48. Hyperparameter Tuning (Practical)

In [10]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [11]:
```
dataset = pd.read_csv(r'Data/level_salaries.csv')
dataset.head(3)
```

Out[11]:

	Level	Salaries
0	1.000000	55167.141530
1	1.019019	48825.036941
2	1.038038	56692.389975

In [ ]:
```

```

In [12]:
```
x = dataset.iloc[:,:-1]
y = dataset['Salaries']
```

In [ ]:
```

```

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

In [ ]:
```

```

In [15]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [16]:
```
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
```

Out[16]:
```
▾ DecisionTreeRegressor

DecisionTreeRegressor()
```

In [ ]:
```

```

In [17]:
```
dt.score(x_test, y_test)*100
```

Out[17]:
```
73.22053360458676
```

In [18]:
```
dt.score(x_train, y_train)*100
```

Out[18]: 100.0

- Model is over-fitting

48.1 Perform Hyperparameters Tuning to reduce over-fitting

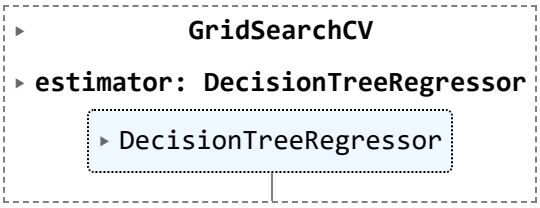
48.1.1 Tuning by GridSearchCV

In [19]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [24]:
```
df = {
    "criterion":["squared_error", "friedman_mse", "absolute_error","poisson"],
    "splitter":["best", "random"],
    "max_depth":[i for i in range(2,20)]
     }
```

In [28]:
```
gd = GridSearchCV(DecisionTreeRegressor(), param_grid=df)
gd.fit(x_train, y_train)
```

Out[28]:



In [29]:
```
gd.best_params_
```

Out[29]: {'criterion': 'squared_error', 'max_depth': 4, 'splitter': 'best'}

In [33]:
```
gd.best_score_
```

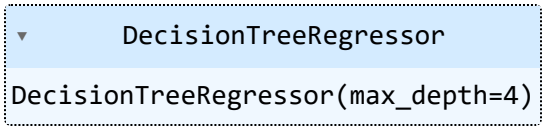
Out[33]: 0.8393136355736118

In [ ]:
```

```

In [30]:
```
dt2 = DecisionTreeRegressor(criterion='squared_error', max_depth=4, splitter='best'
dt2.fit(x_train, y_train)
```

Out[30]:



In [32]:
```
dt.score(x_test, y_test)*100, dt.score(x_train, y_train)*100
```

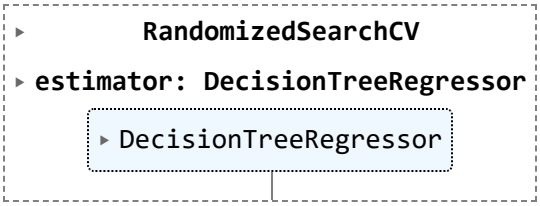
Out[32]: (73.22053360458676, 100.0)

In [ ]:

48.1.2 Tuning by RandomizedSearchCV

In [35]:
```
rd = RandomizedSearchCV(DecisionTreeRegressor(), param_distributions=df, n_iter=20)
rd.fit(x_train, y_train)
```

Out[35]:



In [ ]:

In [37]:
```
rd.score(x_test, y_test)*100, rd.score(x_train, y_train)*100
```

Out[37]:
```
(85.14998219015995, 86.78684301893401)
```

Over-Fitting is reduced significantly in this case

In [38]:
```
rd.best_params_
```

Out[38]:
```
{'splitter': 'best', 'max_depth': 4, 'criterion': 'squared_error'}
```

In [39]:
```
rd.best_score_
```

Out[39]:
```
0.8393136355736118
```

In [ ]: