Centurylink Interview Project

James Nelson

Contents

1. Exploratory Data analysis

The libraries I will be using for this analysis are as follows...

- Pandas - A Python library used to manage and manipulate data in dataframes.
- Numpy - A Python library used to manipulate data into vectors/arrays.
- Seabborn - A Python library used to visualize data.
- PyPlot - A Python library used to visualize data.

First step in my EDA is to load the data into a pandas dataframe and then to view the head to see what it looks like.

```
[2] df = pd.read_excel("/content/drive/My Drive/Centurylink_project/FIFAData_90_percent.xlsx")
    df.head()
```

	sofifa_id	age	dob	height_cm	weight_kg	nationality	club	overall	potential	value_eur	wage_eur	preferred_foot	international_reputation	weak_foo
0	101317	37	1982-04-16	189	85	Germany	SC Paderborn 07	63	63	40000	2000	Right	1	
1	101488	35	1983-08-01	188	84	Germany	FC Erzgebirge Aue	64	64	120000	1000	Right	1	
2	102064	35	1984-06-19	180	69	Italy	Spezia	72	72	950000	3000	Right	1	
3	102593	36	1982-12-22	175	68	England	Exeter City	62	62	60000	2000	Left	1	
4	102881	39	1979-08-28	188	80	Canada	AIK	57	57	10000	1000	Left	1	

So far from just looking at the head the data looks alright with only two NaNs. To get a closer look I will call the shape and info functions for the dataframe.

```
[3] df.shape
```

```
(15854, 58)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15854 entries, 0 to 15853
Data columns (total 58 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   sofifa_id                  15854 non-null  int64
 1   age                        15854 non-null  int64
 2   dob                        15854 non-null  datetime64[ns]
 3   height_cm                  15854 non-null  int64
 4   weight_kg                  15854 non-null  int64
 5   nationality                15854 non-null  object
 6   club                       15854 non-null  object
 7   overall                    15854 non-null  int64
 8   potential                  15854 non-null  int64
 9   value_eur                  15854 non-null  int64
 10  wage_eur                   15854 non-null  int64
 11  preferred_foot             15854 non-null  object
```

```
 12  international_reputation    15854 non-null  int64
 13  weak_foot                   15854 non-null  int64
 14  skill_moves                 15854 non-null  int64
 15  work_rate                   15854 non-null  object
 16  body_type                   15854 non-null  object
 17  real_face                   15854 non-null  object
 18  release_clause_eur          15854 non-null  int64
 19  team_position               15641 non-null  object
 20  joined                      14744 non-null  object
 21  contract_valid_until        15641 non-null  float64
 22  pace                        15854 non-null  int64
 23  shooting                    15854 non-null  int64
 24  passing                     15854 non-null  int64
 25  dribbling                   15854 non-null  int64
 26  defending                   15854 non-null  int64
 27  physic                      15854 non-null  int64
 28  player_traits               6456 non-null   object
 29  attacking_crossing          15854 non-null  int64
 30  attacking_finishing         15854 non-null  int64
 31  attacking_heading_accuracy  15854 non-null  int64
 32  attacking_short_passing     15854 non-null  int64
 33  attacking_volleys           15854 non-null  int64
 34  skill_dribbling             15854 non-null  int64
 35  skill_curve                 15854 non-null  int64
 36  skill_fk_accuracy           15854 non-null  int64
 37  skill_long_passing          15854 non-null  int64
 38  skill_ball_control          15854 non-null  int64
 39  movement_acceleration       15854 non-null  int64
 40  movement_sprint_speed       15854 non-null  int64
```

```
 41  movement_agility            15854 non-null  int64
 42  movement_reactions          15854 non-null  int64
 43  movement_balance            15854 non-null  int64
 44  power_shot_power            15854 non-null  int64
 45  power_jumping               15854 non-null  int64
 46  power_stamina               15854 non-null  int64
 47  power_strength              15854 non-null  int64
 48  power_long_shots            15854 non-null  int64
 49  mentality_aggression        15854 non-null  int64
 50  mentality_interceptions     15854 non-null  int64
 51  mentality_positioning       15854 non-null  int64
 52  mentality_vision            15854 non-null  int64
 53  mentality_penalties         15854 non-null  int64
 54  mentality_composure         15854 non-null  int64
 55  defending_marking           15854 non-null  int64
 56  defending_standing_tackle   15854 non-null  int64
 57  defending_sliding_tackle    15854 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(47), object(9)
memory usage: 7.0+ MB
```

There are 58 features with 15,854 samples of data with 47 of them being of int64 data type, 9 of them an object data type, and one feature is a float64 data type. The features that stand out is "team_position", "joined", "contract_valid_until", "player_traits". All features have a lesser number of samples than the total population of that shape.

Looking back at the df.head() the "joined" feature would indicate when the player joined, and the "player_traits" feature appears to be categorical traits of the player's greatest attributes. I will use df.isnull() to verify the null values if any for either.

```
print(df.isnull().sum())
```

```
sofifa_id                     0
age                           0
dob                           0
height_cm                     0
weight_kg                     0
nationality                   0
club                          0
overall                       0
potential                     0
value_eur                     0
wage_eur                      0
preferred_foot                0
international_reputation      0
weak_foot                     0
skill_moves                   0
work_rate                     0
body_type                     0
real_face                     0
release_clause_eur            0
team_position               213
joined                     1110
contract_valid_until        213
pace                          0
shooting                      0
passing                       0
dribbling                     0
```

```
defending                       0
physic                          0
player_traits                9398
attacking_crossing              0
attacking_finishing             0
attacking_heading_accuracy      0
attacking_short_passing         0
attacking_volleys               0
skill_dribbling                 0
skill_curve                     0
skill_fk_accuracy               0
skill_long_passing              0
skill_ball_control              0
movement_acceleration           0
movement_sprint_speed           0
movement_agility                0
movement_reactions              0
movement_balance                0
power_shot_power                0
power_jumping                   0
power_stamina                   0
power_strength                  0
power_long_shots                0
mentality_aggression            0
mentality_interceptions         0
mentality_positioning           0
mentality_vision                0
mentality_penalties             0
mentality_composure             0
```

```
defending_marking               0
defending_standing_tackle       0
defending_sliding_tackle        0
dtype: int64
```

The features "team_position" and "contract_valid_until" have correlated data errors with one another given they have 213 values each. When I apply a filter to even one of the features and reveal the blank values. The question I have would be what do the values of these player's contracts and entry dates into FIFA have to do with their stats, but their positions might have something to add to them.

Based on the choices given, I have decided to compare Japan's players to the rest of all the other players. What I decided to compare is the player's "overall" feature of Japan's players and how they stack up against the other players. I will use a basic statistics analysis to show this.

```
[6] dfjapan = df.loc[df['nationality'] == "Japan"] # creates a dataframe for only players that contain the value Japan in the feature "nationality".
```

```
dfjapan.head()
```

	sofifa_id	age	dob	height_cm	weight_kg	nationality	club	overall	potential	value_eur	wage_eur	preferred_foot	international_reputation	weak_f
121	140181	39	1979-08-02	187	80	Japan	Kashima Antlers	66	66	70000	1000	Right	1	
122	140196	39	1980-01-28	178	75	Japan	Gamba Osaka	72	72	775000	7000	Right	3	
123	140201	37	1982-06-09	170	73	Japan	JÃºbilo Iwata	69	69	375000	3000	Right	2	
284	156478	37	1981-09-06	178	77	Japan	Urawa Red Diamonds	65	65	110000	2000	Right	1	
378	162388	34	1985-06-21	183	77	Japan	Nagoya Grampus	64	64	170000	2000	Right	1	

```
dfall = df[df.nationality != "Japan"] # This removes all samples that contain the value "Japan" in the feature "nationality".
```

```
[14] dfjapan['overall'].describe()
```

```
count    398.000000
mean      63.658291
std        5.482173
min       48.000000
25%       60.000000
50%       64.000000
75%       67.000000
max       80.000000
Name: overall, dtype: float64
```

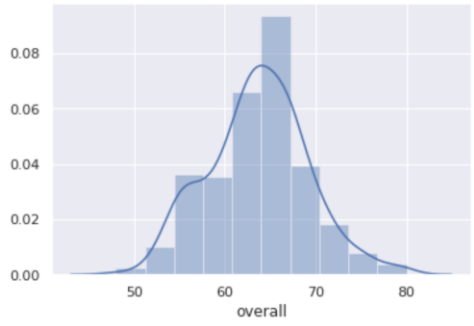
```
[15] dfall['overall'].describe()
```

```
count    15456.000000
mean        66.161426
std          6.962767
min         48.000000
25%         62.000000
50%         66.000000
75%         71.000000
max         94.000000
Name: overall, dtype: float64
```

Japan's players average out at having an overall score of 63 .66 while the rest of the world's players average out at 66.16. Given this data if I were to play the game FIFA right now, I would not choose Japan based off their "overall" stats. I want to visualize the distribution of this though in a histogram for both though.

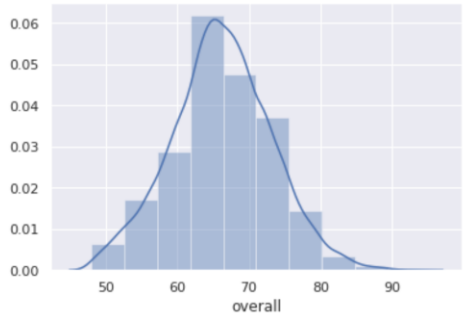
```
sns.distplot(dfjapan['overall'], bins=10) # distribution visualization for Japan player's "overall" stats.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8d683bcc0>
```



```
sns.distplot(dfall['overall'], bins=10) # disgtribution plot for all player's (excluding Japan) overall stats.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8d6708940>
```



Interesting enough Japan has a higher looking distribution between 60 and 70 leaning closer to 70 than the rest of the world does which might cause me to rethink my strategy on picking my teams. Let me get them side by side though.

```
x1 = dfjapan['overall'].tolist() # places all "overall" stat values for Japan players into a list.
x2 = dfall['overall'].tolist() # places all "overall" stat values for all players( excluding Japan) into a list.

colors = ['#E69F00', '#56B4E9'] # Colors orange and blue.
names = ['Japan Players', 'All Players (excluding Japan)']

plt.hist([x1, x2], bins = int(180/15),
         color = colors, label=names)

# Plot formatting
plt.legend()
plt.xlabel('Delay (min)')
plt.ylabel('Normalized Overall stats')
plt.title('Side-by-Side Histogram Japan Player stats vs All player stats')
```

Obviously, the distribution is too small to see for Japan players, but this distribution shows that Japan has a higher frequency of players with overall stats next to the highest frequency of values with the rest of the players of the world. However, I will do a value_counts() to determine how many players Japan has between 60 and 70 for their feature in "overall" stats.

```
[28] ranges = [60,70]
     japan_range = dfjapan['overall']
     japan_range.groupby(pd.cut(dfjapan.overall,ranges)).count()
```

```
overall
(60, 70]    253
Name: overall, dtype: int64
```

```
dfjapan.shape
```

```
(398, 58)
```

```
dfjapan['overall'].value_counts(sort=False)
```

```
48     1
51     2
53     8
54     5
55    20
56    15
57    11
58    11
59    14
60    20
61    18
62    29
63    37
64    32
65    25
66    32
67    30
68    22
69    16
70    12
71    10
72     7
73     6
74     1
75     6
76     3
78     1
79     3
80     1
Name: overall, dtype: int64
```

```
[30] all_range = dfall['overall']
     all_range.groupby(pd.cut(dfall.overall,ranges)).count()
```

```
overall
(60, 70]    8311
Name: overall, dtype: int64
```

```
[31] dfall.shape
```

```
(15456, 58)
```

With 253 players out of 398 total from Japan's FIFA game team falling between 60 and 70 on overall stats Japan has 63.57% of their players competing at those overall stat levels vs the 8311 out of 15456 players from the rest of the world leaving 53.77% of those players competing with those overall stats at those ranges.

2. Ordinary Least Squares Model

The libraries I used for this section were...

- Sklearn – SelectKBest, chi2, DBSCAN, IsolationForest, train_test_split, mean_squared_error, r2_score, mean_absolute_error
- statsmodels – variance_inflation_factor, sm
- random

For the OLS model since ethe target variable is "overall", I think it will be a good idea to visualize correlation in order to see what variables might be correlated to that feature. Before I do that, I will do just a regular pandas .corr function call with relevant features above .5.

```
df_cor = df.corr()['overall']# Get correlation with overall feature.

relevant_features_pos = df_cor[df_cor>0.5]# Selecting highly correlated features with "overall" and display.
relevant_features_pos
```

```
overall                   1.000000
potential                 0.638494
value_eur                 0.635413
wage_eur                  0.572631
release_clause_eur        0.605457
attacking_short_passing   0.599373
skill_long_passing        0.557882
skill_ball_control        0.562244
movement_reactions        0.865415
power_shot_power          0.589507
mentality_vision          0.539806
mentality_composure       0.759325
Name: overall, dtype: float64
```

This gives a general idea for me on what variables I should use for the OLS model to predict the target feature "overall". I want to see what other variables might be correlates with each other to avoid correlations so I will now visualize to get a general look at significant correlated variables greater than .5.

```
# Display significant correlation in plot heatmap
corr = df[['overall', 'potential', 'value_eur', 'wage_eur', 'release_clause_eur',
           'attacking_short_passing', 'skill_long_passing', 'skill_ball_control',
           'movement_reactions', 'power_shot_power', 'mentality_vision',
           'mentality_composure']].corr()
plt.figure(figsize=(8, 12))
sns.heatmap(corr[(corr >= 0.5) ],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size":8}, square=True);
```

I have a feeling my predictor features will be "movement_reactions", "attacking_short_passing", "value_eur", and "potential". To be sure there is no multicollinearity between them I will run VIF to figure this out.

I will also do a chi-square test for feature importance also.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
df_X =  df[['potential', 'value_eur', 'wage_eur', 'release_clause_eur',
           'attacking_short_passing', 'skill_long_passing', 'skill_ball_control',
           'movement_reactions', 'power_shot_power', 'mentality_vision',
           'mentality_composure']]
df_y = df['overall']
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(df_X,df_y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(df_X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
                   Specs         Score
3       release_clause_eur  3.510898e+11
1                value_eur  1.782745e+11
2                 wage_eur  5.972576e+08
10     mentality_composure  2.146199e+04
6       skill_ball_control  2.002484e+04
5       skill_long_passing  1.977933e+04
4   attacking_short_passing  1.788341e+04
8         power_shot_power  1.730032e+04
9         mentality_vision  1.637280e+04
7       movement_reactions  1.615469e+04
```

This chi2 test indicates that the most important feature is "wage_eur". However, I am still going to be using the feature "movement_reactions" as my most important feature due to its high correlation.

Now onto the VIF test for multicollinearity.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)

df_multi =  df[['potential', 'value_eur', 'wage_eur', 'release_clause_eur',
           'attacking_short_passing', 'skill_long_passing', 'skill_ball_control',
           'movement_reactions', 'power_shot_power', 'mentality_vision',
           'mentality_composure']]
calc_vif(df_multi)
```

	variables	VIF
0	potential	70.601405
1	value_eur	31.996605
2	wage_eur	4.599042
3	release_clause_eur	28.494699
4	attacking_short_passing	234.500044
5	skill_long_passing	75.748387
6	skill_ball_control	113.548277
7	movement_reactions	113.099499
8	power_shot_power	42.682411
9	mentality_vision	47.416040
10	mentality_composure	89.224045

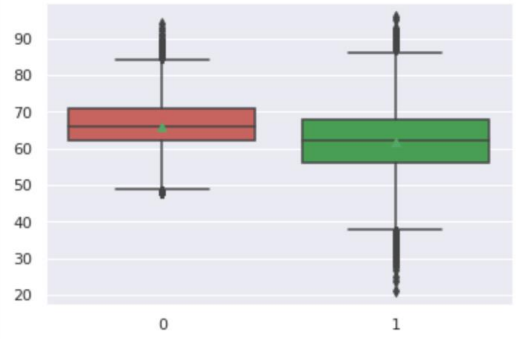
The idea behind a VIF test is to find the features giving the highest VIF score and eliminate them as predictor features for the OLS model because of the high multicollinearity. So, I will eliminate all the ones I was not concerned about anyways and then test.

So, the removal of all those other variables did decrease the VIF scores of my main predictor feature as well as the others, but there still may be high multicollinearity for this model. However, I am curious to see how they perform regardless.

Before I go create the model I will go explore "movement_reactions" and "mentality_composure" features a little more closely to see if they need to be cleaned up a bit so that they are properly engineered for the model, but looking back at the EDA I did in the beginning these two don't have any missing values so I think the only question would be to see if there are any outliers and for this I will do a regular side by side boxplot of the two just to visually see quickly if there are any.

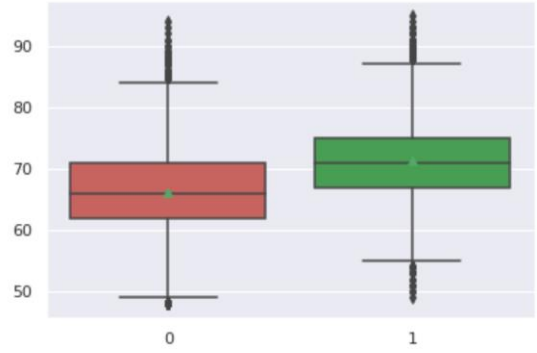
```
sns.boxplot(
    data=[df_over_move_at_po_wa['overall'],
          df_over_move_at_po_wa['movement_reactions'],
          ],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2cc72b748>
```



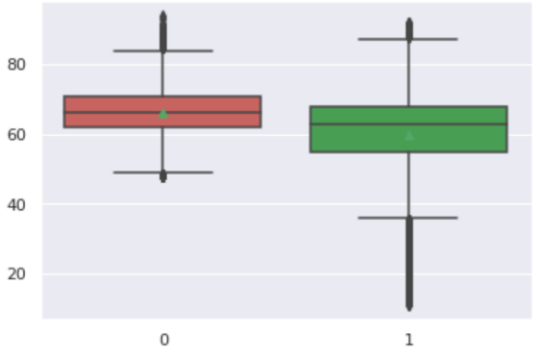

```
sns.boxplot(
    data=[df_over_move_at_po_wa['overall'],
          df_over_move_at_po_wa['potential'],
          ],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2cc6ab198>
```



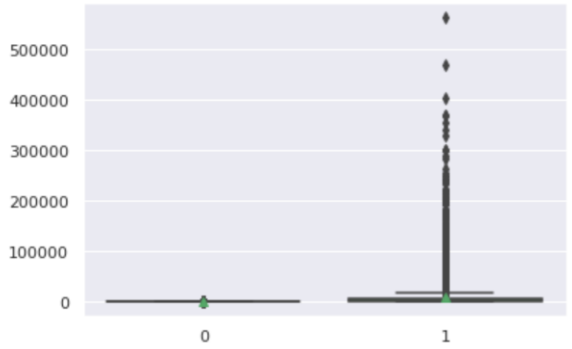
```
sns.boxplot(
    data=[df_over_move_at_po_wa['overall'],
          df_over_move_at_po_wa['attacking_short_passing'],
          ],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2cc5f6c88>
```

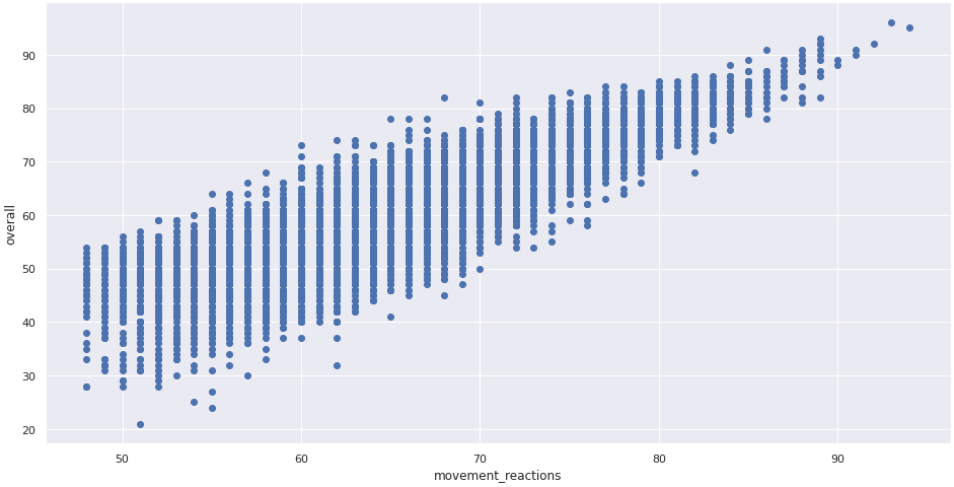



```
sns.boxplot(
    data=[df_over_move_at_po_wa['overall'],
          df_over_move_at_po_wa['wage_eur'],
          ],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
```

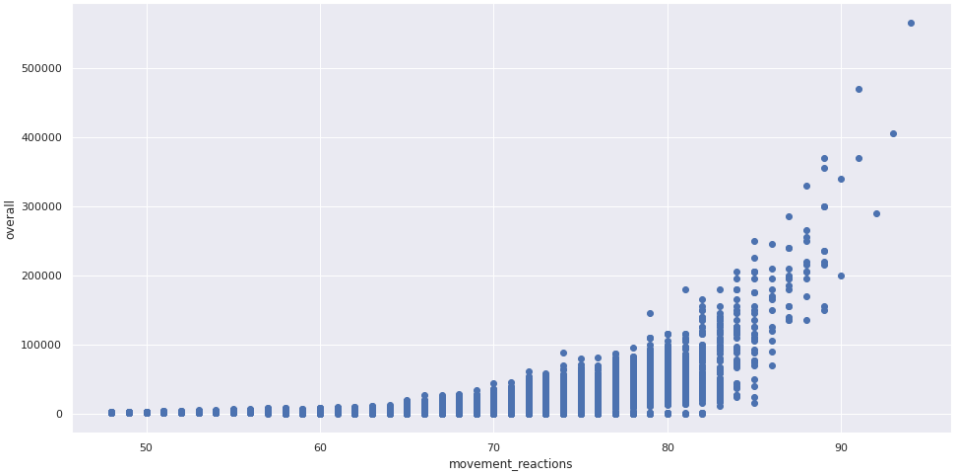
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2cc590940>
```



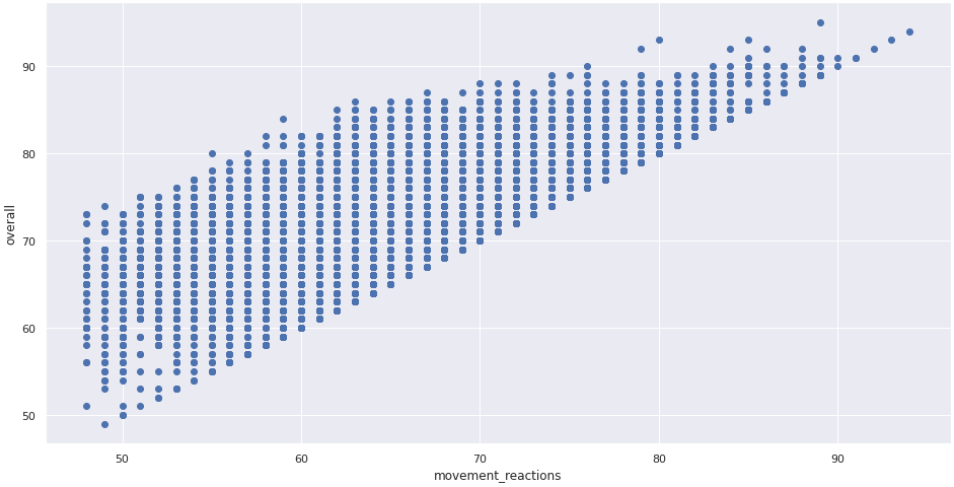

```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_over_move_at_po_wa['overall'], df_over_move_at_po_wa['movement_reactions'])
ax.set_xlabel('movement_reactions')
ax.set_ylabel('overall')
plt.show()
```

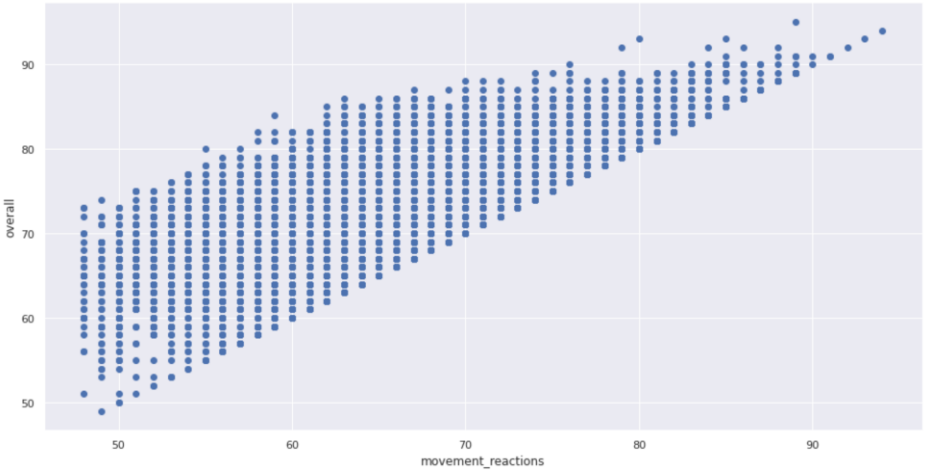



```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_over_move_at_po_wa['overall'], df_over_move_at_po_wa['wage_eur'])
ax.set_xlabel('movement_reactions')
ax.set_ylabel('overall')
plt.show()
```

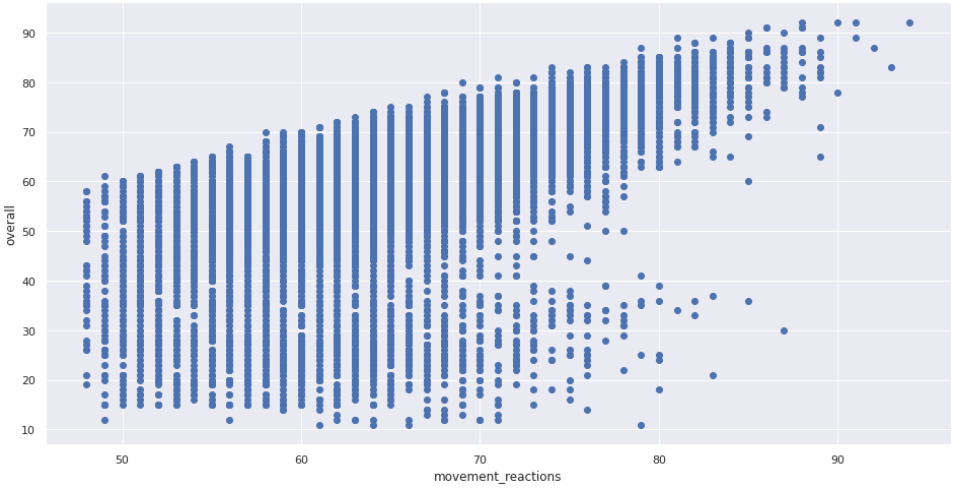



```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_over_move_at_po_wa['overall'], df_over_move_at_po_wa['potential'])
ax.set_xlabel('movement_reactions')
ax.set_ylabel('overall')
plt.show()
```






```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_over_move_at_po_wa['overall'], df_over_move_at_po_wa['attacking_short_passing'])
ax.set_xlabel('movement_reactions')
ax.set_ylabel('overall')
plt.show()
```



There is quite a bit of outliers on all the predictor features. Before I consider removing any of these outliers, I am going to apply DBscan clustering and isolation forest to see the outliers from a different perspective rather than a basic statistical plot.

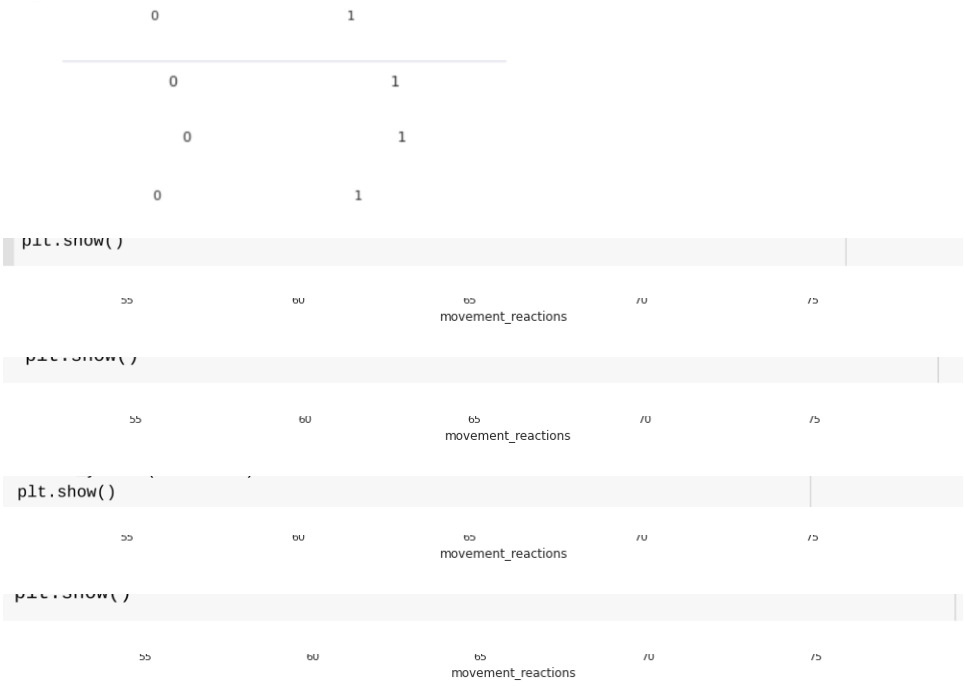
```
{-1: 7130, 1: 8724}
```

It is odd that DBscan indicated there are 0 outlier clusters, but the isolation forest shows there are 7130 outliers. The question now that remains is what to do with these outliers. I don't think the removal of outliers in this particular case is a good choice though because the fact that these are player values that give them are unique stats for each player and to how they perform and so of course there are going to be players who are outstanding and will create that outlier and removing these outliers means removing players and their talent just because they are good athletes.

However, for the performance of the model, I do believe to predict the accuracy for the target feature "overall" it is important to do something about the outliers, so it does not completely distort the model. So, I will cap the outliers at the 5th and 95th percentile to replace these extreme outliers with more acceptable limits.

```
df_over_move_at_po_wa.loc[(df_over_move_at_po_wa['wage_eur'] > upper_lim_wa),'wage_eur'] = upper_lim_wa
df_over_move_at_po_wa.loc[(df_over_move_at_po_wa['wage_eur'] < lower_lim_wa),'wage_eur'] = lower_lim_wa
```

Now I am going to rerun the side by side boxplots and the isolation forest to see if this helped out with the outliers at all.



The flooring/capping of the outliers for all of these predictor variables and the target variable has made a significant improvement compared to what they were looking like before especially the features "wage_eur" and "attacking_short_passing".

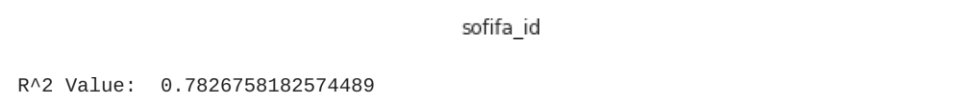
Now time to build the model.

```
modelcap.summary()
```

```
strong multicollinearity or other numerical problems.
```

```
242306       69.0  69.553306
```

```
plt.show()
```




```
R^2 Value:  0.7826758182574489
```

Based off the Rsquared value, this is a good fit, and those predictions are close also. However, just to show that this is a better performing model because of the feature engineering I did with the outliers, I will run the OLS model again on the features without the flooring/capping of the outliers.

```
# Print out the statistics
model_no_cap.summary()
```

```
strong multicollinearity or other numerical problems.
```

strong multicollinearity or other numerical problems.

```
plt.show()
```

So flooring/capping the data does improve the performance of the model, but I wouldn't want to floor or cap the data more than I already have because of the fact that it defeats the purpose of using the training/testing data for an OLS model if every value has been replaced to improve performance and ruins the model integrity.

There is high multicollinearity due to the warning indication for the condition number, but I tried this with the one variable "movement_reactions" and that was still an okay performing model, but the problem was the Rsquared was .48.

Since this was only the training/testing of the first dataset, I am going to now run the floor/capped model against the FIFAData_predictions now and save the predicted values into a .csv file.

113422	80

3. Decision Tree

The libraries I used for this section were...

- Sklearn – OneHotEncoder, DecisionTreeClassifier, StringIO, graph_from_dot_data, confusion_matrix, accuracy_score
- Pydot - export_graphviz
- IPython - Image

This is my first time I will be using a decision tree, and my reason for choosing it is so I can learn another model I never used before. My approach with predicting which players have "United States" as classified for their "nationality" feature will be to first explore the data and find some insight on player nationality first before creating the decision tree model. I will select the "sofifa_id","nationality", and "club" features for exploration, since the "club" feature appears to be the only feature that can classify the nationality of each player categorically.

Looking at the value_counts for all the players and the shape of the subset data frame of that for United States players shows me that there are only 295 U.S. players in the FIFA game out of 15,854 players with only 698 club classifications to indicate if the player is from the U.S. or not.

So now I will begin to prep the data for the decision tree by getting dummies for numerical categorization of the nationality and transforming the club feature data to train the decision tree. Then I will train/test split the data to get ready for the decision tree.

What I did below took a lot of trial and error. Since the prediction data has an input dimension of 633 labels in comparison to the main dataset which has all 698 labels this creates an input mismatch error when trying to finally predict upon the data. So, I had to concatenate both datasets together to give the prediction dataset the same labeled features as the main dataset to pass it through the decision tree.

Now time to fit the decision tree with the training data to run the model on the predictions.

After fitting the model, I learned that a visual representation of the decision tree at work is sometimes good to show what is going on internally in the model.

Now to predict on the testing data and get the model accuracy.

```
[ 66   12]]
```

This confusion matrix right here is indicating that the decision tree model is performing at 97.9% accuracy at correctly classifying the nationality of FIFA players being of United States nationality with there being 3870 true positives, 16 false positives, 66 false negatives, and 12 true negatives.

Now that the decision tree model is predicting at 97% accuracy it is time to accurately predict the player nationalities.

```
[68] df_nation_pred_X = df_nation_pred[['club']]
     df_nation_pred_X = m.fit_transform(df_nation_pred_test)
```

```
[69] y_pred_nation_dt = dt.predict(df_nation_pred_test)
```

```
y_pred_nation_dt
```

```
array([[1, 0],
       [1, 0],
       [1, 0],
       ...,
       [1, 0],
       [1, 0],
       [0, 1]], dtype=uint8)
```

Since this array does not make any sense, I am going to use np.argmax() to inverse the predictions back to their regular categorical factors so I can manipulate the dataset in a spreadsheet afterwards.

From here I went into google spreadsheets and changed the integers over to their respectful predictions and saved the file in .csv format.

14		12	140445	Not United State