In [16]:
```
import sqlite3
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [17]:
```
# Create your connection.
cnx = sqlite3.connect('database.sqlite')
df = pd.read_sql_query("SELECT * FROM Player_Attributes", cnx)
```

In [18]:
```
df.head()
```

Out[18]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attacking_work_rate	defensive_work_rate	crossing	...	visior
0	1	218353	505942	2016-02-18 00:00:00	67.0	71.0	right	medium	medium	49.0	...	54.(
1	2	218353	505942	2015-11-19 00:00:00	67.0	71.0	right	medium	medium	49.0	...	54.(
2	3	218353	505942	2015-09-21 00:00:00	62.0	66.0	right	medium	medium	49.0	...	54.(
3	4	218353	505942	2015-03-20 00:00:00	61.0	65.0	right	medium	medium	48.0	...	53.(
4	5	218353	505942	2007-02-22 00:00:00	61.0	65.0	right	medium	medium	48.0	...	53.(

5 rows × 42 columns

In [19]:
```
df.shape
```

Out[19]:
```
(183978, 42)
```

df.columns

Declare the Columns You Want to Use as Features

In [20]:
```
features = [
       'potential', 'crossing', 'finishing', 'heading_accuracy',
       'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy',
       'long_passing', 'ball_control', 'acceleration', 'sprint_speed',
       'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'stamina',
       'strength', 'long_shots', 'aggression', 'interceptions', 'positioning',
       'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle',
       'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning',
       'gk_reflexes']
```

Specify the Prediction Target

In [21]:
```
target = ['overall_rating']
```

Clean the Data

In [22]:
```
df = df.dropna()
```

Extract Features and Target ('overall_rating') Values into Separate Dataframes

In [23]:
```
X = df[features]
```

In [24]:
```
y = df[target]
```

Let us look at a typical row from our features:

In [25]:
```
X.iloc[1]
```

Out[25]:
```
potential             71.0
crossing              49.0
finishing             44.0
heading_accuracy      71.0
short_passing         61.0
volleys               44.0
dribbling             51.0
curve                 45.0
free_kick_accuracy    39.0
long_passing          64.0
ball_control          49.0
acceleration          60.0
sprint_speed          64.0
agility               59.0
reactions             47.0
balance               65.0
shot_power            55.0
jumping               58.0
stamina               54.0
strength              76.0
long_shots            35.0
aggression            71.0
interceptions         70.0
positioning           45.0
vision                54.0
penalties             48.0
marking               65.0
standing_tackle       69.0
sliding_tackle        69.0
gk_diving              6.0
gk_handling           11.0
gk_kicking            10.0
gk_positioning         8.0
gk_reflexes            8.0
Name: 1, dtype: float64
```

Let us also display our target values:

In [26]:
```
y
```

Out[26]:

	overall_rating
0	67.0
1	67.0
2	62.0
3	61.0
4	61.0
5	74.0
6	74.0
7	73.0
8	73.0
9	73.0
10	73.0
11	74.0
12	73.0
13	71.0
14	71.0
15	71.0
16	70.0
17	70.0
18	70.0
19	70.0
20	70.0
21	70.0
22	69.0
23	69.0
24	69.0
25	69.0
26	69.0

Split the Dataset into Training and Test Datasets

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=324)
```

(1) Linear Regression: Fit a model to the training set

In [29]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[29]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Perform Prediction using Linear Regression Model

In [30]:
```
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[30]:
```
array([[ 66.51284879],
       [ 79.77234615],
       [ 66.57371825],
       ...,
       [ 69.23780133],
       [ 64.58351696],
       [ 73.6881185 ]])
```

What is the mean of the expected target value in test set ?

In [31]:
```
y_test.describe()
```

Out[31]:

	overall_rating
count	59517.000000
mean	68.635818
std	7.041297
min	33.000000
25%	64.000000
50%	69.000000
75%	73.000000
max	94.000000

Evaluate Linear Regression Accuracy using Root Mean Square Error

In [32]:
```
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
```

In [34]:
```
print(RMSE)
```

```
2.8053030468552103
```

(2) Decision Tree Regressor: Fit a new regression model to the training set

In [35]:
```
regressor = DecisionTreeRegressor(max_depth=20)
regressor.fit(X_train, y_train)
```

Out[35]:
```
DecisionTreeRegressor(criterion='mse', max_depth=20, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

Perform Prediction using Decision Tree Regressor

In [36]:
```
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[36]:
```
array([ 62.        ,  84.        ,  62.38666667, ...,  71.        ,
        62.        ,  73.        ])
```

For comparision: What is the mean of the expected target value in test set ?

In [37]:
```
y_test.describe()
```

Out[37]:

	overall_rating
count	59517.000000
mean	68.635818
std	7.041297
min	33.000000
25%	64.000000
50%	69.000000
75%	73.000000
max	94.000000

Evaluate Decision Tree Regression Accuracy using Root Mean Square Error

In [38]:
```
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
```

In [39]:
```
print(RMSE)
```

```
1.4477824494932103
```