## 1.Problem Statement:

The client wants to predict insurance charges based on several parameters such as age, gender, BMI, number of children, smoking habits. I have a dataset containing these variables, along with the corresponding insurance charges. As a data scientist, my task is to develop a machine learning model that accurately predicts the insurance charges for new data inputs based on these parameters.

## 2.Basic info of dataset :

Here there are 6 columns like age, gender, bmi, children, smoker, charges.

Here there are 1338 rows.

Here we give input as independent=dataset[['age','bmi', 'children','sex_male', 'smoker_yes']]

Here we give output as dependent=dataset[["charges"]]

## 4.According to my work done on various algorithm, I got best performance on random forest with 0.8710 r_score.

Based on the performance scores , the **best model** for this dataset is:

**Random Forest**

- **Criterion**: squared_error
- **Max features**: 'sqrt'
- **n_estimators**: 100
- **Performance Score**: 0.8710

This configuration gives the highest score and the best model compared with other algorithm.

**5.To find machine learning regression method using in r2 value.**

1. **MULTIPLE LINEAR REGRESSION: r_score : 0.7894790349867009**

2. **SUPPORT VECTOR MACHINE:**

| S.NO | HYPER PARAMETER | LINEAR (r_score) | RBF (r_score) | POLY (r_score) | SIGMOID (r_score) |
|---|---|---|---|---|---|
| 1. | C10 | -0.0403 | -0.09506 | -0.1202 | -0.0992 |
| 2. | C100 | 0.5218 | -0.09506 | -0.1319 | -0.1241 |
| 3. | C500 | 0.6133 | -0.15659 | -0.1166 | -0.4197 |
| 4. | C1000 | 0.6188 | -0.14956 | -0.0923 | -1.5217 |
| 5. | C2000 | 0.6253 | -0.14219 | -0.0423 | -5.0920 |
| 6. | C3000 | 0.6662 | -0.13060 | 0.0062 | -10.947 |

3. **DECISION TREE:**

| S.NO | CRITERION | MAX FEATURES | SPLITTER | R_VALUE |
|---|---|---|---|---|
| 1 | squared_error | Sqrt | Random | 0.7313 |
| 2 | squared_error | sqrt | Best | 0.6384 |
| 3 | squared_error | log2 | Best | 0.5672 |
| 4 | squared_error | log2 | Random | 0.6583 |
| 5 | friedman_mse | log2 | Random | 0.6913 |
| 6 | friedman_mse | log2 | Best | 0.7112 |
| 7 | friedman_mse | Sqrt | Best | 0.7152 |
| 8 | friedman_mse | Sqrt | Random | 0.6126 |
| 9 | absolute_error | Sqrt | Random | 0.7426 |
| 10 | absolute_error | Sqrt | Best | 0.7384 |
| 11 | absolute_error | log2 | Best | 0.7609 |
| 12 | absolute_error | log2 | Random | 0.7389 |
| 13 | Poisson | log2 | Random | 0.6561 |
| 14 | Poisson | log2 | Best | 0.7337 |
| 15 | Poisson | Sqrt | Best | 0.7337 |
| 16 | Poisson | sqrt | Random | 0.6697 |

### 4. RANDOM FOREST:

| S.NO | CRITERION | MAX FEATURES | N_ESTIMATORS | R_VALUE |
|---|---|---|---|---|
| **1** | absolute_error | sqrt | 10 | 0.8574 |
| **2** | absolute_error | Log2 | 10 | 0.8574 |
| **3** | absolute_error | Sqrt | 100 | 0.8710 |
| **4** | absolute_error | Log2 | 100 | 0.8710 |
| **5** | friedman_mse | Sqrt | 10 | 0.8502 |
| **6** | friedman_mse | Sqrt | 100 | 0.8710 |
| **7** | friedman_mse | Log2 | 10 | 0.8502 |
| **8** | friedman_mse | Log2 | 100 | 0.8710 |
| **9** | Poisson | Sqrt | 10 | 0.8544 |
| **10** | Poisson | Sqrt | 100 | 0.8680 |
| **11** | Poisson | Log2 | 10 | 0.8544 |
| **12** | Poisson | Log2 | 100 | 0.8680 |
| **13** | squared_error | Sqrt | 10 | 0.8520 |
| **14** | squared_error | Sqrt | 100 | 0.8710 |
| **15** | squared_error | Log2 | 10 | 0.8520 |
| **16** | squared_error | Log2 | 100 | 0.8710 |

The configurations with the highest performance (score: 0.8710) are:

1. absolute_error with sqrt, 100 estimators
2. absolute_error with log2, 100 estimators
3. friedman_mse with sqrt, 100 estimators
4. friedman_mse with log2, 100 estimators
5. squared_error with sqrt, 100 estimators
6. squared_error with log2, 100 estimators

All these configurations give the same highest performance of 0.8710.

### 6. FROM THE FOUR MODEL RANDOM FOREST GIVES US THE BEST R_SCORE COMPARED TO OTHERS.

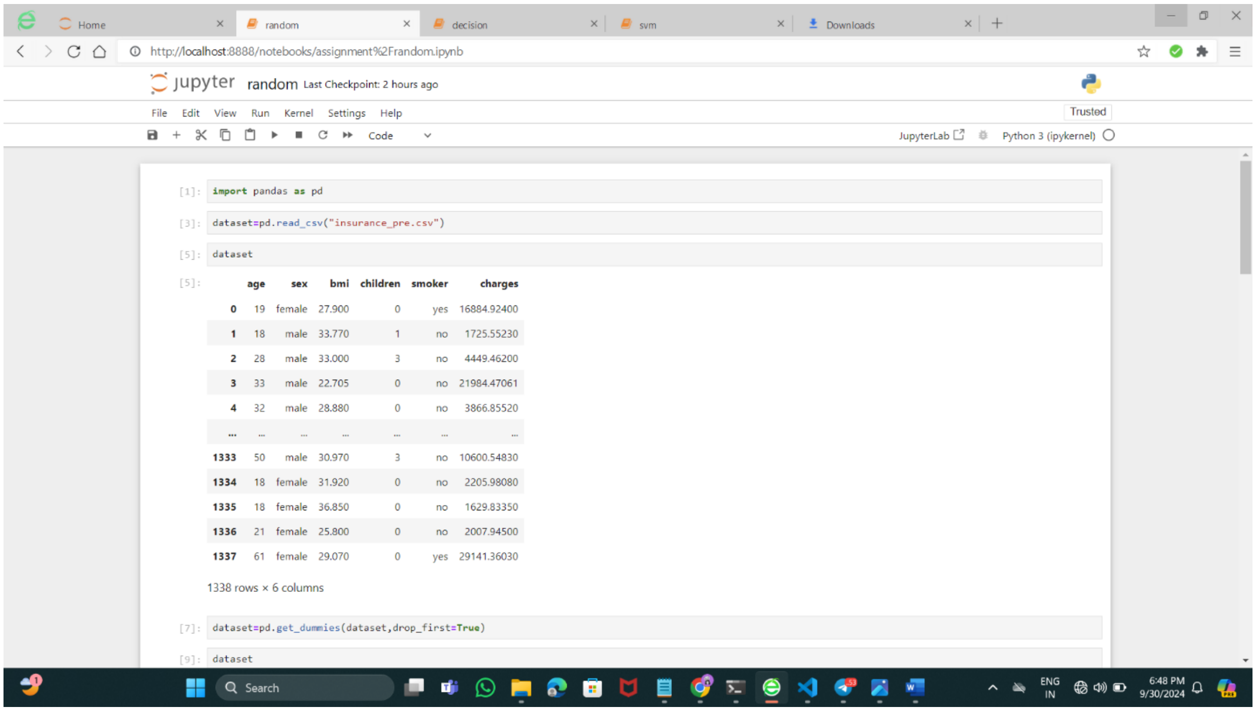
Random Forest is the best model because:

1. It has the highest score (0.8710).

2. It prevents overfitting by averaging many decision trees.

3. It works well with both simple and complex data.

In short, it’s accurate, reliable, and works for many situations.


Home
random
decision
svm
Downloads
http://localhost:8888/notebooks/assignment%2Frandom.ipynb
jupyter random Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Search
ENG
IN
6:48 PM
9/30/2024

```
[1]: import pandas as pd
```

```
[3]: dataset=pd.read_csv("insurance_pre.csv")
```

```
[5]: dataset
```

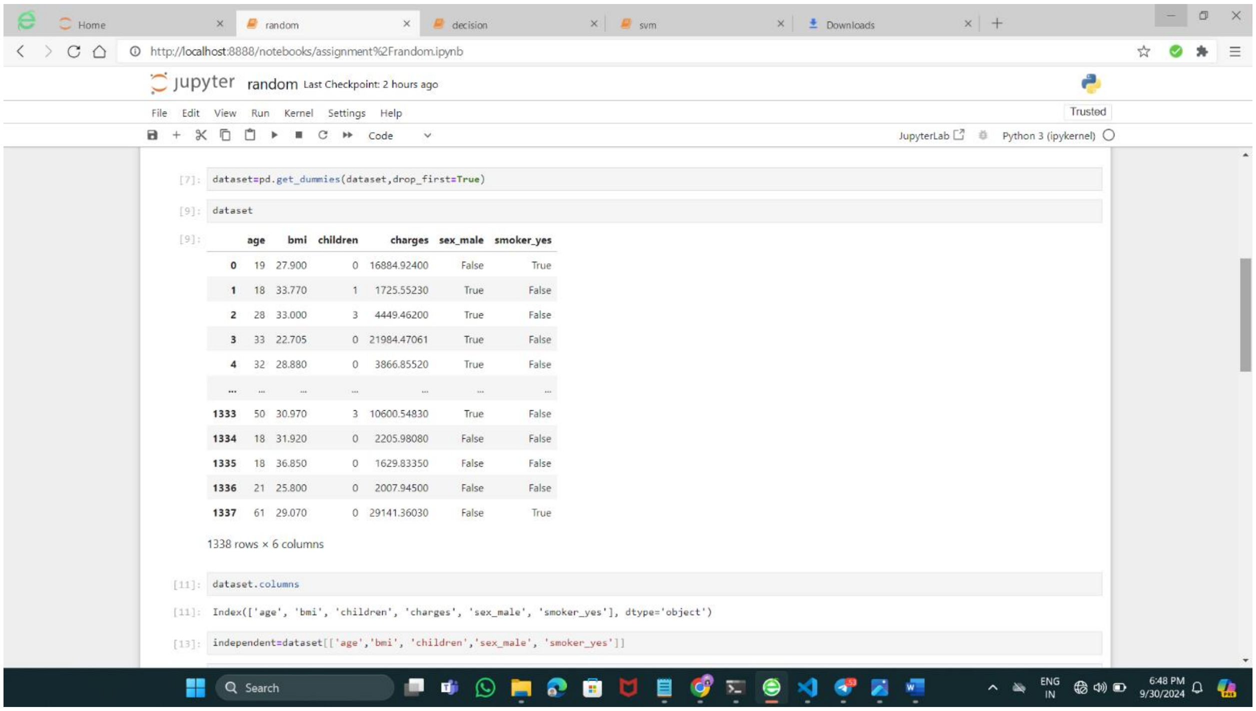
[5]:

| | age | sex | bmi | children | smoker | charges |
|---|---|---|---|---|---|---|
| **0** | 19 | female | 27.900 | 0 | yes | 16884.92400 |
| **1** | 18 | male | 33.770 | 1 | no | 1725.55230 |
| **2** | 28 | male | 33.000 | 3 | no | 4449.46200 |
| **3** | 33 | male | 22.705 | 0 | no | 21984.47061 |
| **4** | 32 | male | 28.880 | 0 | no | 3866.85520 |
| **...** | ... | ... | ... | ... | ... | ... |
| **1333** | 50 | male | 30.970 | 3 | no | 10600.54830 |
| **1334** | 18 | female | 31.920 | 0 | no | 2205.98080 |
| **1335** | 18 | female | 36.850 | 0 | no | 1629.83350 |
| **1336** | 21 | female | 25.800 | 0 | no | 2007.94500 |
| **1337** | 61 | female | 29.070 | 0 | yes | 29141.36030 |

1338 rows × 6 columns

```
[7]: dataset=pd.get_dummies(dataset,drop_first=True)
```

```
[9]: dataset
```


Home
random
decision
svm
Downloads
http://localhost:8888/notebooks/assignment%2Frandom.ipynb
jupyter random Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Search
ENG
IN
6:48 PM
9/30/2024

```
[7]: dataset=pd.get_dummies(dataset,drop_first=True)
```

```
[9]: dataset
```

[9]:

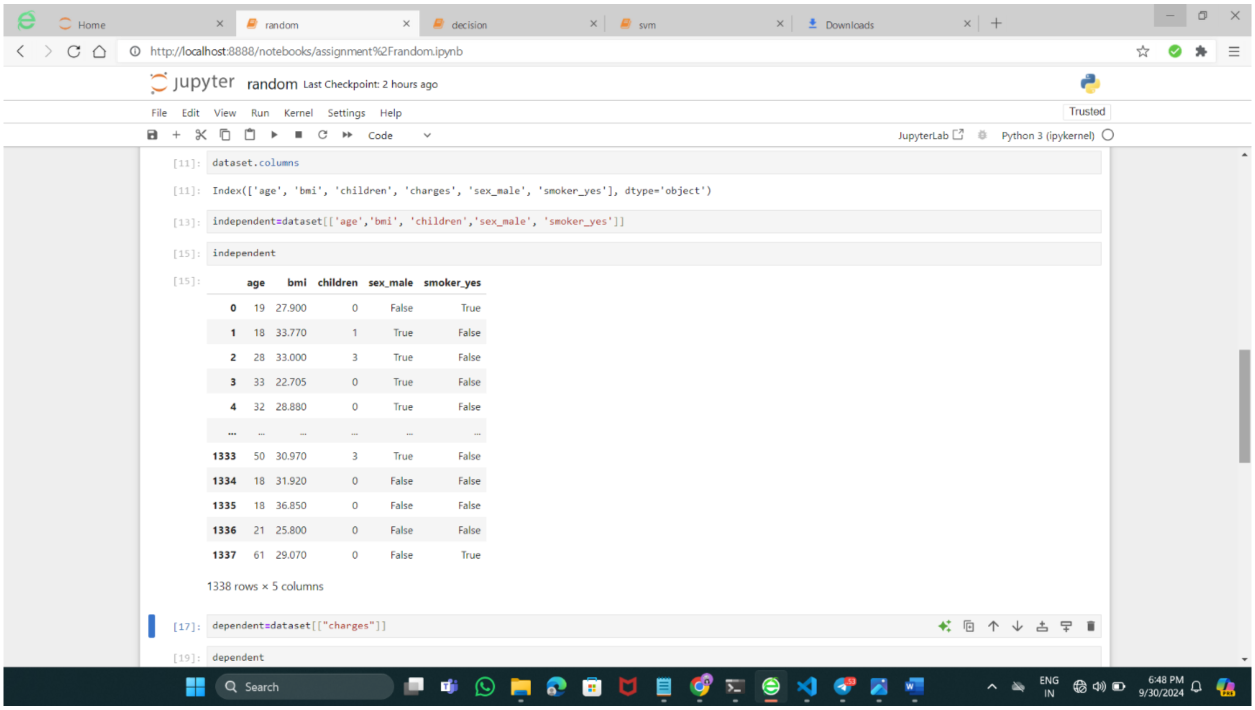
| | age | bmi | children | charges | sex_male | smoker_yes |
|---|---|---|---|---|---|---|
| **0** | 19 | 27.900 | 0 | 16884.92400 | False | True |
| **1** | 18 | 33.770 | 1 | 1725.55230 | True | False |
| **2** | 28 | 33.000 | 3 | 4449.46200 | True | False |
| **3** | 33 | 22.705 | 0 | 21984.47061 |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

1338 rows × 6 columns

```
[11]: dataset.columns
```

```
[11]: Index(['age', 'bmi', 'children', 'charges', 'sex_male', 'smoker_yes'], dtype='object')
```

```
[13]: independent=dataset[['age','bmi', 'children','sex_male', 'smoker_yes']]
```


Home
random
decision
svm
Downloads
http://localhost:8888/notebooks/assignment%2Frandom.ipynb
jupyter random Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Search
ENG IN
6:48 PM 9/30/2024

```
[11]: dataset.columns

[11]: Index(['age', 'bmi', 'children', 'charges', 'sex_male', 'smoker_yes'], dtype='object')

[13]: independent=dataset[['age','bmi', 'children','sex_male', 'smoker_yes']]

[15]: independent
```

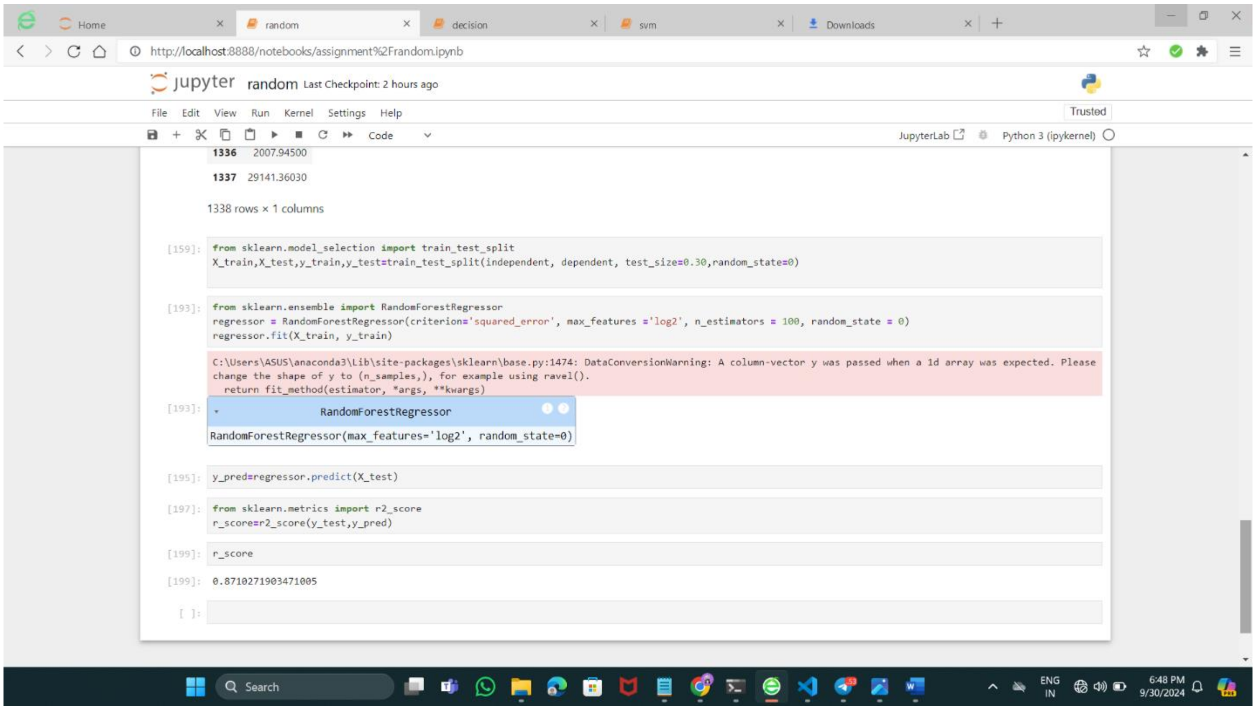
[15]:

|  | age | bmi | children | sex_male | smoker_yes |
|---|---|---|---|---|---|
| **0** | 19 | 27.900 | 0 | False | True |
| **1** | 18 | 33.770 | 1 | True | False |
| **2** | 28 | 33.000 | 3 | True | False |
| **3** | 33 | 22.705 | 0 | True | False |
| **4** | 32 | 28.880 | 0 | True | False |
| **...** | ... | ... | ... | ... | ... |
| **1333** | 50 | 30.970 | 3 | True | False |
| **1334** | 18 | 31.920 | 0 | False | False |
| **1335** | 18 | 36.850 | 0 | False | False |
| **1336** | 21 | 25.800 | 0 | False | False |
| **1337** | 61 | 29.070 | 0 | False | True |

1338 rows × 5 columns

```
[17]: dependent=dataset[["charges"]]

[19]: dependent
```


Home
random
decision
svm
Downloads
http://localhost:8888/notebooks/assignment%2Frandom.ipynb
jupyter random Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Search
ENG IN
6:48 PM 9/30/2024

|  |  |
|---|---|
| **1336** | 2007.94500 |
| **1337** | 29141.36030 |

1338 rows × 1 columns

```
[159]: from sklearn.model_selection import train_test_split
       X_train,X_test,y_train,y_test=train_test_split(independent, dependent, test_size=0.30,random_state=0)

[193]: from sklearn.ensemble import RandomForestRegressor
       regressor = RandomForestRegressor(criterion='squared_error', max_features ='log2', n_estimators = 100, random_state = 0)
       regressor.fit(X_train, y_train)

       C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\base.py:1474: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
       change the shape of y to (n_samples,), for example using ravel().
         return fit_method(estimator, *args, **kwargs)

[193]: RandomForestRegressor
       RandomForestRegressor(max_features='log2', random_state=0)

[195]: y_pred=regressor.predict(X_test)

[197]: from sklearn.metrics import r2_score
       r_score=r2_score(y_test,y_pred)

[199]: r_score

[199]: 0.8710271903471005

[ ]:
```