Regression Assignment: V1

Steps: Import all the necessary Libraries

* First 5 Rows of the dataset:

S.no	age	sex	bmi	children	smoker	charges
0	19	female	27.9	0	yes	16884.92
1	18	male	33.77	1	no	1725.552
2	28	male	33	3	no	4449.462
3	33	male	22.705	0	no	21984.47
4	32	male	28.88	0	no	3866.855

1. Identifying the problem statement:

- Stage 1:
  * We have Input data's are Numbers - so Domain will be Machine Learning
- Stage 2:
  * Supervised Learning- It's a Clear requirement and we have IP/OP data.
- stage 3:
  * Numerical values (Target features) - so we can go with Regression

2. Basic info about the dataSet:

* Total Row : 1388
* Total Columns : 6
* Dataset contains both numerical and categorical values
* Column names : AGE, SEX, BMI, CHILDREN, SMOKER, CHARGES(Traget variable)
* age,children are Integer type.
* bmi and charges are Float type.
* sex and smoker are in Object type.
* There is no null values in the dataset.
* Requirnment is clear.

3.Preprocessing Method:

Converting categorical variable to continuous numerical variable

After Converting data:

s.no	age	bmi	children	charges	sex_male	smoker_yes
0	19	27.9	0	16884.924	0	1

1	18	33.77	1	1725.5523	1	0
2	28	33	3	4449.462	1	0
3	33	22.705	0	21984.47061	1	0
4	32	28.88	0	3866.8552	1	0

- Separate the data as independent and dependent
- Split the data using train_test_split concept
- If the data is looks too different within the independent variable then go for Scaling concept

4. Create different Models to predict the charges and choose the best model with high r2_score value.

a) Multiple Linear Regression:

Results: **R2_Score = 0.77239**

b) SVM Regression: Default value:

Model 2 - SVR

[131]:

```
from sklearn.svm import SVR
svr_model = SVR()
svr_model.fit(X_train, y_train)
```

[131]:

```
▾ SVR
SVR()
```

[133]:

```
y_pred = svr_model.predict(X_test)
```

[135]:

```
svr_r2score = r2_score(y_test,y_pred)
svr_r2score
```

[135]:

```
-0.09743369320622963
```

Model with Different parameters

Best : RBF, C=3000 and R2_Score = 0.84223

```
# SVR_R2_Score : Results with all the combinations:
for kernel, C, score in svr_r2score:
    print("Kernel:",kernel,"- C:",C,"- R² Score",score)
```

```
Kernel: linear - C: 10 - R² Score 0.4411154085859137
Kernel: linear - C: 100 - R² Score 0.6205250918367728
Kernel: linear - C: 500 - R² Score 0.7486844603269274
Kernel: linear - C: 1000 - R² Score 0.7187303718390283
Kernel: linear - C: 2000 - R² Score 0.717185774596472
Kernel: linear - C: 3000 - R² Score 0.7161570985448058
Kernel: rbf - C: 10 - R² Score -0.044010385361080706
Kernel: rbf - C: 100 - R² Score 0.3229298217686596
Kernel: rbf - C: 500 - R² Score 0.6597447201916936
Kernel: rbf - C: 1000 - R² Score 0.7923789012887561
Kernel: rbf - C: 2000 - R² Score 0.8332041538541531
Kernel: rbf - C: 3000 - R² Score 0.8422384746495604
Kernel: poly - C: 10 - R² Score 0.024580030588300605
Kernel: poly - C: 100 - R² Score 0.5897995020893756
Kernel: poly - C: 500 - R² Score 0.8038815140652048
Kernel: poly - C: 1000 - R² Score 0.8338303273683121
Kernel: poly - C: 2000 - R² Score 0.8403113295667679
Kernel: poly - C: 3000 - R² Score 0.8390286005966449
Kernel: sigmoid - C: 10 - R² Score 0.022978265830825184
Kernel: sigmoid - C: 100 - R² Score 0.5099640007962808
Kernel: sigmoid - C: 500 - R² Score 0.46829640989809973
Kernel: sigmoid - C: 1000 - R² Score 0.3012743569891182
Kernel: sigmoid - C: 2000 - R² Score -0.8798843869243349
Kernel: sigmoid - C: 3000 - R² Score -2.9047335174024607
```


c) Decision Tree Regressor:

With Default Values score:

Model 3 - Decision Tree Regressor

[187]:

```
from sklearn.tree import DecisionTreeRegressor
DT_model = DecisionTreeRegressor()
DT_model.fit(X_train,y_train)
y_pred = DT_model.predict(X_test)
DT_r2 = r2_score(y_test,y_pred)
DT_r2
```

[187]:

```
0.7064796314633288
```

With different parameters:

Model 3 - Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
criterion1 = ['squared_error','friedman_mse','absolute_error','poisson']
max_ft1 = ['sqrt','log2']
spliter1 = ['best','random']
DT_r2 = []

for cri in criterion1:
    for mx_ft in max_ft1:
        for split in spliter1:
            DT_model = DecisionTreeRegressor(criterion=cri,max_features=mx_ft,splitter=split)
            DT_model.fit(X_train,y_train)
            y_pred = DT_model.predict(X_test)
            dt_r2 = r2_score(y_test,y_pred)
            DT_r2.append((cri,mx_ft,split,dt_r2))
```

```
for criterion,max_features,splitter,score in DT_r2:
    print("Criterion:",criterion,"max_features:",max_features,"splitter:",splitter,"R2_Score:",score)
```

```
Criterion: squared_error max_features: sqrt splitter: best R2_Score: 0.6905149345691342
Criterion: squared_error max_features: sqrt splitter: random R2_Score: 0.6089158084430758
Criterion: squared_error max_features: log2 splitter: best R2_Score: 0.7208084204160041
Criterion: squared_error max_features: log2 splitter: random R2_Score: 0.5885137699183175
Criterion: friedman_mse max_features: sqrt splitter: best R2_Score: 0.741628247754792
Criterion: friedman_mse max_features: sqrt splitter: random R2_Score: 0.632694706926831
Criterion: friedman_mse max_features: log2 splitter: best R2_Score: 0.6353895211454534
Criterion: friedman_mse max_features: log2 splitter: random R2_Score: 0.6909830368433382
Criterion: absolute_error max_features: sqrt splitter: best R2_Score: 0.6465435342030181
Criterion: absolute_error max_features: sqrt splitter: random R2_Score: 0.7744421292327105
Criterion: absolute_error max_features: log2 splitter: best R2_Score: 0.712613522330376
Criterion: absolute_error max_features: log2 splitter: random R2_Score: 0.6719344044572477
Criterion: poisson max_features: sqrt splitter: best R2_Score: 0.5326888585822843
Criterion: poisson max_features: sqrt splitter: random R2_Score: 0.6616041177188116
Criterion: poisson max_features: log2 splitter: best R2_Score: 0.5525323495374564
Criterion: poisson max_features: log2 splitter: random R2_Score: 0.6235679214227591
```

After Using diff parameters the best **R2_Score is : 0.774421**

D) Random Forest Regressor: with Default values

Random_Forest_Regressor

[204]:

```
from sklearn.ensemble import RandomForestRegressor
rff = RandomForestRegressor()
rff.fit(X_train,y_train)
y_pred = rff.predict(X_test)
randomforest_r2score = r2_score(y_test,y_pred)
randomforest_r2score
```

[204]:

```
0.8373310073237741
```

With different parameters:

Random_Forest_Regressor

```
08]: from sklearn.ensemble import RandomForestRegressor

n_estimators = [50,100,200,250,300,350,400,500]
criterion2 = ['squared_error','friedman_mse','poisson']
max_ft2 = ['sqrt','log2']
randomforest_r2score= []

for estimators in n_estimators:
    for crt in criterion2:
        for max_frt in max_ft2:
            rff = RandomForestRegressor(n_estimators=estimators ,criterion=crt ,max_features=max_frt ,random_state=32)
            rff.fit(X_train,y_train)
            y_pred = rff.predict(X_test)
            rf_r2score = r2_score(y_test,y_pred)
            randomforest_r2score.append((estimators,crt,max_frt,rf_r2score))
```

```
09]: for n_estimators,criterion,max_features,score in randomforest_r2score:
    print("n_estimators:",n_estimators,"Criterion:",criterion,"max_features:",max_features,"R2_Score:",score)
```

n_estimators: 50 Criterion: squared_error max_features: sqrt R2_Score: 0.8502984279389878
n_estimators: 50 Criterion: squared_error max_features: log2 R2_Score: 0.8502984279389878
n_estimators: 50 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8509127600875582
n_estimators: 50 Criterion: friedman_mse max_features: log2 R2_Score: 0.8509127600875582
n_estimators: 50 Criterion: poisson max_features: sqrt R2_Score: 0.8468833336548729
n_estimators: 50 Criterion: poisson max_features: log2 R2_Score: 0.8468833336548729
n_estimators: 100 Criterion: squared_error max_features: sqrt R2_Score: 0.8492395564499937
n_estimators: 100 Criterion: squared_error max_features: log2 R2_Score: 0.8492395564499937
n_estimators: 100 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8492782583516395
n_estimators: 100 Criterion: friedman_mse max_features: log2 R2_Score: 0.8492782583516395
n_estimators: 100 Criterion: poisson max_features: sqrt R2_Score: 0.8474801406941771
n_estimators: 100 Criterion: poisson max_features: log2 R2_Score: 0.8474801406941771
n_estimators: 200 Criterion: squared_error max_features: sqrt R2_Score: 0.85125883509619
n_estimators: 200 Criterion: squared_error max_features: log2 R2_Score: 0.85125883509619
n_estimators: 200 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8513141835838371
n_estimators: 200 Criterion: friedman_mse max_features: log2 R2_Score: 0.8513141835838371
n_estimators: 200 Criterion: poisson max_features: sqrt R2_Score: 0.849928342290375
n_estimators: 200 Criterion: poisson max_features: log2 R2_Score: 0.849928342290375
n_estimators: 250 Criterion: squared_error max_features: sqrt R2_Score: 0.8509722090294829
n_estimators: 250 Criterion: squared_error max_features: log2 R2_Score: 0.8509722090294829
n_estimators: 250 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8510874568342656
n_estimators: 250 Criterion: friedman_mse max_features: log2 R2_Score: 0.8510874568342656
n_estimators: 250 Criterion: poisson max_features: sqrt R2_Score: 0.8500928704645765
n_estimators: 250 Criterion: poisson max_features: log2 R2_Score: 0.8500928704645765
n_estimators: 300 Criterion: squared_error max_features: sqrt R2_Score: 0.8515867676508716
n_estimators: 300 Criterion: squared_error max_features: log2 R2_Score: 0.8515867676508716
n_estimators: 300 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8516592052895238
n_estimators: 300 Criterion: friedman_mse max_features: log2 R2_Score: 0.8516592052895238
n_estimators: 300 Criterion: poisson max_features: sqrt R2_Score: 0.8509948045947353
n_estimators: 300 Criterion: poisson max_features: log2 R2_Score: 0.8509948045947353
n_estimators: 350 Criterion: squared_error max_features: sqrt R2_Score: 0.8510703568787773
n_estimators: 350 Criterion: squared_error max_features: log2 R2_Score: 0.8510703568787773
n_estimators: 350 Criterion: friedman_mse max_features: sqrt R2_Score: 0.851144935393941
n_estimators: 350 Criterion: friedman_mse max_features: log2 R2_Score: 0.851144935393941
n_estimators: 350 Criterion: poisson max_features: sqrt R2_Score: 0.8502215755711253
n_estimators: 350 Criterion: poisson max_features: log2 R2_Score: 0.8502215755711253
n_estimators: 400 Criterion: squared_error max_features: sqrt R2_Score: 0.8509950319542703
n_estimators: 400 Criterion: squared_error max_features: log2 R2_Score: 0.8509950319542703
n_estimators: 400 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8512436813151877
n_estimators: 400 Criterion: friedman_mse max_features: log2 R2_Score: 0.8512436813151877
n_estimators: 400 Criterion: poisson max_features: sqrt R2_Score: 0.8508266616553146
n_estimators: 400 Criterion: poisson max_features: log2 R2_Score: 0.8508266616553146
n_estimators: 500 Criterion: squared_error max_features: sqrt R2_Score: 0.8515876208578533

n_estimators: 500 Criterion: squared_error max_features: log2 R2_Score: 0.8515876208578533
n_estimators: 500 Criterion: friedman_mse max_features: sqrt R2_Score: 0.8515351540982423
n_estimators: 500 Criterion: friedman_mse max_features: log2 R2_Score: 0.8515351540982423
n_estimators: 500 Criterion: poisson max_features: sqrt R2_Score: 0.8518107418752442

Best Param with best R2_Score :

n_estimators: 500
Criterion: poisson
max_features: log2

R2_Score: 0.8518107418752442

Conclusion:

- Based on the **R^2 Score** values from the table, I conclude that the **Random Forest Regressor** is the best-performing model.
- Other models (MLR, SVM, Decision Tree) have lower R^2 scores compared to Random Forest model.

sl.no	Model_Name	R2_Score
1	MLR	0.77239
2	SVM_Regressor	0.84223
3	Decision Tree	0.77444
4	Random Forest	0.85181