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.

Data set Analysis:

1. Domain selection: Machine Learning
   Learning selection: Supervised Learning
   Method: Regression

2. Dataset contains 6 columns: Age, sex, BMI, children, smoker, charge &1138 rows

3. I could see 2 nominal columns: sex and smoker which must be converted with the help of one hot encoding

Since the data involves multiple inputs, here we can’t use Simple Linear Regression. So, let's start with other available algorithms.

1.MLR – Multiple Linear Regression

R2_score value: `0.7894790349867009`

```
418    14418.28040
920    13451.12200

[402 rows x 1 columns],
(402, 1))
```

```
]: from sklearn.linear_model import LinearRegression
   regressor=LinearRegression()
   regressor.fit(x_train,y_train)
```

```
]: ▾ LinearRegression
   LinearRegression()
```

```
]: weight=regressor.coef_
   weight
```

```
]: array([[   257.8006705 ,    321.06004271,    469.58113407,    -41.74825718,
          23418.6671912 ]])
```

```
]: bais=regressor.intercept_
   bais
```

```
]: array([-12057.244846])
```

```
]: y_pred=regressor.predict(x_test)
```

```
]: from sklearn.metrics import r2_score
   R_value=r2_score(y_test,y_pred)
```

```
]: R_value
```

```
]: 0.7894790349867009
```

2. Support Vector Machine – SVM:

R2 score value (without parameter tuning): -0.08338238593619329

With Parameter turning:

S.NO	Hyper parameter	Linear	Non-Linear (RBF)	Poly	Sigmoid
1	C=0.01	-0.08883133439168489	-0.08964553739867864	-0.08956828487671076	-0.0895650159341983
2	C=0.10	-0.080959968427891	-0.08907451521042731	-0.08830237655410711	-0.08826991450485111
3	C=1	-0.010102665316081394	-0.08338238593619329	-0.07569965570860893	-0.07542924281107188
4	C=10	0.46246841423396834	-0.03227329390671052	0.038716222760231456	0.03930714378274347
5	C=100	0.6288792857320369	0.3200317832050831	0.6179569624059795	0.5276103546510407
6	C=1000	0.7649311738597411	0.8102064851758545	0.8566487675946572	0.28747069486976173
7	C=2000	0.7440418308107846	0.8547766425392979	0.8605579258597704	-0.5939509731283505

```
 [402 rows x 1 columns],
 (402, 1))
```

[217]:
```python
from sklearn.svm import SVR
regressor=SVR(kernel='poly',C=2000)
regressor.fit(x_train,y_train)
```

```
C:\Anaconda\Lib\site-packages\sklearn\utils\validation.py:1408: DataConversionWarning: A co
lumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_s
amples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[217]:
```
SVR
SVR(C=2000, kernel='poly')
```

[218]:
```python
regressor.intercept_
```

[218]:
```
array([8281.60615314])
```

[219]:
```python
regressor.n_support_
```

[219]:
```
array([936], dtype=int32)
```

[220]:
```python
y_pred=regressor.predict(x_test)
```

[221]:
```python
from sklearn.metrics import r2_score
R_value=r2_score(y_test,y_pred)
```

[222]:
```python
R_value
```

[222]:
```
0.8605579258597704
```

3. Decision Tree:

R2 score value (without parameter tuning): 0.6966865293326343

With Parameter turning:

s.no	criterion	Max feature	splitter	R score
1.	Squared error	sqrt	best	0.702035228596572
2.	Squared error	log2	best	0.7419054872546678
3.	Squared error	sqrt	random	0.682074724492034
4.	Squared error	log2	random	0.6980896260139877
5.	Squared error	none	best	0.6996994481671908
6.	Squared error	none	random	0.7358200878574843
7.	Friedman mse	sqrt	best	0.7381415595482762
8.	Friedman mse	log2	best	0.7319704870186678
9.	Friedman mse	sqrt	random	0.6457245773642908
10.	Friedman mse	log2	random	0.688878706147374
11.	Friedman mse	none	best	0.6879110478137831
12.	Friedman mse	none	random	0.6949890484996232
13.	MAE	sqrt	best	0.7297458122874192
14.	MAE	log2	best	0.6420218622840612
15.	MAE	sqrt	random	0.7607657594740639
16.	MAE	log2	random	0.7240231565275986
17.	MAE	none	best	0.6563984953703915
18.	MAE	none	random	0.6888543930260137
19.	Poisson	sqrt	best	0.7052009320217727
20.	Poisson	log2	best	0.6199992359174482
21.	Poisson	sqrt	random	0.646019598776802
22.	Poisson	log2	random	0.6597185842121801
23.	Poisson	none	best	0.7240970726548698
24.	Poisson	none	random	0.7360710221488609

4. Random Forest:

R2 score value (without parameter tuning): `0.857238919537012`

With Parameter turning:

s.no	N estimators	criterion	Max features	R score
1	10	Squared error		0.8297825655643111
2	100	Squared error		0.8575979556497877
3	10	Friedman mse		0.8318936415190324
4	100	Friedman mse		0.8480668109283795
5	10	MAE		0.8368339827516286
6	100	MAE		0.856093154513048
7	10	Poisson		0.8274737517834232
8	100	Poisson		0.8540244751233814
9	100	Squared error	sqrt	0.8724894738130522
10	100	Squared error	log2	0.8676627351539128
11	100	MAE	sqrt	0.8704424718543277
12	100	MAE	log2	0.8736096572373303

```
[18]: n sklearn.ensemble import RandomForestRegressor
      ressor=RandomForestRegressor(n_estimators=100,criterion="absolute_error",max_features='log2')
      ressor.fit(x_train,y_train)
```

```
C:\Anaconda\Lib\site-packages\sklearn\base.py:1389: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), fo
r example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

```
[18]: RandomForestRegressor
      RandomForestRegressor(criterion='absolute_error', max_features='log2')
```

```
[19]: y_pred=regressor.predict(x_test)
```

```
[20]: from sklearn.metrics import r2_score
      R_value=r2_score(y_test,y_pred)
```

```
[21]: R_value
```

```
[21]: 0.8736096572373303
```

Conclusion:

For this dataset, I use different types of algorithm and hyper parameters from this I find the better performing algorithm in **Random Forest.** In that I created a finial model.

The best predicted value of **R_Score is** 0.8736096572373303 in **random forest (n_estimators=100, criterion=MAE,max feature=log2).**