Regresssion with scikit-learn using Soccer Dataset

We will again be using the open dataset from the popular site [Kaggle (https://www.kaggle.com)](https://www.kaggle.com) that we used in Week 1 for our example.

Recall that this [European Soccer Database (https://www.kaggle.com/hugomathien/soccer)](https://www.kaggle.com/hugomathien/soccer) has more than 25,000 matches and more than 10,000 players for European professional soccer seasons from 2008 to 2016.

Note: Please download the file *database.sqlite* if you don't yet have it in your *Week-7-MachineLearning* folder.

Import Libraries

In [1]:

```
import sqlite3
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from math import sqrt
```

Read Data from the Database into pandas

In [2]:

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

In [3]:

```
df.head()
```

Out[3]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attackin
0	1	218353	505942	2016-02-18 00:00:00	67.0	71.0	right	
1	2	218353	505942	2015-11-19 00:00:00	67.0	71.0	right	
2	3	218353	505942	2015-09-21 00:00:00	62.0	66.0	right	
3	4	218353	505942	2015-03-20 00:00:00	61.0	65.0	right	
4	5	218353	505942	2007-02-22 00:00:00	61.0	65.0	right	

5 rows × 42 columns

In [4]:

```
df.shape
```

Out[4]:

```
(183978, 42)
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['id', 'player_fifa_api_id', 'player_api_id', 'date', 'overall_r
ating',
       'potential', 'preferred_foot', 'attacking_work_rate',
       'defensive_work_rate', 'crossing', 'finishing', 'heading_accura
cy',
       'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_ac
curacy',
       'long_passing', 'ball_control', 'acceleration', 'sprint_speed',
       'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'st
amina',
       'strength', 'long_shots', 'aggression', 'interceptions', 'posit
ioning',
       'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_t
ackle',
       'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning',
       'gk_reflexes'],
      dtype='object')
```

Declare the Columns You Want to Use as Features

In [6]:

```
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 [7]:

```
target = ['overall_rating']
```

Clean the Data

In [8]:

```
df = df.dropna()
```

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

In [9]:

```
X = df[features]
```

In [10]:

```
y = df[target]
```

Let us look at a typical row from our features:

In [11]:

```
X.iloc[2]
```

Out[11]:

```
potential              66.0
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             63.0
interceptions          41.0
positioning            45.0
vision                 54.0
penalties              48.0
marking                65.0
standing_tackle        66.0
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: 2, dtype: float64
```

Let us also display our target values:

In [12]:

```
y
```

Out[12]:

	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
27	69.0
28	69.0
29	68.0
...	...
183933	76.0
183934	75.0

	overall_rating
183935	77.0
183936	77.0
183937	63.0
183938	63.0
183939	63.0
183940	63.0
183941	63.0
183942	66.0
183943	66.0
183944	66.0
183945	66.0
183946	66.0
183947	68.0
183948	68.0
183949	68.0
183950	68.0
183951	67.0
183952	67.0
183968	78.0
183969	81.0
183970	81.0
183971	81.0
183972	83.0
183973	83.0
183974	78.0
183975	77.0
183976	78.0
183977	80.0

180354 rows × 1 columns

Split the Dataset into Training and Test Datasets

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_s
```

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

In [14]:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[14]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=
False)
```

Perform Prediction using Linear Regression Model

In [15]:

```
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[15]:

```
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 [16]:

```
y_test.describe()
```

Out[16]:

	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 [17]:

```
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE)
```

```
2.8053030468552085
```

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

In [18]:

```
regressor = DecisionTreeRegressor(max_depth = 20)
regressor.fit(X_train, y_train)
```

Out[18]:

```
DecisionTreeRegressor(criterion='mse', max_depth=20, max_features=Non
e,
           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 [19]:

```
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[19]:

```
array([62.        , 84.        , 62.38666667, ..., 71.        ,
       62.        , 72.        ])
```

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

In [20]:

```
y_test.describe()
```

Out[20]:

	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 [21]:

```
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
```

In [22]:

```
print(RMSE)
```

```
1.4667473787036156
```

As as result, the Decision Tree Regression algorithm has a better prediction accuracy than Linear Regression

In [ ]:

```

```