Project Report
Medical Insurance Cost Prediction

1. Introduction

Medical emergencies can occur at any moment without any prior notice. In such cases, a medical insurance policy is your saviour. Nowadays, companies want to predict the insurance charges of a person based on their profile before they take up the insurance. Therefore, our task is to find the insurance cost of a person based on their features.

2. Problem Statement

To accurately predict the insurance charges of a person based on certain features. The features are- the person's age, sex, BMI, number of children, region, and whether he/she smokes or not.

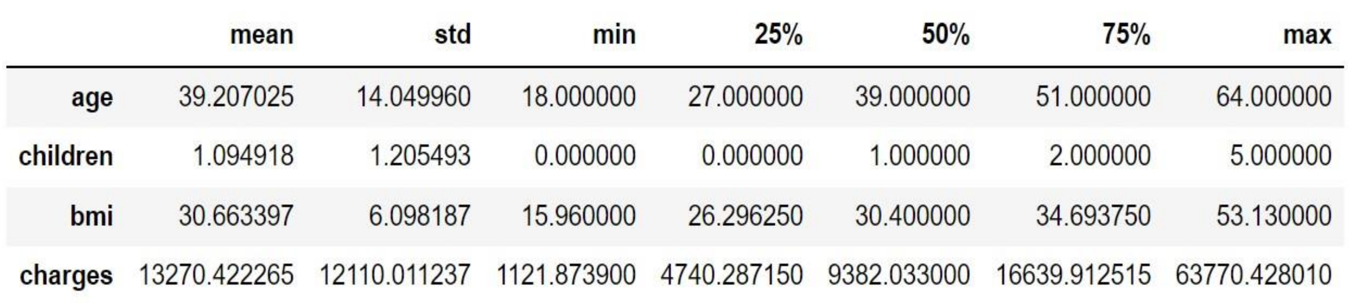
3. Our Approach in Brief

After visualizing and pre-processing the data, I split the data into training and testing data using stratified sampling, splitting it into a 4:1 ratio. I used the training dataset for training 6 models- Linear Regression, AdaBoost Regression, Gradient Boost Regression, Random Forest Regression, KNeighours Regression, and XGBoost Regression. I tested these models on the testing dataset and compared their performance on 3 factors: a. Accuracy of the model
b. Mean Absolute Error
c. Mean Squared Error

4. Salient features of the data

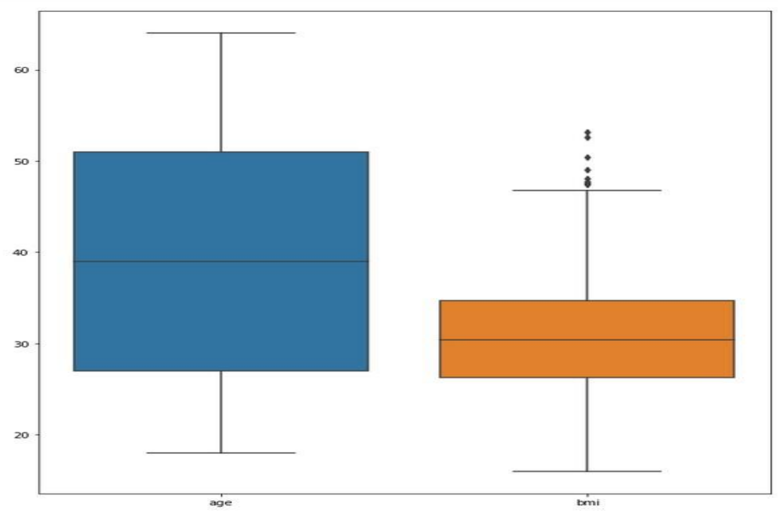
- A total of 7 columns and 1338 rows are present.
- None of the columns has a NULL value therefore, data cleaning is not required.
- Categorical Features: sex, smoker, region
- Continuous Features: age, children, BMI, charges
- Label/dependent variable : charges

5. Data Visualization/Understanding

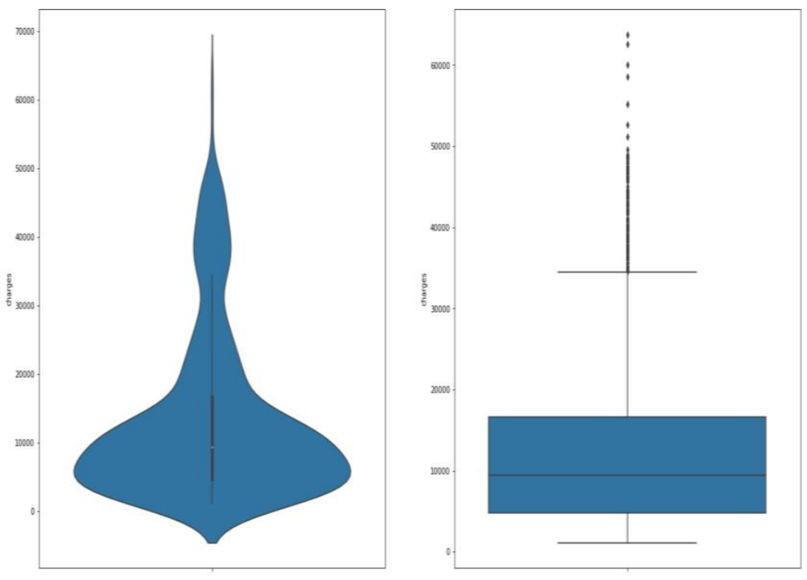


	mean	std	min	25%	50%	75%	max
age	39.207025	14.049960	18.000000	27.000000	39.000000	51.000000	64.000000
children	1.094918	1.205493	0.000000	0.000000	1.000000	2.000000	5.000000
bmi	30.663397	6.098187	15.960000	26.296250	30.400000	34.693750	53.130000
charges	13270.422265	12110.011237	1121.873900	4740.287150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

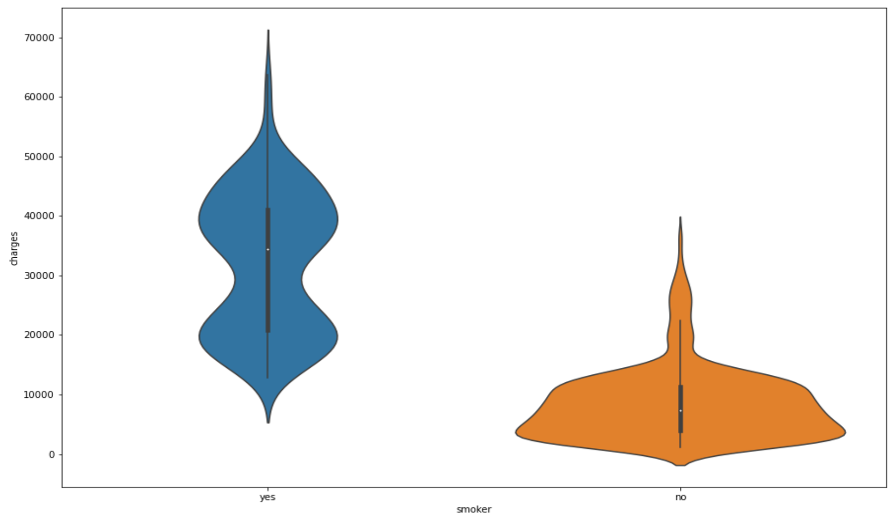
1. **There are 9 outliers in the BMI attribute in the data frame. So, we removed all 9 outliers from our dataset.**



2. **The insurance charges of most of the population are lying below 20000**

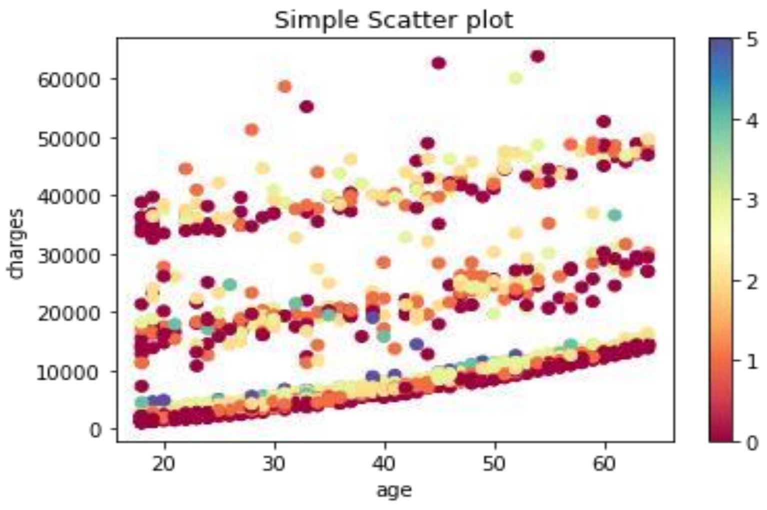


3. **If Charges are greater than Rs. 40,000, a high probability is that the person is a smoker. And for most non-smokers, insurance charges are less than Rs.20,000**

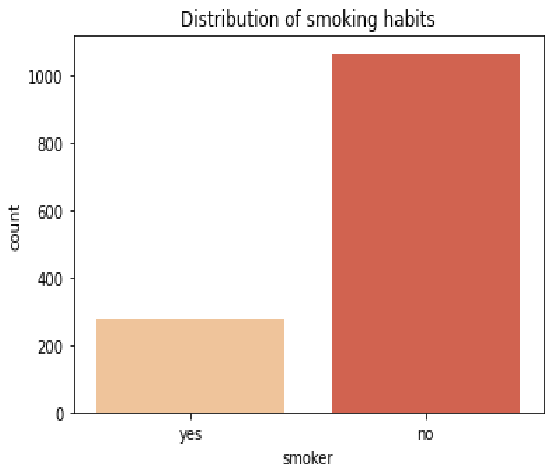


4. **Those who have charges<15000 contain customers having children either 0,1,2 and diversified across a complete range of age.**

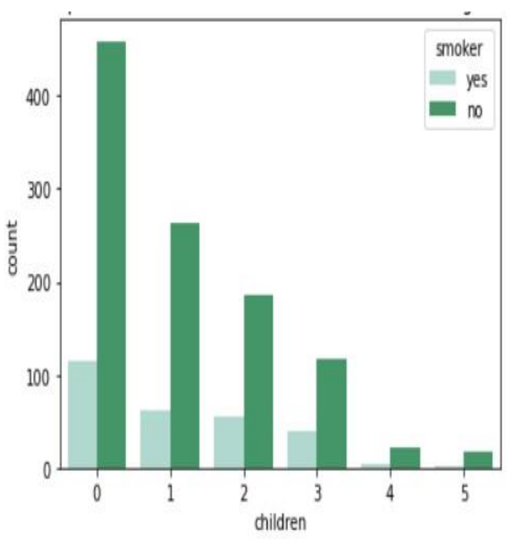
5. **Persons having children 2 or 3 are majorly paying less than 50000 and diversified across a complete range of age.**



6. **Almost 21% of the population from the dataset are smokers**

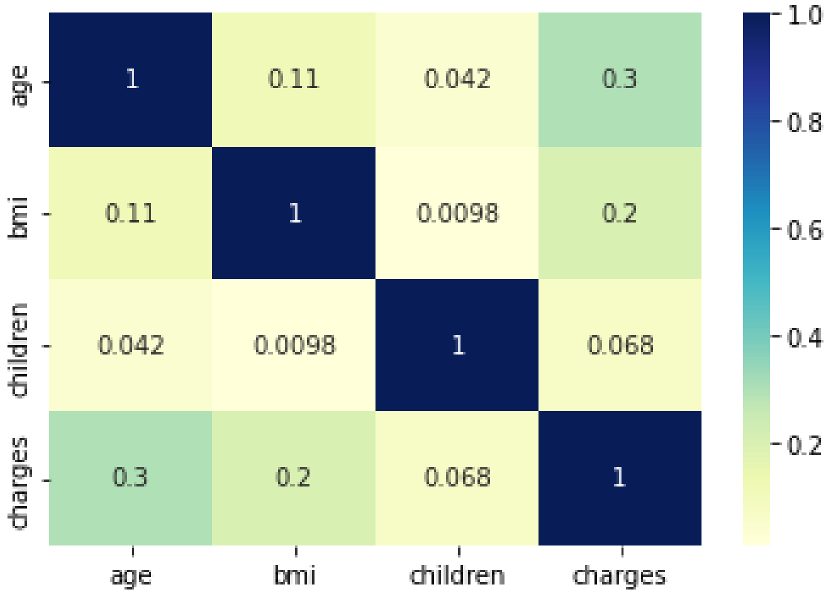


7. **The number of people who smoke decreases as the number of children increases.**



Correlation Heatmap:

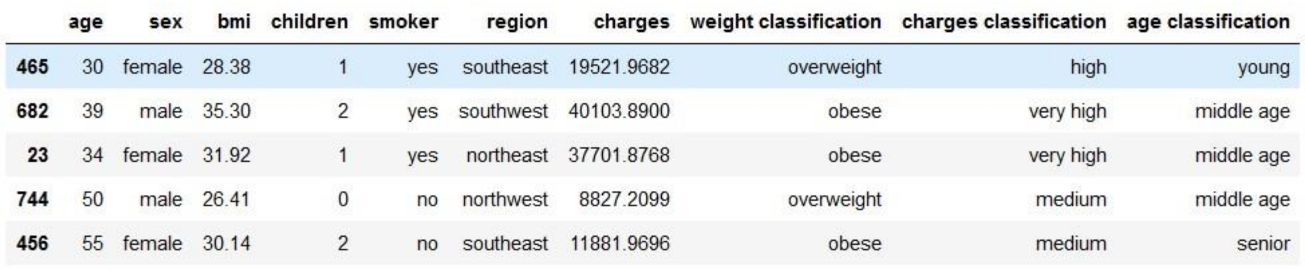
Inference: All the Numeric attributes are positively correlated to each other. The highest co-relation of charges are with age (0.3) and bmi (0.2).



Chi-Square Test:

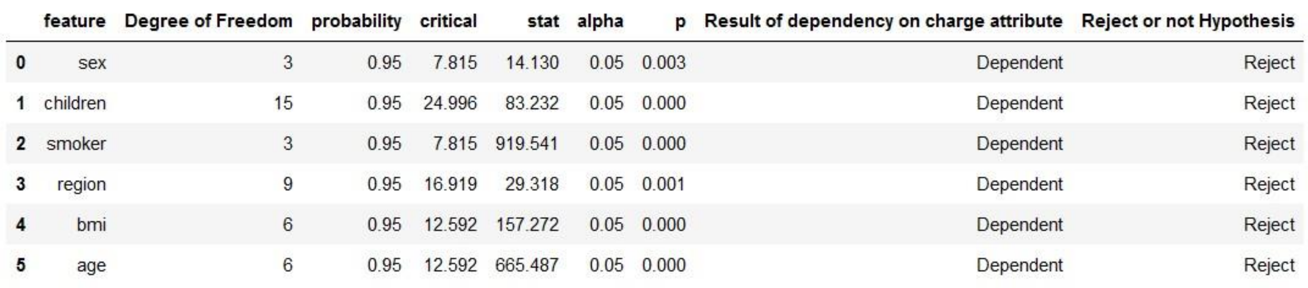
The Chi-square test is only applicable for Nominal Data. That's why we first categorize some Numeric Data. After that, we apply the chi-square test to find the correlation condition.

This is the sample data that we prepared after categorizing the numerical attributes into nominal attributes.



	age	sex	bmi	children	smoker	region	charges	weight classification	charges classification	age classification
465	30	female	28.38	1	yes	southeast	19521.9682	overweight	high	young
682	39	male	35.30	2	yes	southwest	40103.8900	obese	very high	middle age
23	34	female	31.92	1	yes	northeast	37701.8768	obese	very high	middle age
744	50	male	26.41	0	no	northwest	8827.2099	overweight	medium	middle age
456	55	female	30.14	2	no	southeast	11881.9696	obese	medium	senior

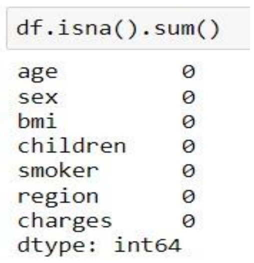
Result of Chi-Square Test:



	feature	Degree of Freedom	probability	critical	stat	alpha	p	Result of dependency on charge attribute	Reject or not Hypothesis
0	sex	3	0.95	7.815	14.130	0.05	0.003	Dependent	Reject
1	children	15	0.95	24.996	83.232	0.05	0.000	Dependent	Reject
2	smoker	3	0.95	7.815	919.541	0.05	0.000	Dependent	Reject
3	region	9	0.95	16.919	29.318	0.05	0.001	Dependent	Reject
4	bmi	6	0.95	12.592	157.272	0.05	0.000	Dependent	Reject
5	age	6	0.95	12.592	665.487	0.05	0.000	Dependent	Reject

From the Chi-square test, we can say that sex, no. of children region, weight classification, age classification and Smoking habit are co-related with the Charge Attribute**.**

6. Regressive Model Building



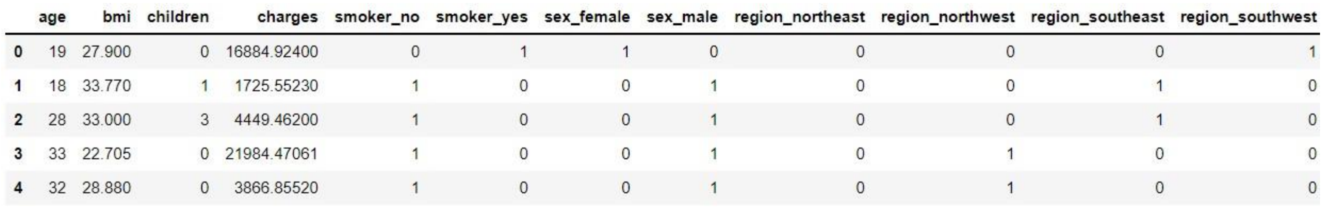
```
df.isna().sum()

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

Firstly, we checked for any null values in our dataset. We found out that there were no null values in our dataset.

- **Data Preprocessing for model building**

Then, we converted any categorical or binary columns into multiple columns by the process of one-hot encoding (using the *pd.get_dummies()* function). In this way, we one-hot encoded the *sex*, *smoker,* and *region* columns.



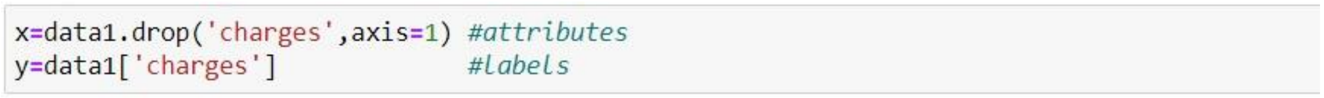
	age	bmi	children	charges	smoker_no	smoker_yes	sex_female	sex_male	region_northeast	region_northwest	region_southeast	region_southwest
0	19	27.900	0	16884.92400	0	1	1	0	0	0	0	1
1	18	33.770	1	1725.55230	1	0	0	1	0	0	1	0
2	28	33.000	3	4449.46200	1	0	0	1	0	0	1	0
3	33	22.705	0	21984.47061	1	0	0	1	0	1	0	0
4	32	28.880	0	3866.85520	1	0	0	1	0	1	0	0

As our dataset was small and had only *7 columns* and *1338 rows*, we chose not to do any *Principal component analysis (PCA)*.

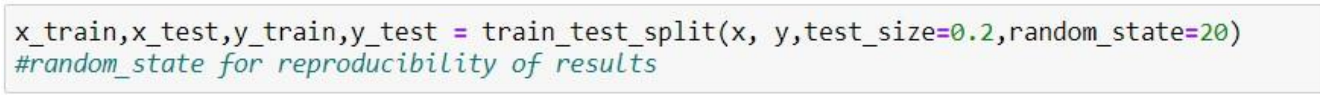
- **Splitting of Data into training and testing and data normalization**

Then we splitted our data into training set and testing set and further into *x_train, x_label (or x_test)* and *x_test, y_label (or y_test)* using the *train_test_split()* function. The test size was 20% of the whole dataset (a total of 268 rows) while the training size was 80% of the whole dataset (a total of 1070 rows).

We also included *random_state* in train_test_split for the reproducibility of the results. As the dataset was small so we wanted more data to be on the training set for better results, we set the ratio of test:train as 20:80.



```
x=data1.drop('charges',axis=1) #attributes
y=data1['charges']             #labels
```



```
x_train,x_test,y_train,y_test = train_test_split(x, y,test_size=0.2,random_state=20)
#random_state for reproducibility of results
```

Then we *normalized* the data using the StandardScalar*()* function. StandardScaler is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.

7. Model Building

For this insurance dataset, I have applied 6 machine learning models for our project. They are listed below as following:

1) **Linear Regression**- Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable.

Accuracy- 79.59%
Mean Squared Error- 30680789.79
Mean Absolute Error- 4042.20

2) **Random Forest Regression-** Random Forest Regression is a supervised learning algorithm that uses ensemble learning methods for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

Accuracy- 89.20%
Mean Squared Error- 16238027.49 **Mean Absolute Error-** 2234.26

3) **XGBoost Regression** - XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.

Accuracy- 87.15%
Mean Squared Error- 19315349.01 **Mean Absolute Error-** 2527.65

4) KNeighours Regression- KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood.

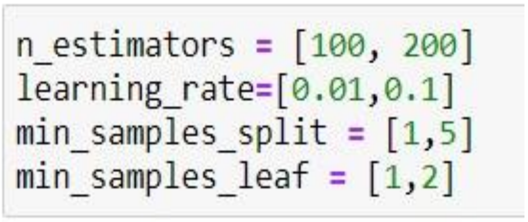
Accuracy- 85.62%
Mean Squared Error- 21614049.63 **Mean Absolute Error-** 2952.66

5) Gradient Boosting Regression- Gradient boosting is one of the variants of ensemble methods where you create multiple weak models and combine them to get better performance as a whole.

Accuracy- 90.42%
Mean Squared Error- 14391794.68 **Mean Absolute Error-** 2204.78

6) AdaBoost Regression- AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

Accuracy- 85.51%
Mean Squared Error- 21787053.53
Mean Absolute Error- 3679.73



```
n_estimators = [100, 200]
learning_rate=[0.01,0.1]
min_samples_split = [1,5]
min_samples_leaf = [1,2]
```

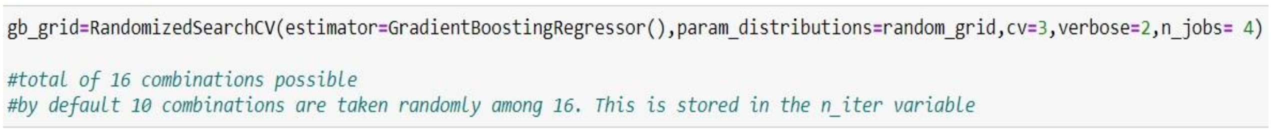
- **Final Selection**

I finally chose the Gradient Boosting Regressor as our final regressor as it had the highest accuracy (90.2%) among all the other six regressors.

Then I used *RandomizedSearchCV* for *hyperparameter tuning* and *k-fold cross-validation* to find the best model parameters for highest accuracy. The features and their values to be run were -

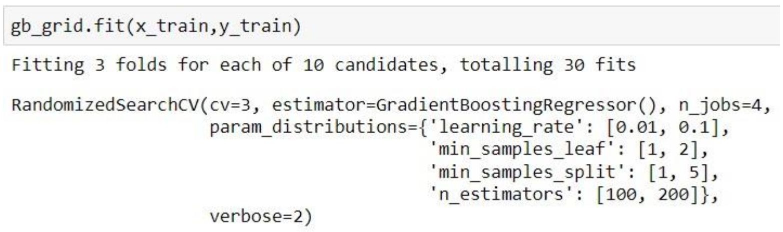
- **Hyperparameter Tuning of Gradient Boosting Regressor**

A ***3-fold cross-validation*** was used **(cv=3)** and the no. of combinations to be trained upon was 10 **(n_iter=10 which is default)** out of **16(2^4)**. Thus, a total of **30** combinations were trained.



```
gb_grid=RandomizedSearchCV(estimator=GradientBoostingRegressor(),param_distributions=random_grid,cv=3,verbose=2,n_jobs= 4)

#total of 16 combinations possible
#by default 10 combinations are taken randomly among 16. This is stored in the n_iter variable
```

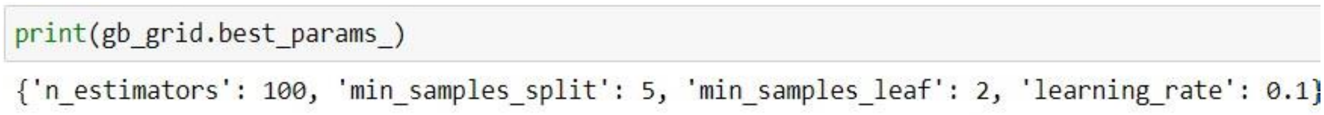


```
gb_grid.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

RandomizedSearchCV(cv=3, estimator=GradientBoostingRegressor(), n_jobs=4,
                   param_distributions={'learning_rate': [0.01, 0.1],
                                        'min_samples_leaf': [1, 2],
                                        'min_samples_split': [1, 5],
                                        'n_estimators': [100, 200]},
                   verbose=2)
```

We found out the best parameters to be -

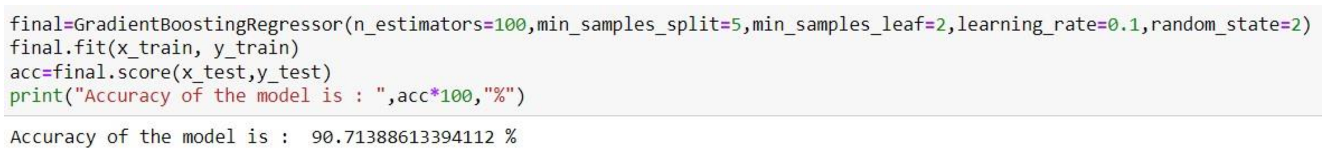


```
print(gb_grid.best_params_)
```

```
{'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'learning_rate': 0.1}
```

8. Model Evaluation

The final accuracy that was obtained using these parameters was **90.71%**. The Root mean squared error was **3736.81** and the Mean absolute error was **2169.19.**



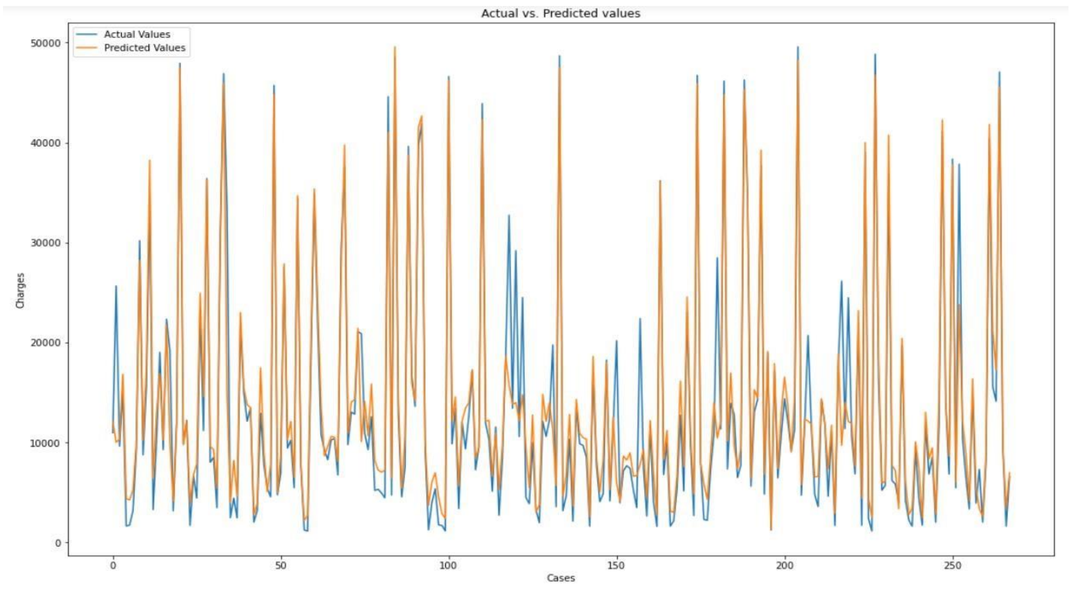
```
final=GradientBoostingRegressor(n_estimators=100,min_samples_split=5,min_samples_leaf=2,learning_rate=0.1,random_state=2)
final.fit(x_train, y_train)
acc=final.score(x_test,y_test)
print("Accuracy of the model is : ",acc*100,"%")
```

```
Accuracy of the model is :  90.71388613394112 %
```

The ***Root mean squared error*** was **3736.81** and the ***Mean absolute error*** was **2169.19**. As there is no ***tree_*** parameter in ***GradientBoostingRegressor***, so we didn't draw the tree.

9. Results Interpretations

- **Actual versus Predictions Result**



I finally plotted the predicted vs actual value in the testing dataset. It can be inferred that apart from some data points almost all other data points were **predicted almost accurately** by the model.