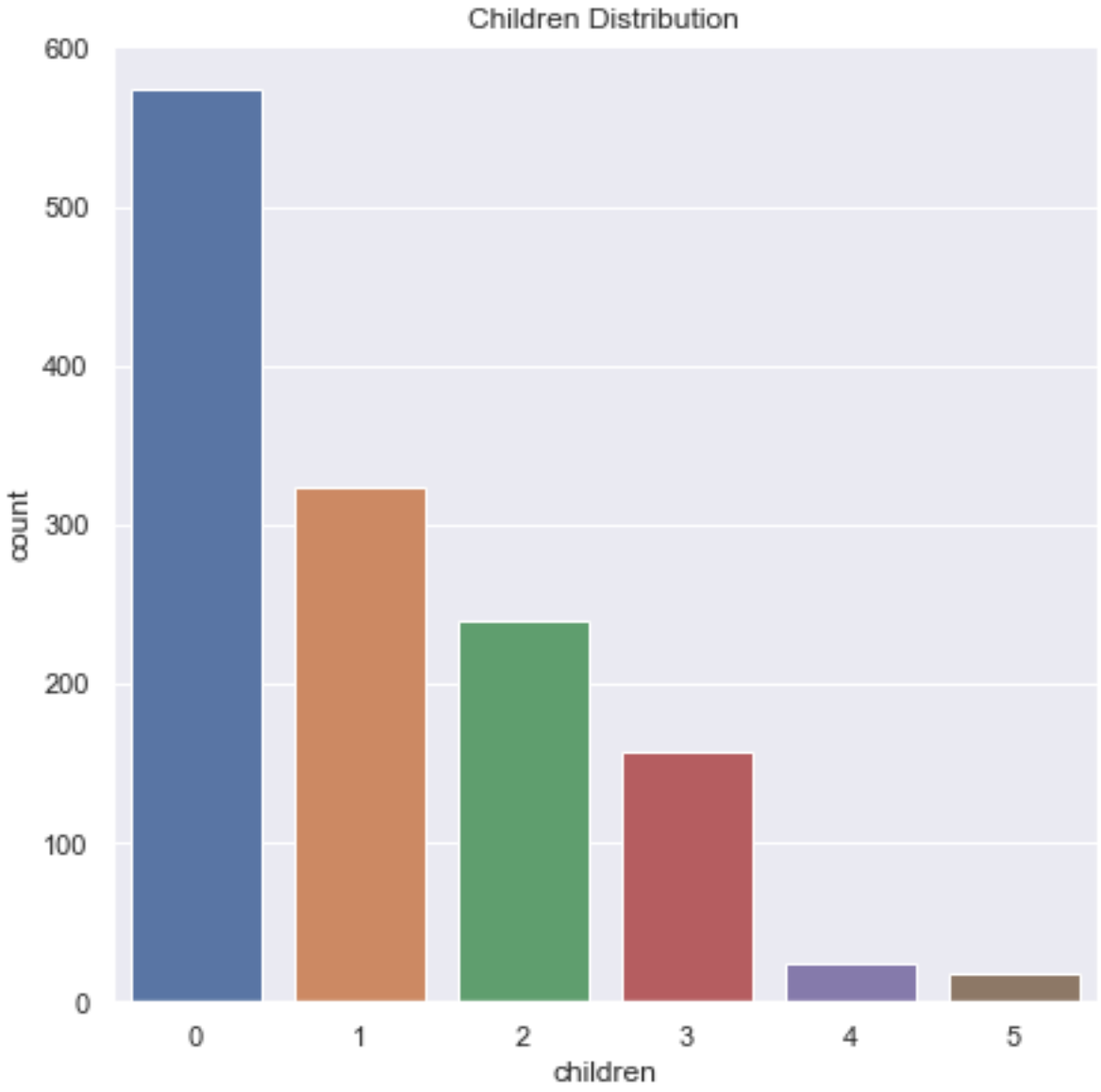
```
# Distribution of 'bmi' value
sns.set()
plt.figure(figsize=(7,7))
sns.distplot(insurance_dataset['bmi'])
plt.title('BMI Distribution')
plt.show()
```

```
C:\Users\Lenovo\anaconda3\lib\site-
packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function
and will be removed in a future version. Please adapt
your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Normal BMI Range ----> 18.5 to 24.9

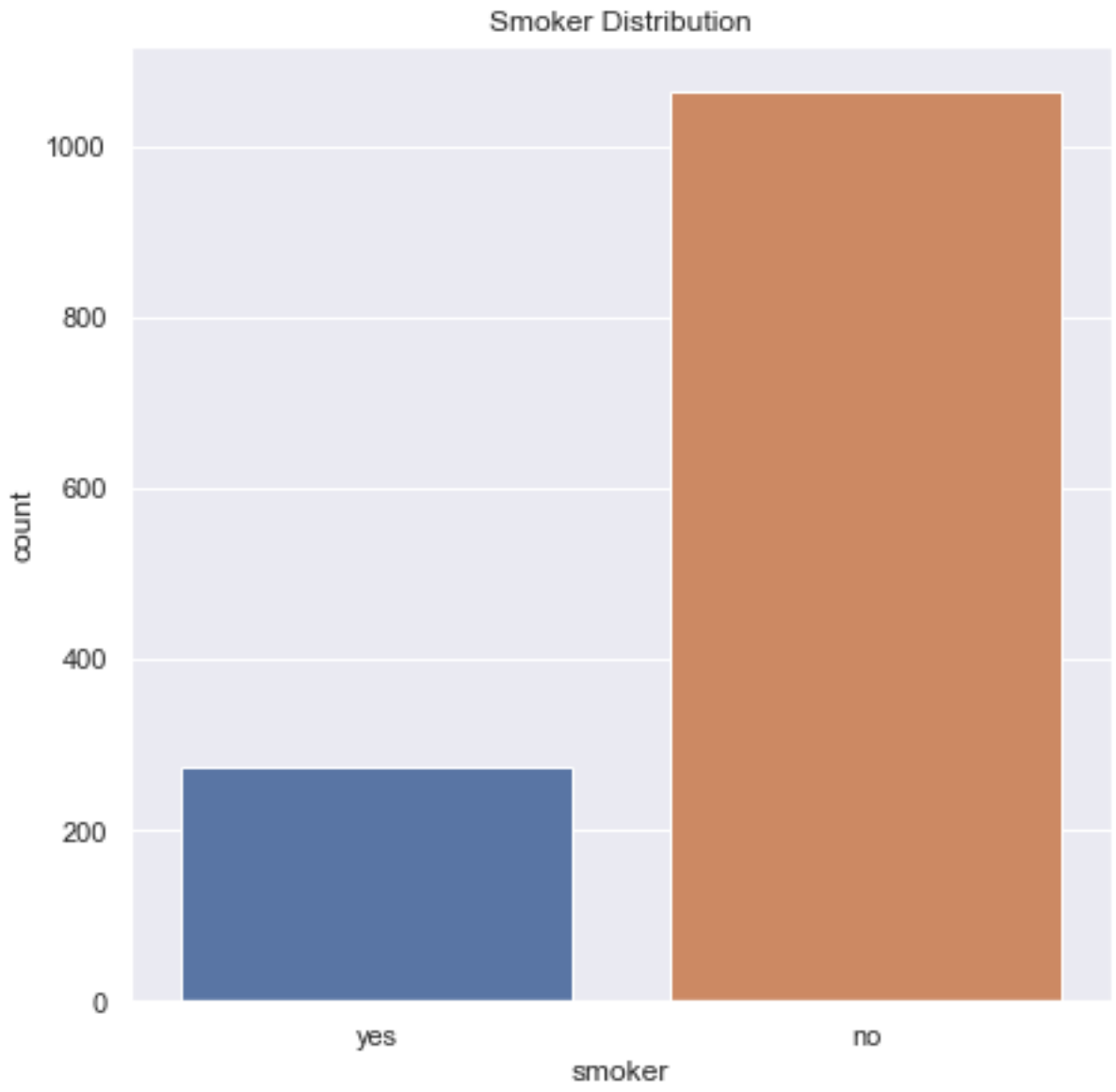
```
# Distribution of 'children' column
plt.figure(figsize=(7,7))
sns.countplot(x='children',data=insurance_dataset)
plt.title("Children Distribution")
plt.show()
```



```
# Number of childrens
insurance_dataset['children'].value_counts()
```

```
0   574
1   324
2   240
3   157
4    25
5    18
Name: children, dtype: int64
```

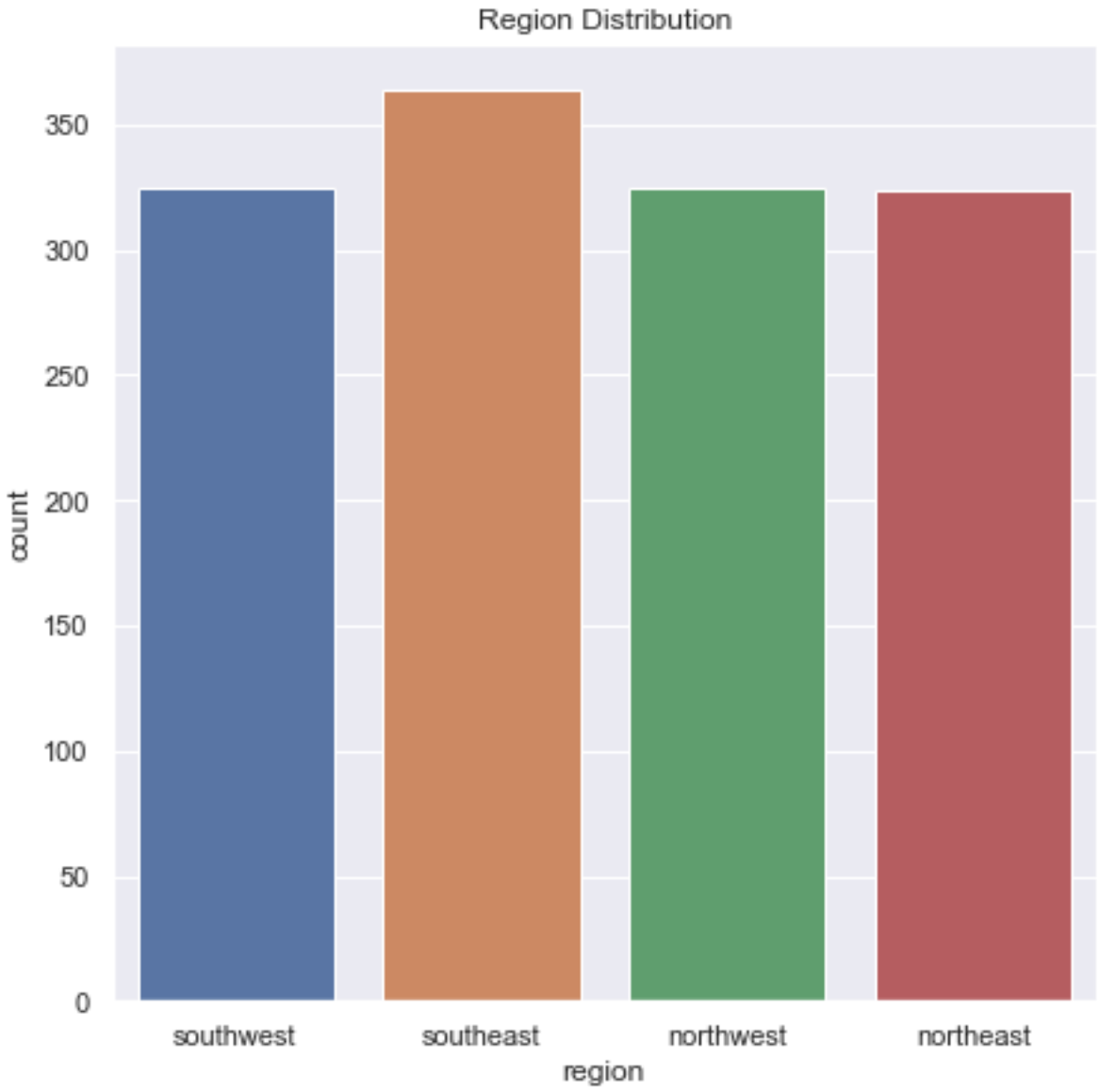
```
# Distribution of 'smoker' column
plt.figure(figsize=(7,7))
sns.countplot(x='smoker',data=insurance_dataset)
plt.title("Smoker Distribution")
plt.show()
```




```
# Number of smokers
insurance_dataset['smoker'].value_counts()
```

```
no    1064
yes    274
Name: smoker, dtype: int64
```

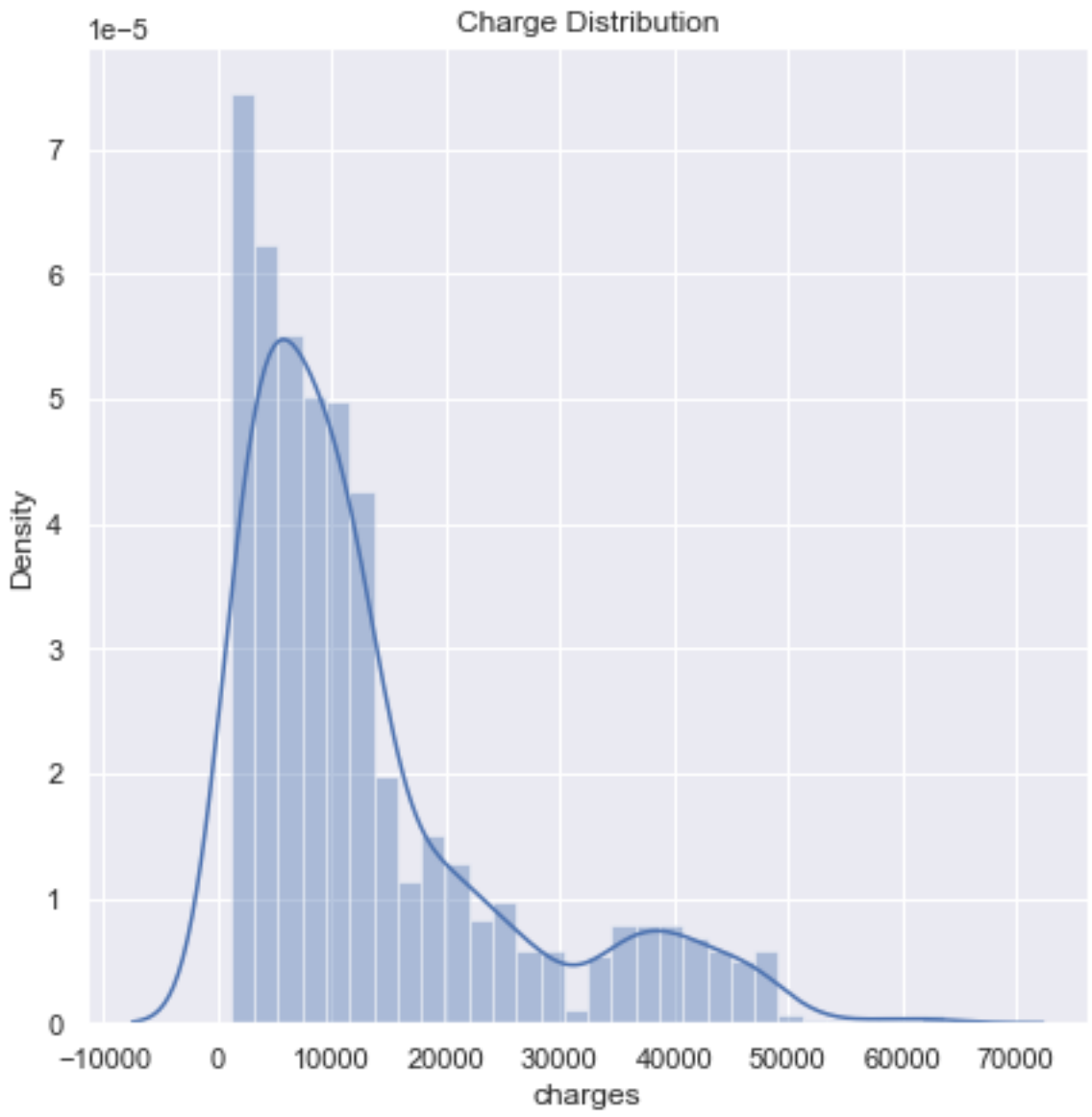
```
# Distribution of 'region' column
plt.figure(figsize=(7,7))
sns.countplot(x='region',data=insurance_dataset)
plt.title("Region Distribution")
plt.show()
```



```
# Number of people in each region
insurance_dataset['region'].value_counts()
```

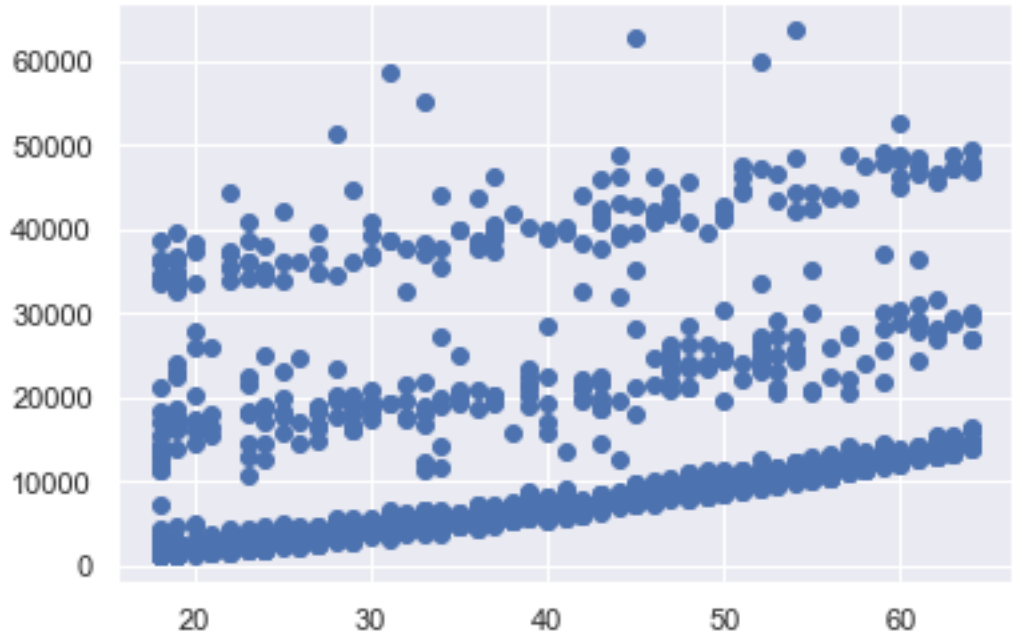
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

```
# Distribution of 'charges' value
sns.set()
plt.figure(figsize=(7,7))
sns.distplot(insurance_dataset['charges'])
plt.title('Charge Distribution')
plt.show()
```


Charge Distribution
1e−5
Density
0
1
2
3
4
5
6
7
−10000
0
10000
20000
30000
40000
50000
60000
70000
charges

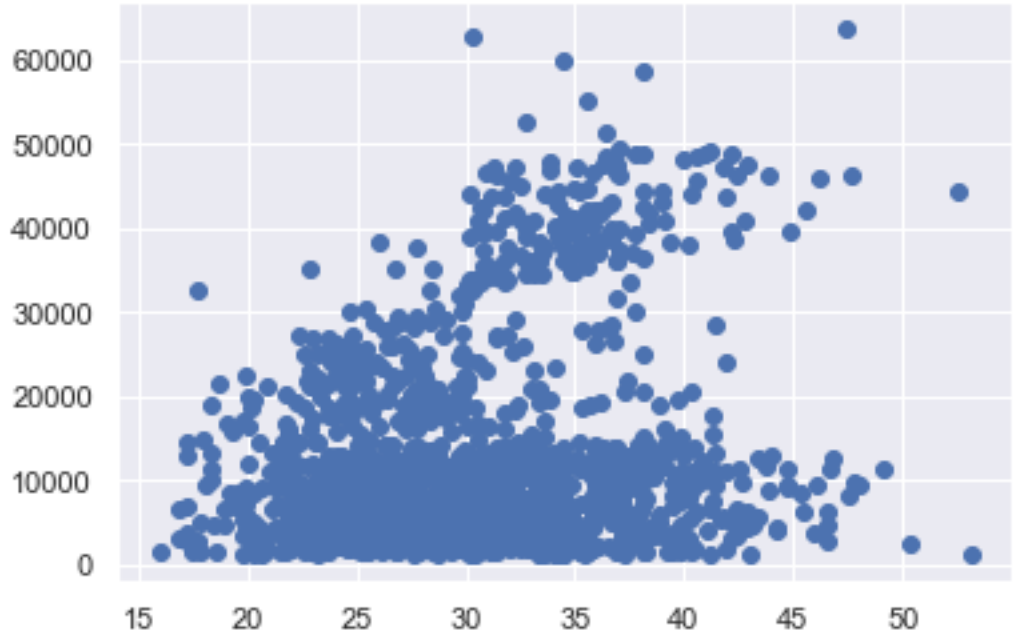
```
#scatter plot between Age and Charges
plt.scatter(insurance_dataset.age,insurance_dataset.charges)
```

```
<matplotlib.collections.PathCollection at 0x2ad37a99250>
```



```
#scatter plot between BMI and Charges
plt.scatter(insurance_dataset.bmi,insurance_dataset.charges)
```

```
<matplotlib.collections.PathCollection at 0x2ad37abf910>
```



DATA PRE-PROCESSING

Encoding Categorical Features

```
# Encoding 'sex' column
insurance_dataset.replace({'sex':{'male':0 , 'female'
:1}},inplace=True)

# Encoding 'smoker' column
insurance_dataset.replace({'smoker':{'yes':0 , 'no':1
}},inplace=True)

# Encoding 'region' column
insurance_dataset.replace({'region':{'southeast':0 ,
'southwest':1 , 'northwest':2 , 'northeast':3}},inpla
ce=True)
```

	age	sex	bmi	children	smoker	region	charges
0	19	1	27.900	0	0	1	16884.92400
1	18	0	33.770	1	1	0	1725.55230
2	28	0	33.000	3	1	0	4449.46200
3	33	0	22.705	0	1	2	21984.47061
4	32	0	28.880	0	1	2	3866.85520
...	...	...	...	...	...	...	...
1333	50	0	30.970	3	1	2	10600.54830
1334	18	1	31.920	0	1	3	2205.98080
1335	18	1	36.850	0	1	0	1629.83350
1336	21	1	25.800	0	1	1	2007.94500
1337	61	1	29.070	0	0	2	29141.36030

1338 rows × 7 columns

SPLITTING THE FEATURES AND TARGET

```
X1 = insurance_dataset.drop(columns='charges',axis=1)
Y1 = insurance_dataset['charges']
```

```
print(X1)
```

```
      age  sex     bmi  children  smoker  region
0      19    1  27.900         0       0       1
1      18    0  33.770         1       1       0
2      28    0  33.000         3       1       0
3      33    0  22.705         0       1       2
4      32    0  28.880         0       1       2
...   ...  ...     ...       ...     ...     ...
1333   50    0  30.970         3       1       2
1334   18    1  31.920         0       1       3
1335   18    1  36.850         0       1       0
1336   21    1  25.800         0       1       1
1337   61    1  29.070         0       0       2

[1338 rows x 6 columns]
```

```
print(Y1)
```

```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

SPLITTING THE DATA INTO TRAINING DATA AND TESTING DATA

```
X_train,X_test,Y_train,Y_test = train_test_split(X1,Y
1,test_size=0.2,random_state=2)


print(X1.shape , X_train.shape , X_test.shape)


print(X1.shape , X_train.shape , X_test.shape)
```

MODEL TRAINING

LINEAR REGRESSION

```
# Loading the Linear Regression Model
regressor = LinearRegression()
```

```
regressor.fit(X_train , Y_train)
```

```
LinearRegression()
```

MODEL EVALUATION

```
# Prediction on Training Data
training_data_prediction = regressor.predict(X_train)
```

```
# R Squarred Value
r2_train = metrics.r2_score(Y_train,training_data_pre
diction)
print('R Squarred Value : ',r2_train)
```

```
R Squarred Value :  0.7518195459072954
```

```
# Prediction on Test Data
test_data_prediction = regressor.predict(X_test)
```

```
#RMSE Value
from sklearn.metrics import mean_squared_error
```

```
rmse_linear = (np.sqrt(mean_squared_error(Y_test, tes
t_data_prediction)))
print('RMSE : {0:.3f}'.format(rmse_linear))
```

```
RMSE : 6182.956
```

```
# R Squarred Value
r2_test = metrics.r2_score(Y_test,test_data_predictio
n)
print('R Squarred Value : ',r2_test)
accuracy_lr = round(r2_score(Y_test,test_data_predict
ion),2)*100
print('Accuracy :',accuracy_lr,'%')
```

```
R Squarred Value :  0.7454471618659976
Accuracy : 75.0 %
```

BUILDING A PREDICTIVE SYSTEM

```
input_data = (30,0,22.85,1,0,3)

# Changing input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array
input_data_reshaped = input_data_as_numpy_array.resha
pe(1,-1)

prediction = regressor.predict(input_data_reshaped)
print(prediction)
```

DECISION TREE REGRESSION

```
#Building & Loading Model
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(max_depth = 3)


# Fitting the model
dt.fit(X_train,Y_train)


DecisionTreeRegressor(max_depth=3)
```

MODEL EVALUATION

```
# Prediction on Training Data
train_data_prediction = dt.predict(X_train)


# R Squarred Value
r2_train_DT = metrics.r2_score(Y_train,train_data_pre
diction)
print('R Squarred Value : ',r2_train_DT)


R Squarred Value :  0.8582042344756327


# Prediction on Test Data
testing_data_prediction = dt.predict(X_test)


#RMSE Value
rmse_dt = (np.sqrt(mean_squared_error(Y_test, testing
_data_prediction)))
print('RMSE : {0:.3f}'.format(rmse_dt))
```

```
RMSE : 4731.645
```

```
# R Squarred Value
r2_test_DT = metrics.r2_score(Y_test,testing_data_prediction)
print('R Squarred Value : ',r2_test_DT)
accuracy_dt = round(r2_score(Y_test,testing_data_prediction),2)*100
print('Accuracy :',accuracy_dt,'%')
```

```
R Squarred Value :  0.8509231896803976
Accuracy : 85.0 %
```

BUILDING A PREDICTIVE SYSTEM

```
input_data = (30,0,22.85,1,0,3)

# Changing input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = dt.predict(input_data_reshaped)
print(prediction)
```

[18273.37754623]
C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature names
  warnings.warn(

RANDOM FOREST REGRESSION

```
#Building & Loading Model
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(max_depth = 3, n_estimator
s=500)
```

```
# Fitting the model
rf.fit(X_train , Y_train)
```

```
RandomForestRegressor(max_depth=3, n_estimators=500)
```

MODEL EVALUATION

```
# Prediction on Training Data
train_prediction = rf.predict(X_train)
```

```
# R Squarred Value
r2_train_RF = metrics.r2_score(Y_train,train_predicti
on)
print('R Squarred Value : ',r2_train_RF)
```

```
R Squarred Value :  0.8630053711712216
```

```
# Prediction on Test Data
```

```
test_prediction = rf.predict(X_test)
```

```
#RMSE Value
rmse_rf = (np.sqrt(mean_squared_error(Y_test, test_pr
ediction)))
print('RMSE : {0:.3f}'.format(rmse_rf))
```

```
RMSE : 4606.607
# R Squarred Value
r2_test_RF = metrics.r2_score(Y_test,test_prediction)
print('R Squarred Value : ',r2_test_RF)
accuracy_rf = round(r2_score(Y_test,test_prediction),
2)*100
print('Accuracy :',accuracy_rf,'%')
```

```
R Squarred Value :  0.8586981131974661
Accuracy : 86.0 %
```

BUILDING A PREDICTIVE SYSTEM

```
# input_data = (30,0,22.85,1,0,3)

# Changing input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array
input_data_reshaped = input_data_as_numpy_array.resha
pe(1,-1)

prediction = rf.predict(input_data_reshaped)
```

```
print(prediction)
```

```
[18165.55428273]
C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
RandomForestRegressor was fitted with feature names
  warnings.warn(
```

GRADIENT BOOST REGRESSION

```
#Building & Loading Model
from sklearn.ensemble import GradientBoostingRegresso
r
gbr = GradientBoostingRegressor(max_depth=3, n_estima
tors=50, learning_rate =.2)
```

```
# Fitting the model
gbr.fit(X_train , Y_train)
```

```
GradientBoostingRegressor(learning_rate=0.2,
n_estimators=50)
```

MODEL EVALUATION

```
# Prediction on Training Data
training_prediction = gbr.predict(X_train)
```

```
# R Squarred Value
r2_train_GBR = metrics.r2_score(Y_train,training_pred
iction)
print('R Squarred Value : ',r2_train_GBR)
```

```
R Squarred Value :  0.9052167524729932
```

```
# Prediction on Test Data
testing_prediction = gbr.predict(X_test)


#RMSE Value
rmse_gbr = (np.sqrt(mean_squared_error(Y_test, testin
g_prediction)))
print('RMSE : {0:.3f}'.format(rmse_gbr))


RMSE : 4449.358


# R Squarred Value
r2_test_GBR = metrics.r2_score(Y_test,testing_predict
ion)
print('R Squarred Value : ',r2_test_GBR)
accuracy_gbr = round(r2_score(Y_test,testing_predicti
on),2)*100
print('Accuracy :',accuracy_gbr,'%')



R Squarred Value :  0.8681803006293177
Accuracy : 87.0 %
```

BUILDING A PREDICTIVE SYSTEM

```
# input_data = (30,0,22.85,1,0,3)


# Changing input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array
input_data_reshaped = input_data_as_numpy_array.resha
pe(1,-1)

prediction = gbr.predict(input_data_reshaped)
print(prediction)
```

[18488.49862003]
C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
GradientBoostingRegressor was fitted with feature names
  warnings.warn(

MODEL EVALUATION COMPARISON

```
models = [('Linear Regression', rmse_linear, r2_train
, r2_test, accuracy_lr),
          ('Decision Tree Regression', rmse_dt, r2_tr
ain_DT, r2_test_DT, accuracy_dt),
          ('Random Forest Regression', rmse_rf, r2_tr
ain_RF, r2_test_RF, accuracy_rf),
          ('Gradient Boost Regression', rmse_gbr, r2_
train_GBR, r2_test_GBR, accuracy_gbr)]
```

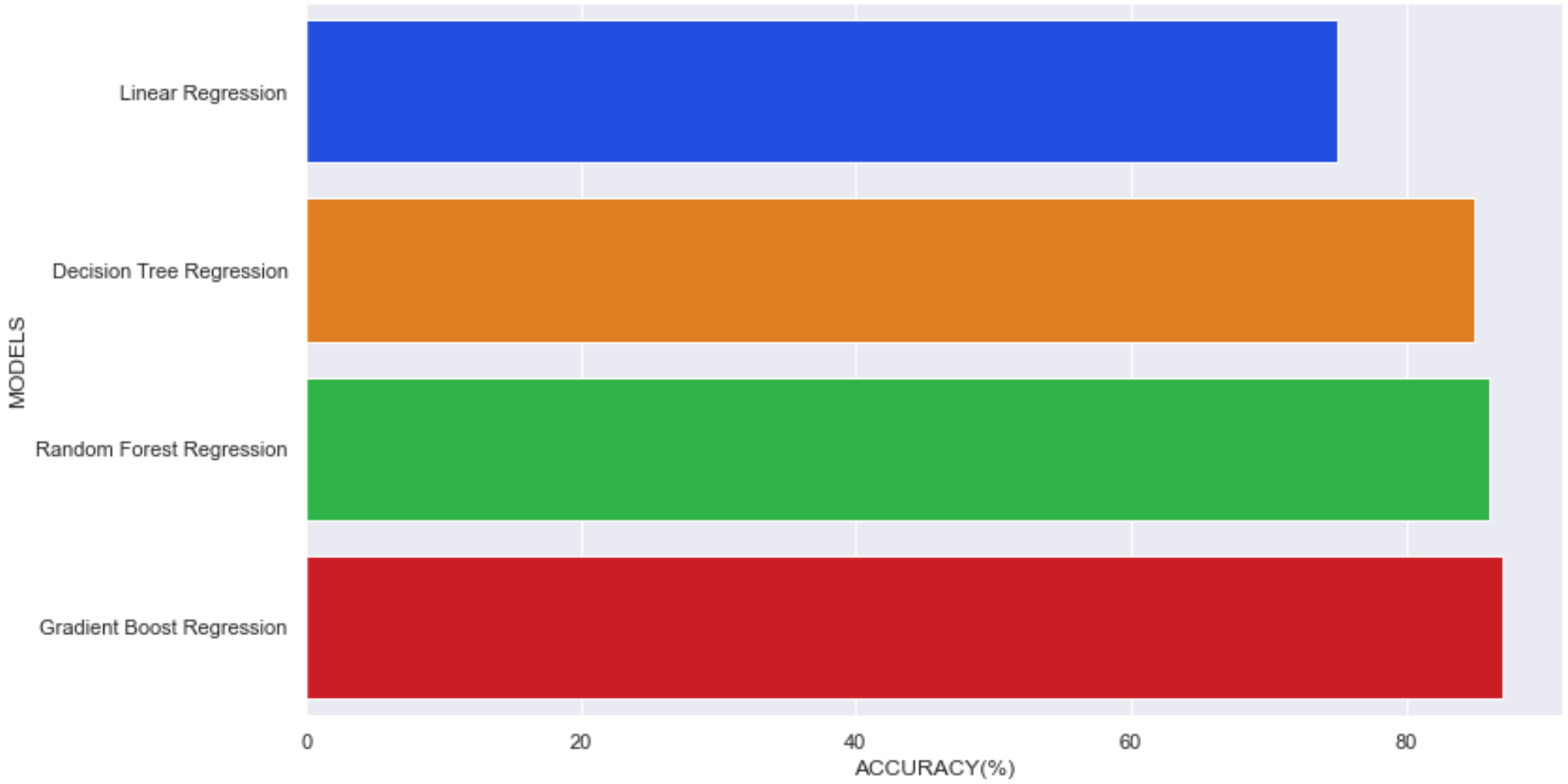
```
prediction = pd.DataFrame(data = models, columns=['Mo
del', 'RMSE', 'R2_Score(training)', 'R2_Score(test)',
 'Accuracy(%)'])
prediction
```

	Model	RMSE	R2_Score(training)	R2_Score(test)	Accuracy(%)
0	Linear Regression	6182.955535	0.751820	0.745447	75.0
1	Decision Tree Regression	4731.645479	0.858204	0.850923	85.0
2	Random Forest Regression	4606.606669	0.863005	0.858698	86.0

	Model	RMSE	R2_Score(training)	R2_Score(test)	Accuracy(%)
3	Gradient Boost Regression	4449.357579	0.905217	0.868180	87.0

```
# Plotting of model performance
plt.figure(figsize=(12,7))
prediction.sort_values(by=['Accuracy(%)'], ascending=
True, inplace=True)

sns.barplot(x='Accuracy(%)', y='Model',data = predict
ion, palette='bright')
plt.xlabel('ACCURACY(%)')
plt.ylabel('MODELS')
plt.show()
```


Linear Regression
Decision Tree Regression
Random Forest Regression
Gradient Boost Regression
MODELS
0
20
40
60
80
ACCURACY(%)

CONCLUSION

Therefore we conclude that Gradient Boost Regression gives us the best accuracy (87%) and the predicted medical insurance cost price using this algorithm is rs `18488.49862003`.

FUTURE SCOPE

In this project , we discussed some of the traditional regression models for our proposed problem statement . Moving forward some of the other techniques like Support Vector Machine (SVM) , XGBoost and Stochastic Gradient Boosting needs to be addressed as the future work . Several optimization techniques such as the Genetic Algorithm or the Gradient Descent Algorithm maybe applied on top of model evaluation. We can also apply some feature selection techniques to our dataset before we train our model to gain a good accuracy value as some of the features may be omitted while predicting the charges. Besides a model to perform well a good balanced dataset with a greater number of observations is required which will reduce the variability of the model so in the future if, we get more data than the model can be trained well.

Certificate

This is to certify that Mr Anuvab Chatterjee of ASANSOL ENGINEERING COLLEGE, registration number: 201080100110092 , has successfully completed a project on **Medical Insurance Cost Prediction using Machine Learning with Python** under the guidance of Mr. Anindya Banerjee.

Mr Anindya Banerjee.

ASANSOL ENGINEERING COLLEGE

Certificate

This is to certify that Ms Manvi Bishnu of ASANSOL ENGINEERING COLLEGE, registration number: 201080100110128 , has successfully completed a project on **Medical Insurance Cost Prediction using Machine Learning with Python** under the guidance of Mr. Anindya Banerjee.

Mr Anindya Banerjee.
ASANSOL ENGINEERING COLLEGE

Certificate

This is to certify that Mr Rohan Pramanik of ASANSOL ENGINEERING COLLEGE, registration number: 201080100110086 , has successfully completed a project on **Medical Insurance Cost Prediction using Machine Learning with Python** under the guidance of Mr Mr. Anindya Banerjee.

Mr Anindya Banerjee.
ASANSOL ENGINEERING COLLEGE

Certificate

This is to certify that Mr Shinjon Mukherjee of ASANSOL ENGINEERING COLLEGE, registration number: 201080100110072 , has successfully completed a project on **Medical Insurance Cost Prediction using Machine Learning with Python** under the guidance of Mr. Anindya Banerjee.

Mr Anindya Banerjee.
ASANSOL ENGINEERING COLLEGE