Regression Algorithm

Problem Statement or Requirement:

A client's requirement is, he wants to predict the insurance charges based on the several parameters. The Client has provided the dataset of the same. As a data scientist, you must develop a model which will predict the insurance charges.

1.) **Identify your problem statement**

In dataset 'age', 'sex', 'bmi', 'children', 'smoker' are input and 'charges' is output by usi ng this data need to predict the insurance charges.

2.) **Tell basic info about the dataset (Total number of rows, columns)**

```
# to find number of rows and col's
no_of_rows_and_col=dataset.shape
print("Number of rows and columns:",no_of_rows_and_col)
# Here the number of rows = 1338 and the columns = 6
Number of rows and columns: (1338, 6)
```

3.) **Mention the pre-processing method if you're doing any (like converting string to number – nominal data)**

Here in this given dataset sex and smoker have ordinal data, so need to change that as numeric data.

```
# preprocessing to convert categorical value to numerical
dataset.sex[dataset.sex == 'male']=1
dataset.sex[dataset.sex == 'female']=2
dataset.smoker[dataset.smoker == 'yes']=1
dataset.smoker[dataset.smoker == 'no']=0
```

sex – male -1, female -2

smoker – yes -1, no -0

4.) **Develop a good model with r2_score. You can use any machine learning algorithm; you can create many models. Finally, you have to come up with final model.**

Developed a model using **sklearn**. The algorithms are **Multiple Linear Regression, Support Vector Machine, Decision Tree Regression and Random Forest**.

5.) **All the research values (r2_score of the models) should be documented.**

(You can make tabulation or screenshot of the results.)

Dataset = insurance_pre.csv

1. **Multiple Linear Regression** – **R^2 value = 0.789**

2. **Support Vector Machine**
   C-Support Vector Classification.
   C: float, default=1.0
   Kernel: {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, **default='rbf'**

S. No	Hyper parameter 'c'	Linear R^2 value	rbf R^2 value	Poly R^2 value	Sigmoid R^2 value
1	10	0.462	-0.032	0.038	0.039
2	100	0.628	0.320	0.617	0.527
3	1000	0.764	0.810	0.856	0.287
4	2000	0.744	0.854	0.860	-0.593
5	3000	0.741	0.866	0.859	-2.124

   Overall in SVM **kernel='rbf', C=3000** is giving a better accuracy than MLR. **R^2 value = 0.866**

3. **Decision Tree**

   Criterion: *{"squared_error", "friedman_mse", "absolute_error", "poisson"}, default="squared_error"*

   Splitter: *{"best", "random"}, default="best"*

   max_features : *int, float or {"sqrt", "log2","auto"}, default=None*

S. No	Criterion	Splitter	max_features	R^2 value
1				0.679
2	squared_error	best		0.683
3	squared_error	best	sqrt	0.696
4	squared_error	best	Log2	0.619
5	squared_error	best	auto	0.690
6	squared_error	random		0.663
7	squared_error	random	sqrt	0.713
8	squared_error	random	Log2	0.696
9	squared_error	random	auto	0.718
10	friedman_mse	best		0.687
11	friedman_mse	best	sqrt	0.708
12	friedman_mse	best	Log2	0.660
13	friedman_mse	best	auto	0.687
14	friedman_mse	random		0.690
15	friedman_mse	random	sqrt	0.610
16	friedman_mse	random	Log2	0.648
17	friedman_mse	random	auto	0.722
18	*absolute_error*	best		0.671
19	*absolute_error*	best	sqrt	0.715
20	*absolute_error*	best	Log2	0.681
21	*absolute_error*	best	auto	0.695
22	*absolute_error*	random		0.727
23	*absolute_error*	random	sqrt	0.669
24	*absolute_error*	random	Log2	0.695
25	*absolute_error*	random	auto	0.713
26	*poisson*	best		0.727
27	*poisson*	best	sqrt	0.703
28	*poisson*	best	Log2	0.704
29	*poisson*	best	auto	0.724
30	*poisson*	random		0.720
31	*poisson*	random	sqrt	0.651
32	*poisson*	random	Log2	0.711
33	*poisson*	random	auto	0.728

Overall in Decision Tree **(Poisson,random,auto)** is giving a better accuracy **R^2 value = 0.728** but not that much while comparing SVM.
In **SVM** **R^2 value = 0.866.**

4. **Random Forest**

n_estimators: ***int, default=100***
Criterion: ***{"squared_error", "absolute_error", "friedman_mse", "poisson"}, default="squared_error"***
max_features : ***{"sqrt", "log2", None,auto}, int or float, default=1.0***
random_state: ***int, RandomState instance or None, default=None***

S. No	n_estimators	Criterion	max_features	R^2 value
1				0.854
2	100	squared_error		0.858
2	100	squared_error	sqrt	0.873
3	100	squared_error	Log2	0.869
4	100	squared_error	auto	0.851
5	50	squared_error		0.850
6	50	squared_error	sqrt	0.868
7	50	squared_error	Log2	0.870
8	50	squared_error	auto	0.851
9	100	absolute_error		0.850
10	100	absolute_error	sqrt	0.872
11	100	absolute_error	Log2	0.876
12	100	absolute_error	auto	0.851
13	50	absolute_error		0.852
14	50	absolute_error	sqrt	0.869
15	50	absolute_error	Log2	0.873
16	50	absolute_error	auto	0.858
17	100	friedman_mse		0.856
18	100	friedman_mse	sqrt	0.871
19	100	friedman_mse	Log2	0.867
20	100	friedman_mse	auto	0.856
21	50	friedman_mse		0.851
22	50	friedman_mse	sqrt	0.873
23	50	friedman_mse	Log2	0.868
24	50	friedman_mse	auto	0.855
25	100	poisson		0.854
26	100	poisson	sqrt	0.870
27	100	poisson	Log2	0.869
28	100	poisson	auto	0.858
29	50	poisson		0.855
30	50	poisson	sqrt	0.869
31	50	poisson	Log2	0.870
32	50	poisson	auto	0.857

Here **R^2 value** is up to mark 0.876
This model (100, absolute_error, log2) is **Good** for this data. While, comparing other model this **Random forest** is giving better result.

6.) Mention your final model, justify why u have chosen the same.

The Below table will show the best results of all model.

S. No	Model	R^2 value
1	Multiple Linear Regression	0.789
2	Support Vector Machine	0.866
3	Decision Tree Regression	0.728
4	Random Forest	**0.876**

So, The Best model for this dataset is "**Random Forest**".