

In [144...]

```
from PIL import Image
img = Image.open(r"D:\OneDrive\Desktop\wp2522020.jpg")
img
```

Out[144...]



Import libraries

```
In [3]: import numpy as np
import pandas as pd
import seaborn as sns
sns.set(style='whitegrid')
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

```
In [4]: import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Read dataset

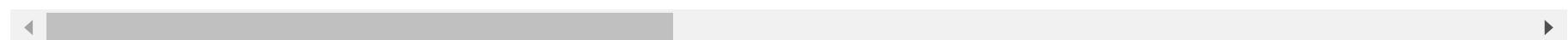
```
In [5]: fifa = pd.read_csv(r"D:\OneDrive\Desktop\FIFA.csv",index_col=0)
```

```
In [6]: fifa
```

Out[6]:

	ID	Name	Age	Photo	Nationality	Flag	Overall	Pote
0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	
1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	
2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	
3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	
4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	
...
18202	238813	J. Lundstram	19	https://cdn.sofifa.org/players/4/19/238813.png	England	https://cdn.sofifa.org/flags/14.png	47	
18203	243165	N. Christoffersson	19	https://cdn.sofifa.org/players/4/19/243165.png	Sweden	https://cdn.sofifa.org/flags/46.png	47	
18204	241638	B. Worman	16	https://cdn.sofifa.org/players/4/19/241638.png	England	https://cdn.sofifa.org/flags/14.png	47	
18205	246268	D. Walker-Rice	17	https://cdn.sofifa.org/players/4/19/246268.png	England	https://cdn.sofifa.org/flags/14.png	47	
18206	246269	G. Nugent	16	https://cdn.sofifa.org/players/4/19/246269.png	England	https://cdn.sofifa.org/flags/14.png	46	

18207 rows × 88 columns



Preview the dataset

In [7]: `fifa.head()`

Out[7]:

	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	
0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	Ba
1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Ju
2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris G
3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Man
4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Man

5 rows × 88 columns



View summary of dataset

In [8]: `fifa.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column           Non-Null Count Dtype
 ---  -----
 0   ID               18207 non-null  int64
 1   Name              18207 non-null  object
 2   Age               18207 non-null  int64
 3   Photo              18207 non-null  object
 4   Nationality        18207 non-null  object
 5   Flag              18207 non-null  object
 6   Overall            18207 non-null  int64
 7   Potential           18207 non-null  int64
 8   Club               17966 non-null  object
 9   Club Logo           18207 non-null  object
 10  Value              18207 non-null  object
 11  Wage               18207 non-null  object
 12  Special             18207 non-null  int64
 13  Preferred Foot      18159 non-null  object
 14  International Reputation 18159 non-null  float64
 15  Weak Foot            18159 non-null  float64
 16  Skill Moves          18159 non-null  float64
 17  Work Rate             18159 non-null  object
 18  Body Type             18159 non-null  object
 19  Real Face             18159 non-null  object
 20  Position              18147 non-null  object
 21  Jersey Number          18147 non-null  float64
 22  Joined               16654 non-null  object
 23  Loaned From            1264 non-null  object
 24  Contract Valid Until    17918 non-null  object
 25  Height                18159 non-null  object
 26  Weight                18159 non-null  object
 27  LS                     16122 non-null  object
 28  ST                     16122 non-null  object
 29  RS                     16122 non-null  object
 30  LW                     16122 non-null  object
 31  LF                     16122 non-null  object
 32  CF                     16122 non-null  object
 33  RF                     16122 non-null  object
 34  RW                     16122 non-null  object
 35  LAM                    16122 non-null  object
```

36	CAM	16122	non-null	object
37	RAM	16122	non-null	object
38	LM	16122	non-null	object
39	LCM	16122	non-null	object
40	CM	16122	non-null	object
41	RCM	16122	non-null	object
42	RM	16122	non-null	object
43	LWB	16122	non-null	object
44	LDM	16122	non-null	object
45	CDM	16122	non-null	object
46	RDM	16122	non-null	object
47	RWB	16122	non-null	object
48	LB	16122	non-null	object
49	LCB	16122	non-null	object
50	CB	16122	non-null	object
51	RCB	16122	non-null	object
52	RB	16122	non-null	object
53	Crossing	18159	non-null	float64
54	Finishing	18159	non-null	float64
55	HeadingAccuracy	18159	non-null	float64
56	ShortPassing	18159	non-null	float64
57	Volleys	18159	non-null	float64
58	Dribbling	18159	non-null	float64
59	Curve	18159	non-null	float64
60	FKAccuracy	18159	non-null	float64
61	LongPassing	18159	non-null	float64
62	BallControl	18159	non-null	float64
63	Acceleration	18159	non-null	float64
64	SprintSpeed	18159	non-null	float64
65	Agility	18159	non-null	float64
66	Reactions	18159	non-null	float64
67	Balance	18159	non-null	float64
68	ShotPower	18159	non-null	float64
69	Jumping	18159	non-null	float64
70	Stamina	18159	non-null	float64
71	Strength	18159	non-null	float64
72	LongShots	18159	non-null	float64
73	Aggression	18159	non-null	float64
74	Interceptions	18159	non-null	float64
75	Positioning	18159	non-null	float64
76	Vision	18159	non-null	float64

```
77  Penalties           18159 non-null float64
78  Composure          18159 non-null float64
79  Marking            18159 non-null float64
80  StandingTackle     18159 non-null float64
81  SlidingTackle      18159 non-null float64
82  GKDiving           18159 non-null float64
83  GKHandling          18159 non-null float64
84  GKKicking           18159 non-null float64
85  GKPositioning       18159 non-null float64
86  GKReflexes          18159 non-null float64
87  Release Clause     16643 non-null object
dtypes: float64(38), int64(5), object(45)
memory usage: 12.4+ MB
```

```
In [9]: fifa['Body Type'].value_counts()
```

```
Out[9]: Body Type
Normal              10595
Lean                6417
Stocky              1140
Messi               1
C. Ronaldo          1
Neymar              1
Courtois             1
PLAYER_BODY_TYPE_25 1
Shaqiri              1
Akinfenwa            1
Name: count, dtype: int64
```

Comment

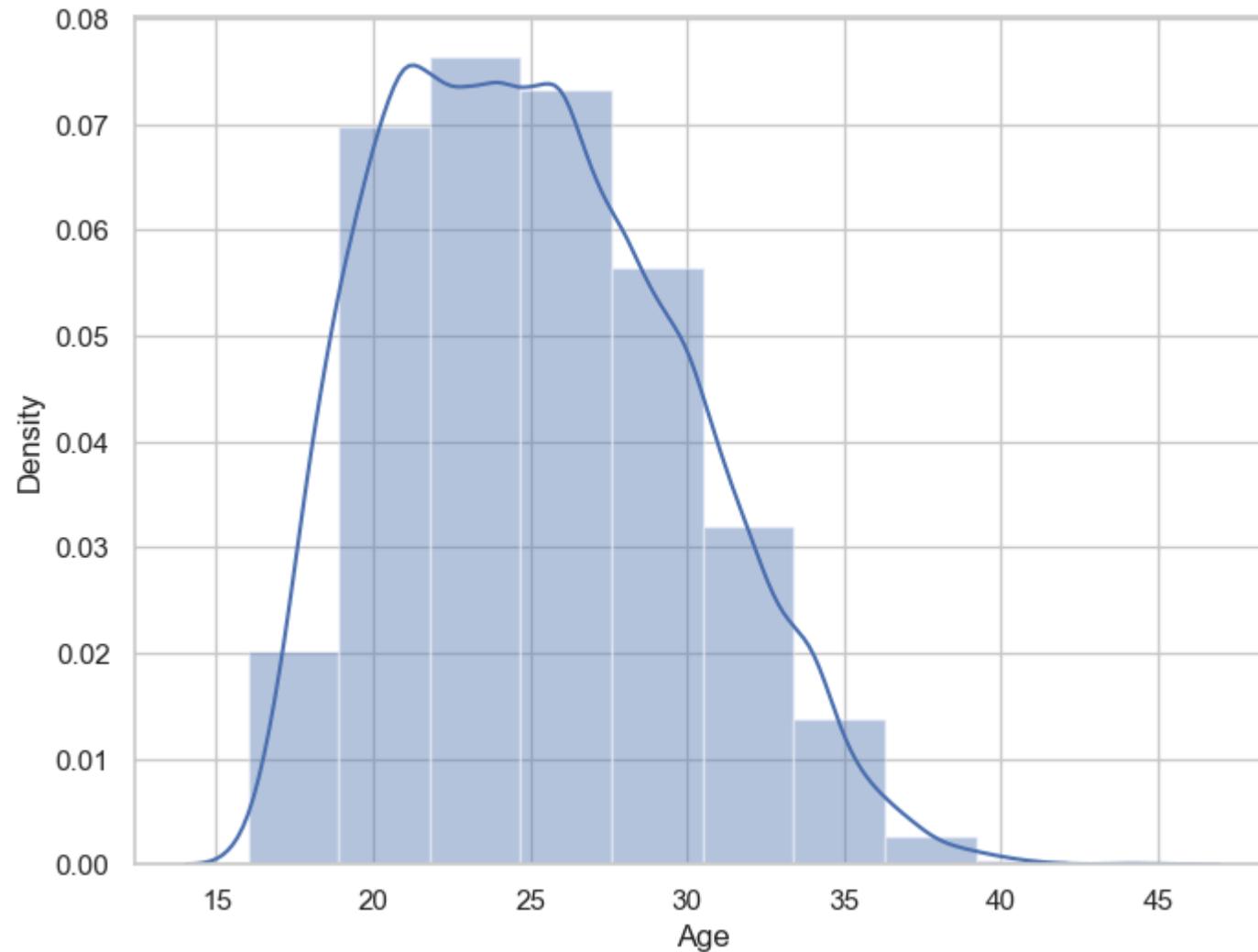
- This dataset contains 89 variables.
- Out of the 89 variables, 44 are numerical variables. 38 are of float64 data type and remaining 6 are of int64 data type.
- The remaining 45 variables are of character data type.
- Let's explore this further.

Explore Age variable

Visualize distribution of Age variable with Seaborn distplot() function

- Seaborn `distplot()` function flexibly plots a univariate distribution of observations.
- This function combines the matplotlib `hist` function (with automatic calculation of a good default bin size) with the seaborn `kdeplot()` and `rugplot()` functions.
- ■ So, let's visualize the distribution of `Age` variable with Seaborn `distplot()` function.

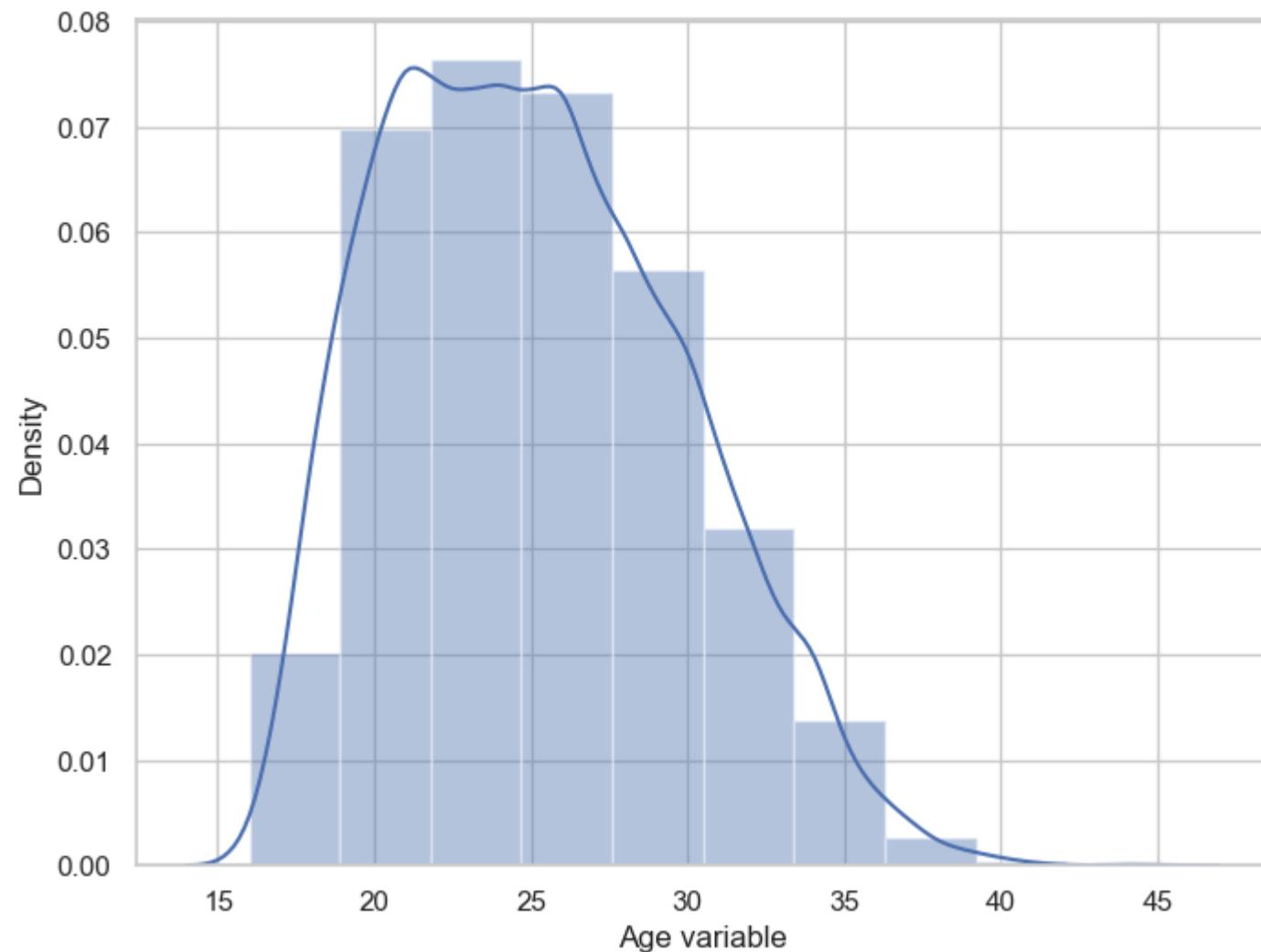
```
In [10]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
ax = sns.distplot(x, bins=10)
plt.show()
```



Comment

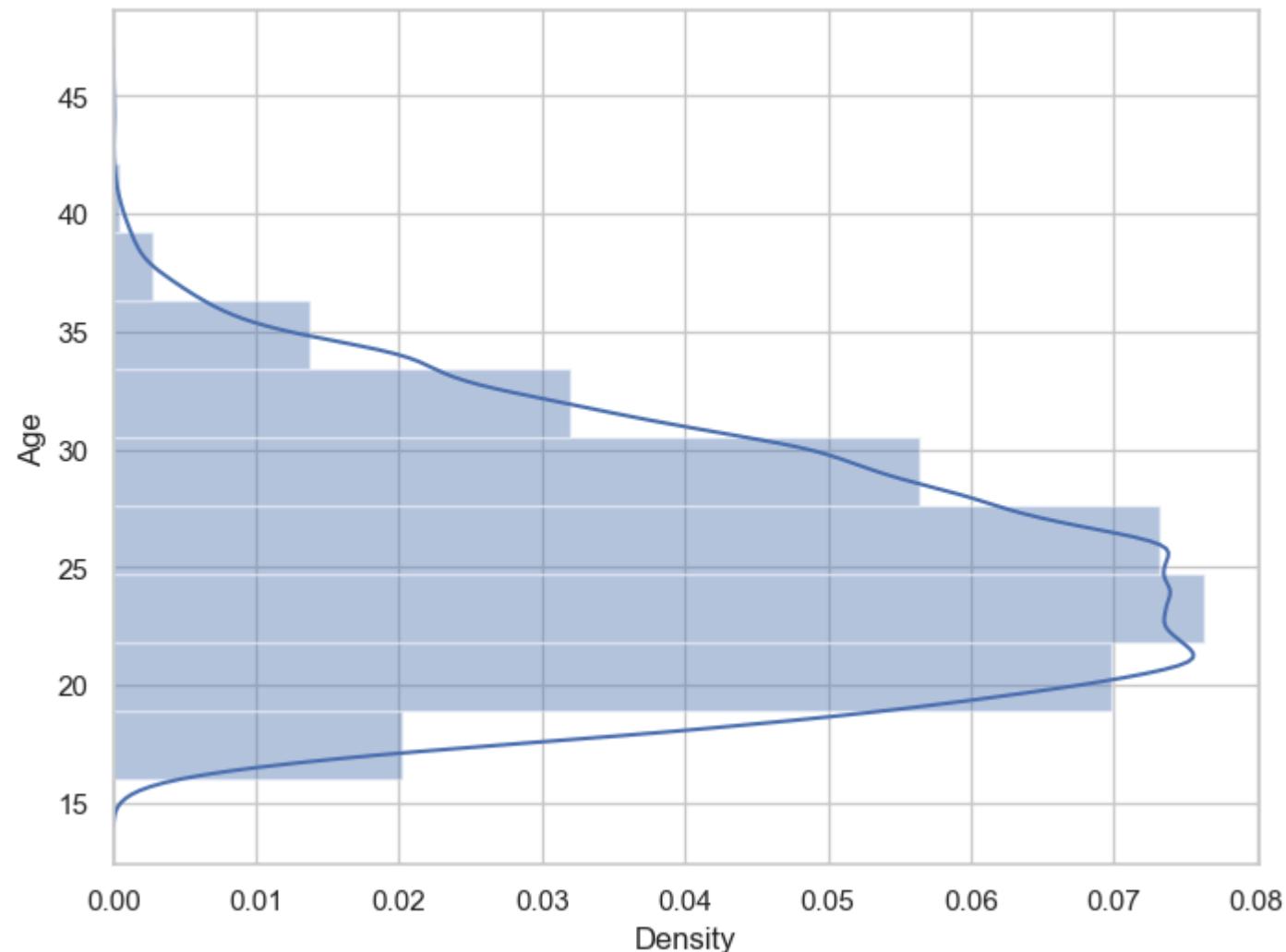
- It can be seen that the `Age` variable is slightly positively skewed.
- we can use pandas series object to get an information axis label as follows

```
In [11]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
x= pd.Series(x,name="Age variable")
ax = sns.distplot(x, bins=10)
plt.show()
```



we can plot the distribution on the vertical axis as follows

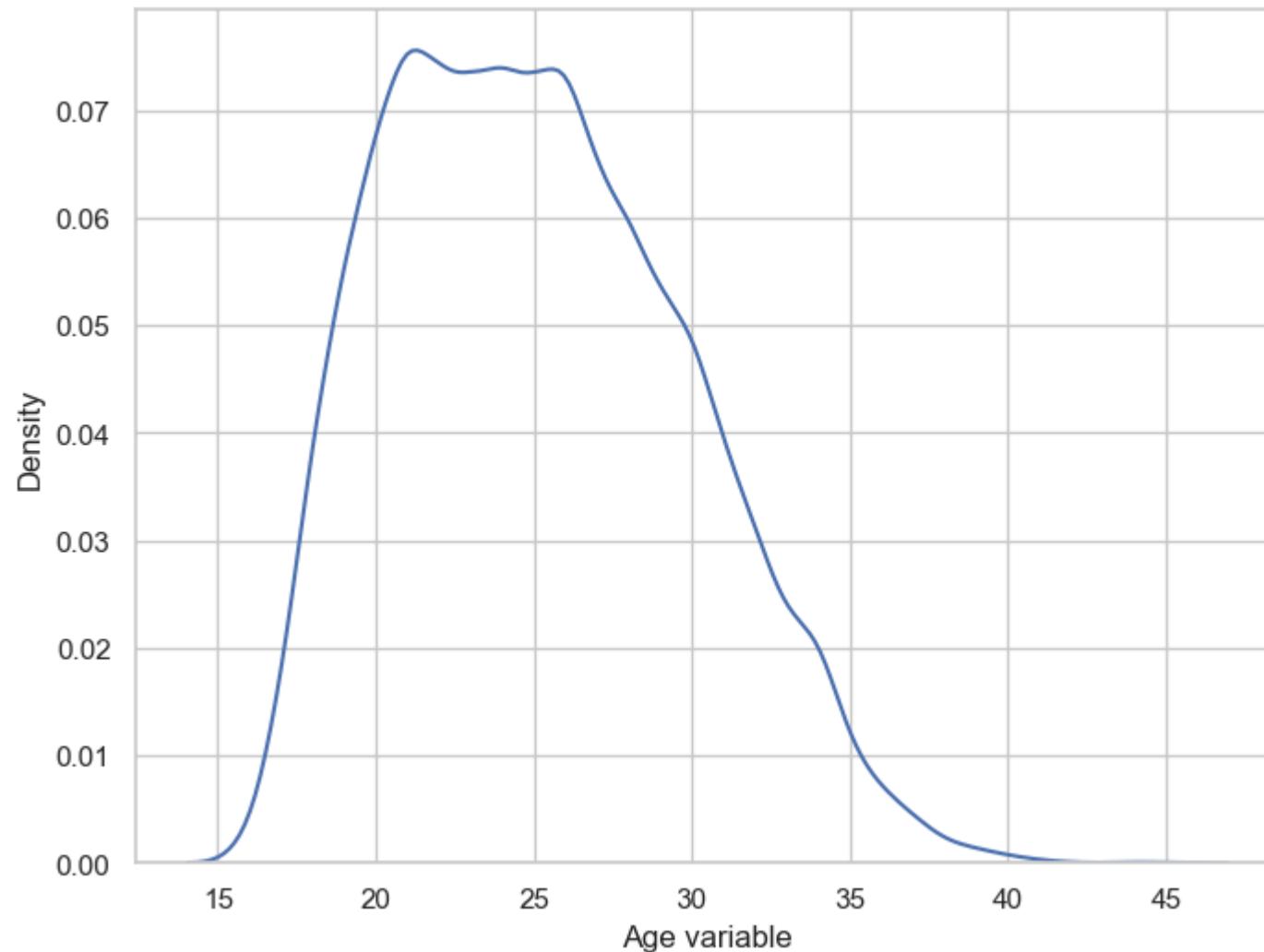
```
In [12]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
# x= pd.Series(x, name="Age variable")
ax = sns.distplot(x, bins=10, vertical = True)
plt.show()
```



Seaborn Kernel Density Estimation (KDE) Plot

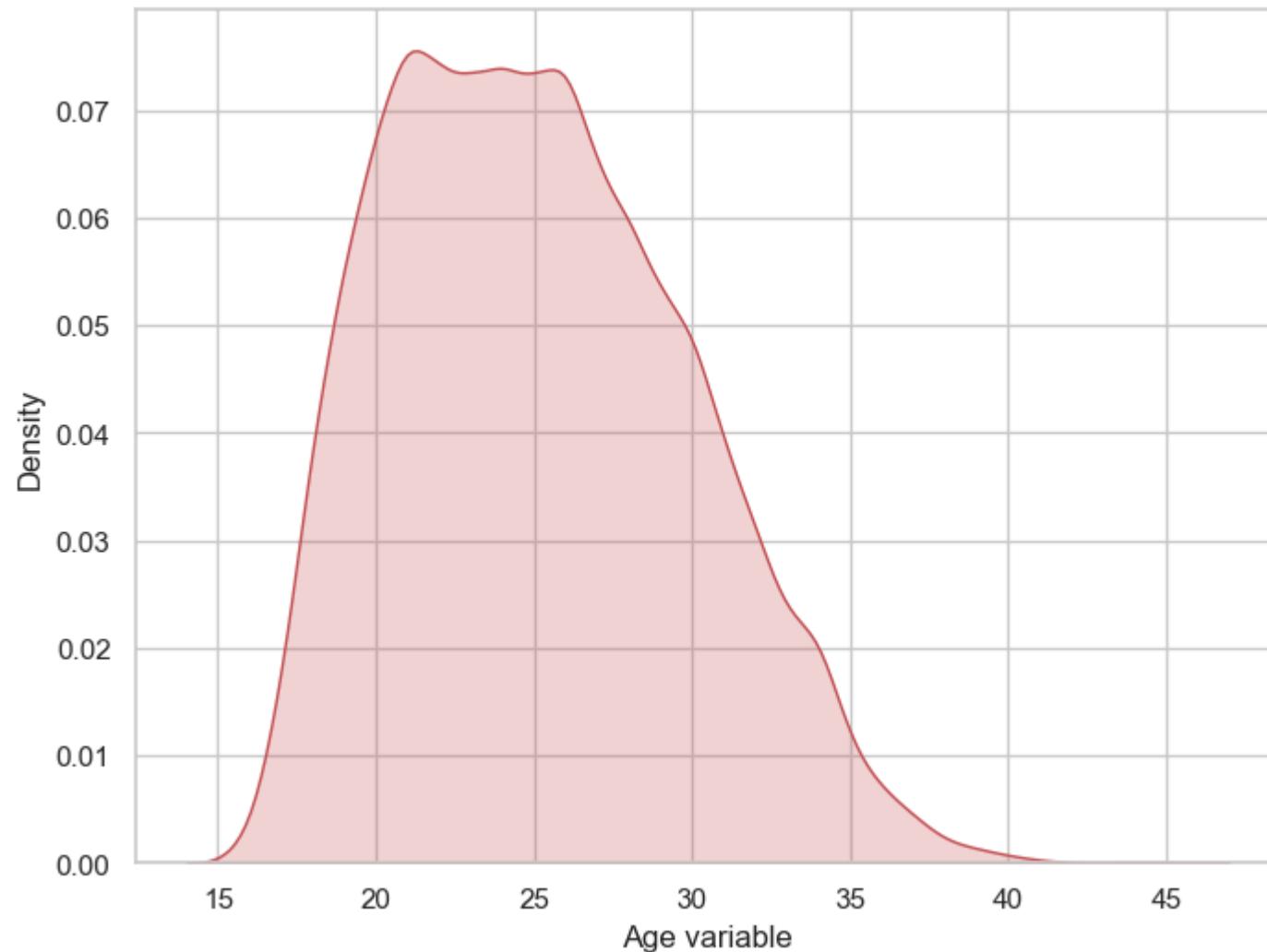
- The `kernel density estimate (KDE)` plot is a useful tool for plotting the shape of a distribution.
- Seaborn `kdeplot` is another seaborn plotting function that fits and plot a univariate or bivariate kernel density estimate.
- Like the histogram, the KDE plots encode the density of observations on one axis with height along the other axis.
- We can plot a KDE plot as follows-

```
In [13]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
x= pd.Series(x,name="Age variable")
ax = sns.kdeplot(x)
plt.show()
```



We can shade under the density curve and use a different color as follows:-

```
In [14]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
x= pd.Series(x,name="Age variable")
ax = sns.kdeplot(x, shade=True,color='r')
plt.show()
```

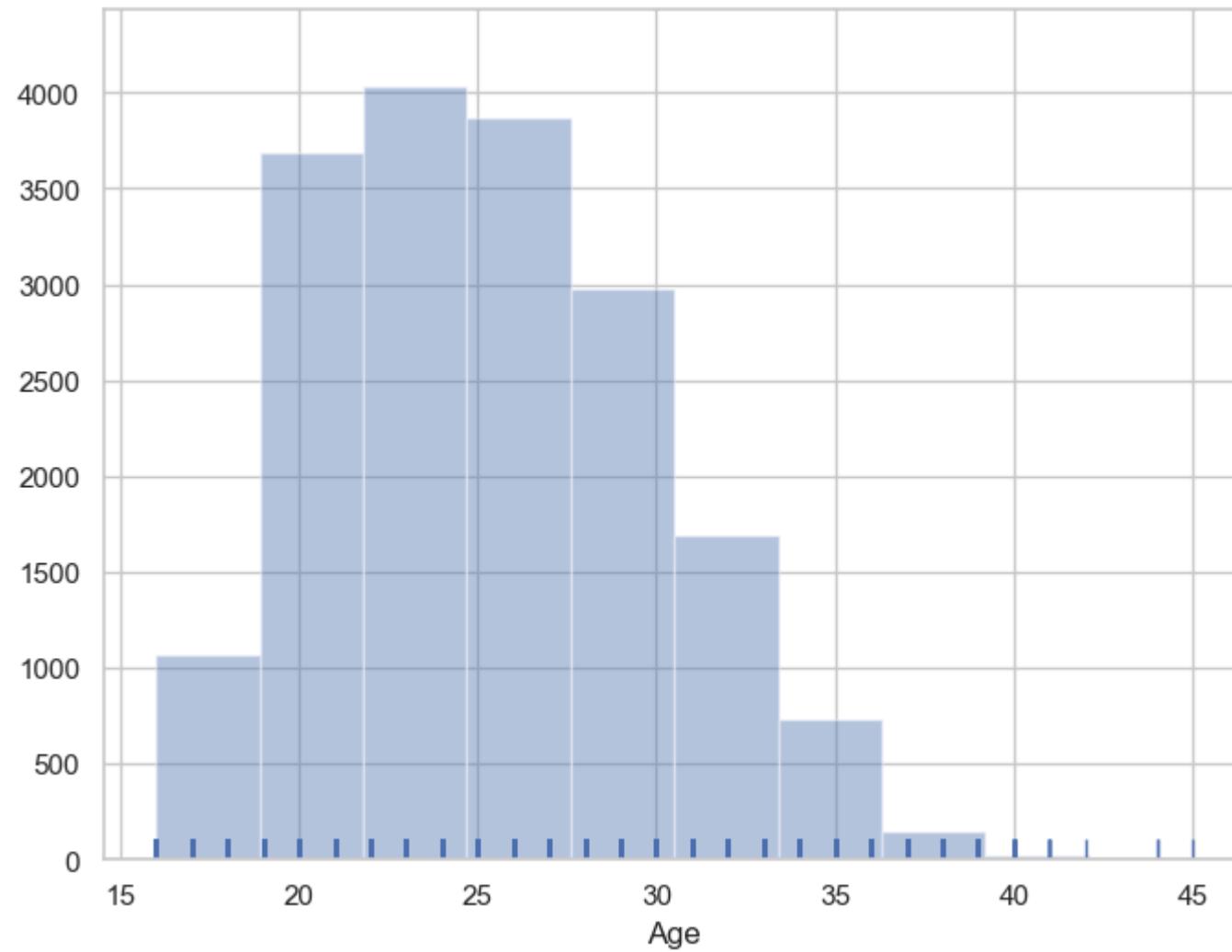


Histograms

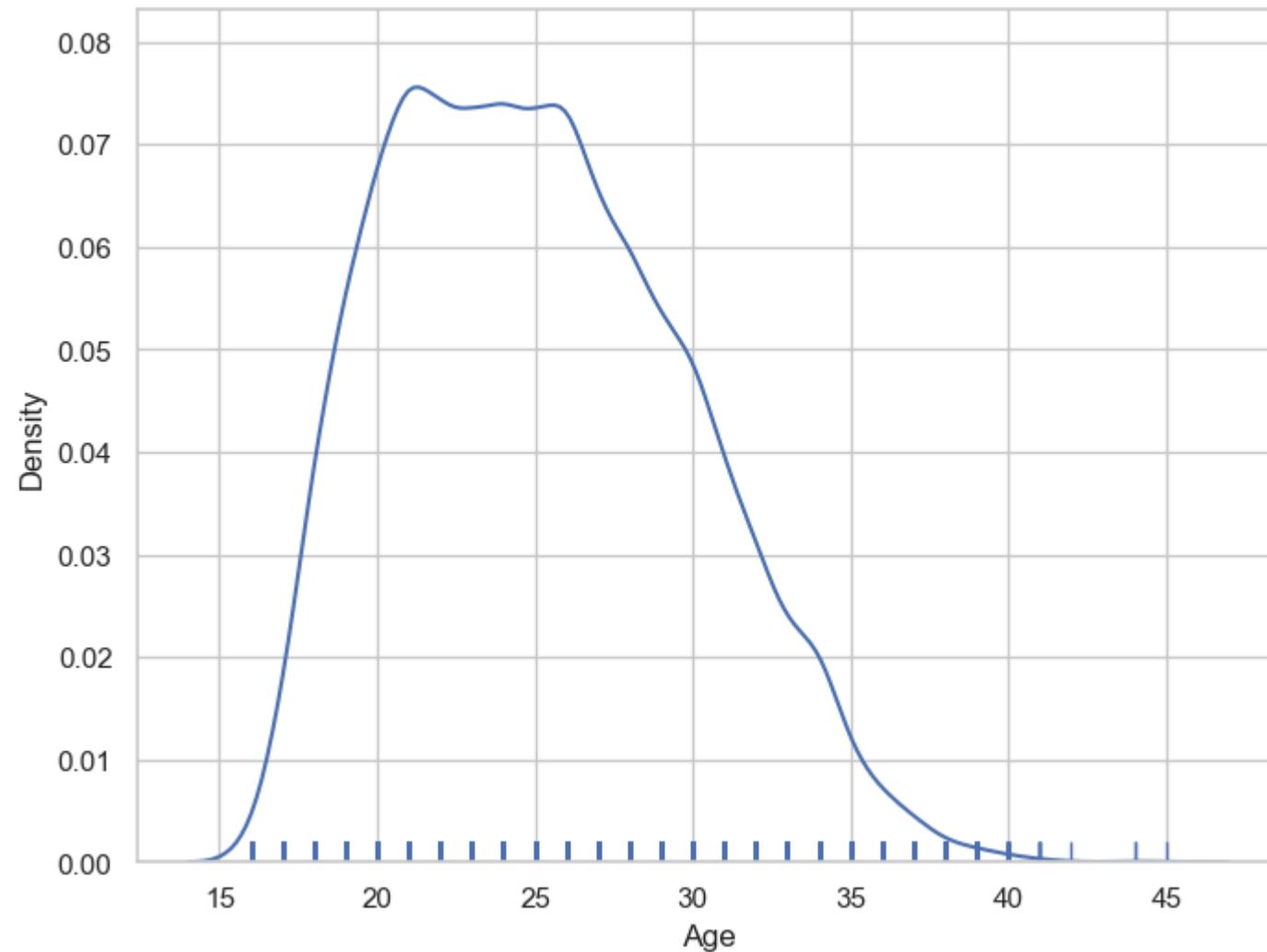
- A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.
- A `hist()` function already exists in matplotlib.

- We can use Seaborn to plot a histogram.

```
In [15]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
ax = sns.distplot(x,kde=False,rug=True,bins=10)
plt.show()
```



```
In [16]: f, ax = plt.subplots(figsize=(8,6))
x = fifa['Age']
ax = sns.distplot(x,hist=False,rug=True,bins=10)
plt.show()
```



Explore preferred FOOT Variable

Check frequency distribution of values in Preferred Foot variable

```
In [17]: fifa['Preferred Foot'].nunique()
```

```
Out[17]: 2
```

We can see that there are two types of unique values in Preferred Foot variable.

```
In [18]: fifa['Preferred Foot'].value_counts()
```

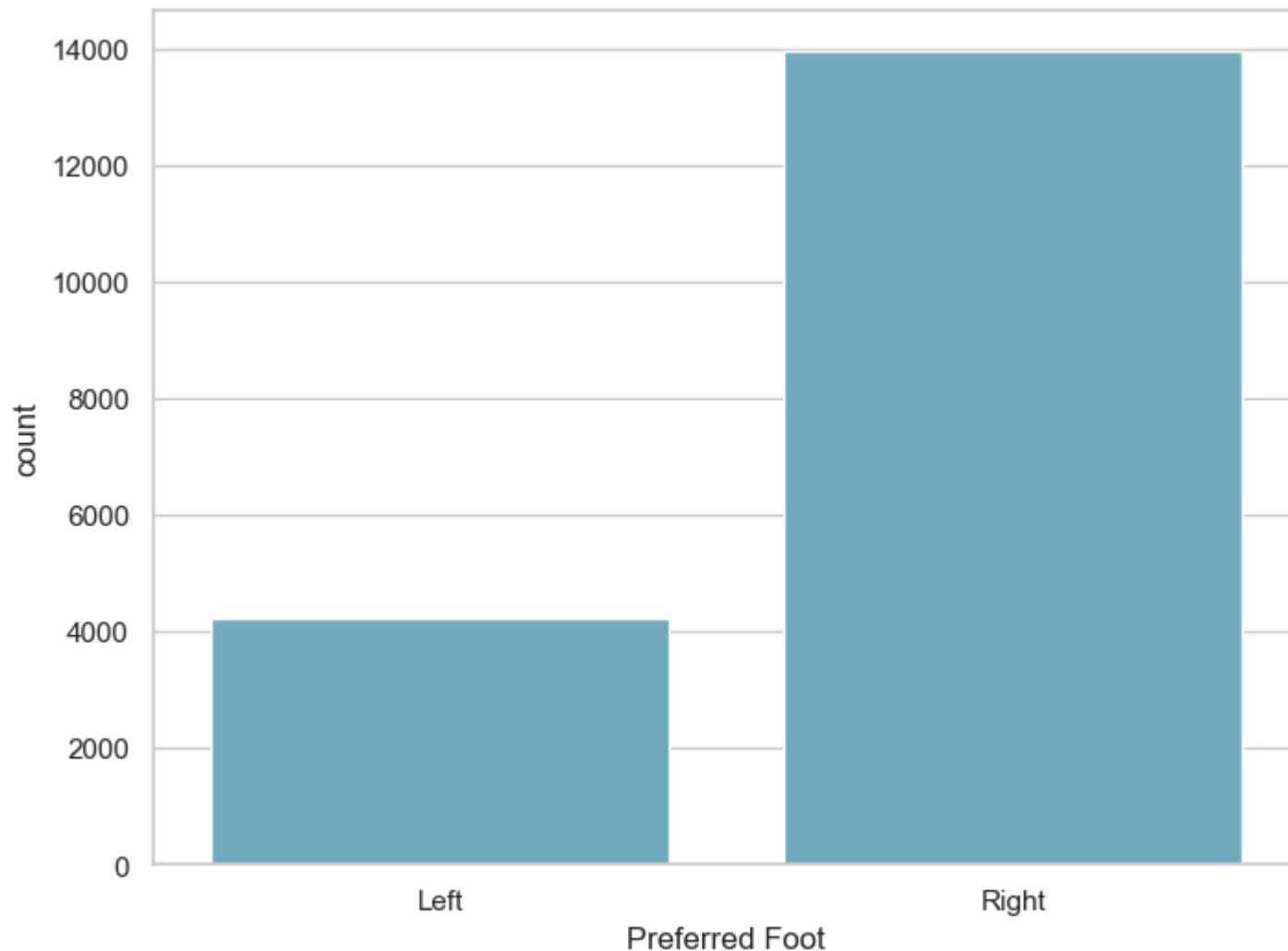
```
Out[18]: Preferred Foot
Right    13948
Left     4211
Name: count, dtype: int64
```

The Preferred Foot variable contains two types of values - Right and Left .

Visualize distribution of values with Seaborn countplot() function.

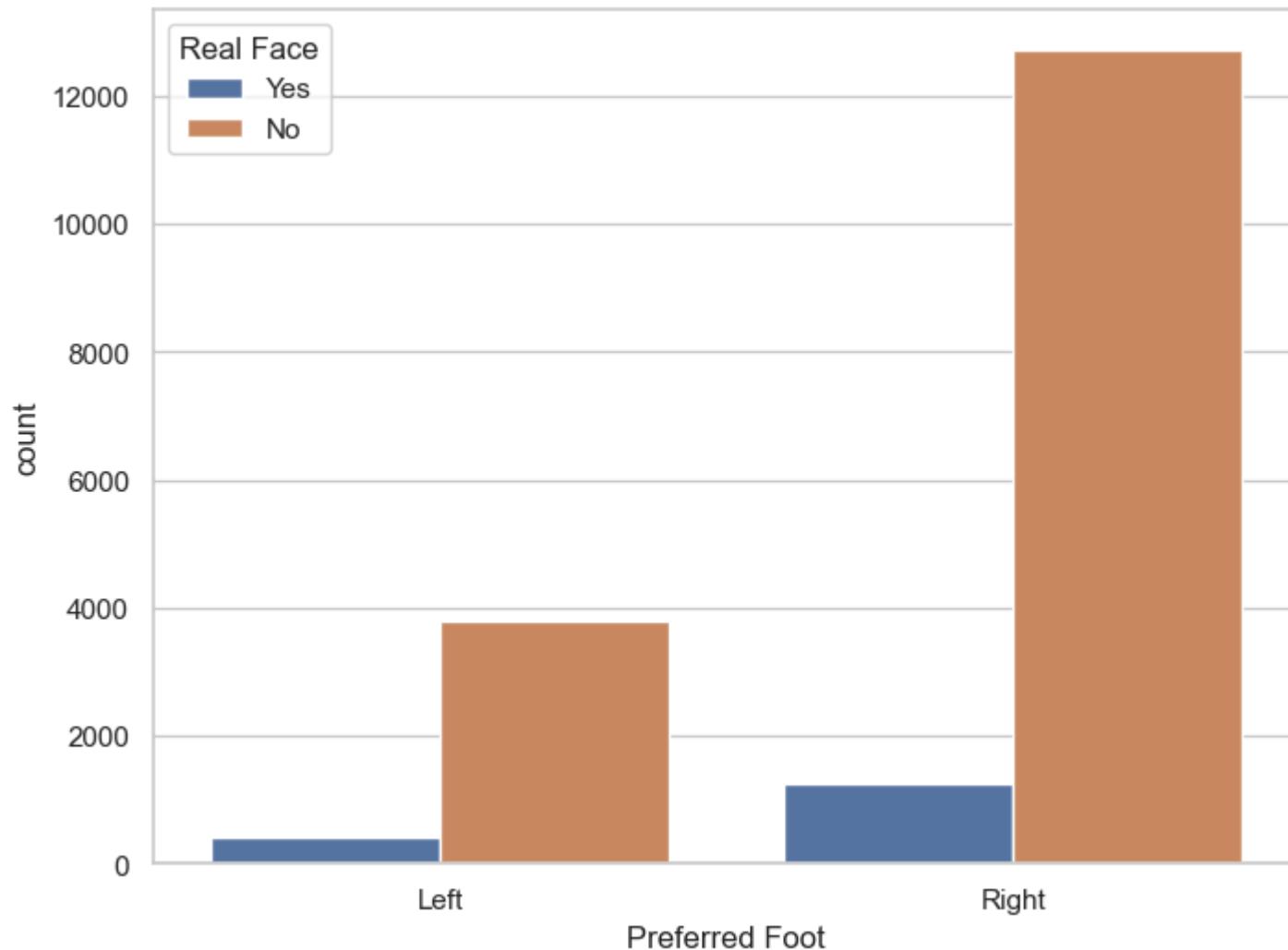
- A countplot shows the counts of observations in each categorical bin using bars.
 - It can be thought of as a histogram across a categorical, instead of quantitative, variable.
 - This function always treats one of the variables as categorical and draws data at ordinal positions (0, 1, ... n) on the relevant axis, even when the data has a numeric or date type.
- We can visualize the distribution of values with Seaborn countplot() function as follows-

```
In [19]: f, ax = plt.subplots(figsize=(8, 6))
sns.countplot(x="Preferred Foot", data=fifa, color="c")
plt.show()
```



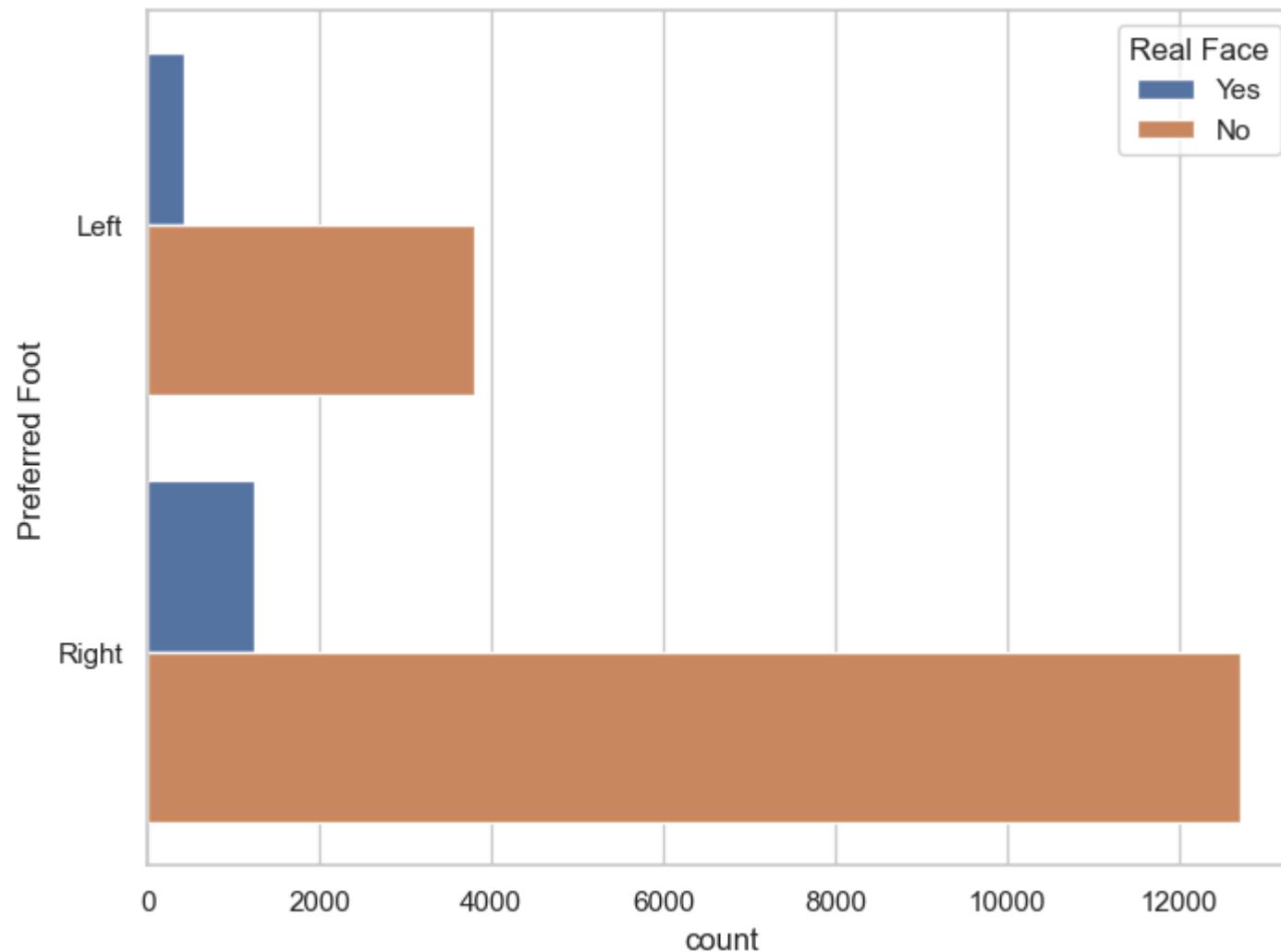
We can show value counts for two categorical variables as follows-

```
In [20]: f, ax = plt.subplots(figsize=(8,6))
sns.countplot(x="Preferred Foot", hue="Real Face", data=fifa)
plt.show()
```



We can draw plot vertically as follows-

```
In [21]: f, ax = plt.subplots(figsize=(8,6))
sns.countplot(y="Preferred Foot", hue="Real Face", data=fifa)
plt.show()
```

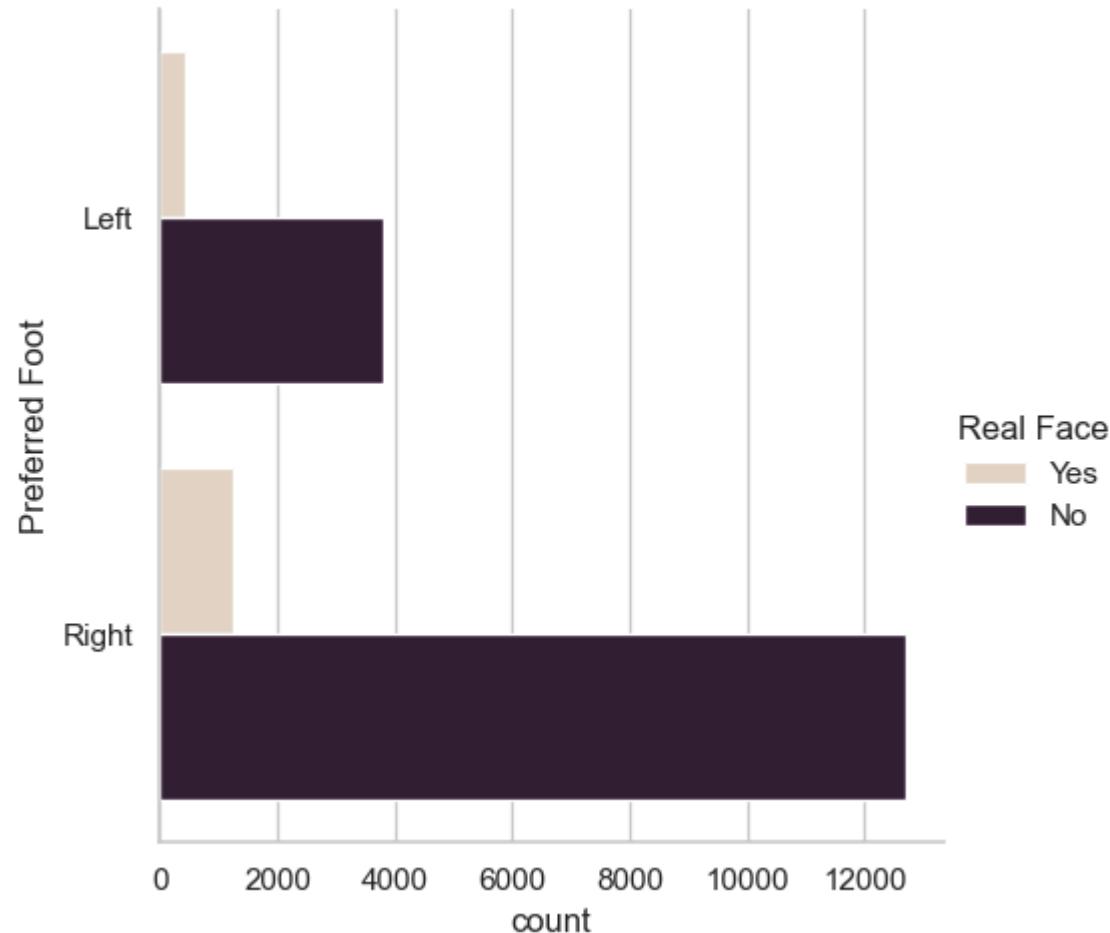


Seaborn `Catplot()` function

- We can use Seaborn `Catplot()` function to plot categorical scatterplots.
- The default representation of the data in `catplot()` uses a scatterplot.
- It helps to draw figure-level interface for drawing categorical plots onto a `facetGrid`.

- This function provides access to several axes-level functions that show the relationship between a numerical and one or more categorical variables using one of several visual representations.
- The `kind` parameter selects the underlying axes-level function to use.

```
In [22]: g = sns.catplot(y="Preferred Foot", hue="Real Face", data=fifa, kind="count", palette="ch:.25")
```



Explore International Reputation variable

Check the number of unique values in International Reputation variable

```
In [23]: fifa['International Reputation'].nunique()
```

```
Out[23]: 5
```

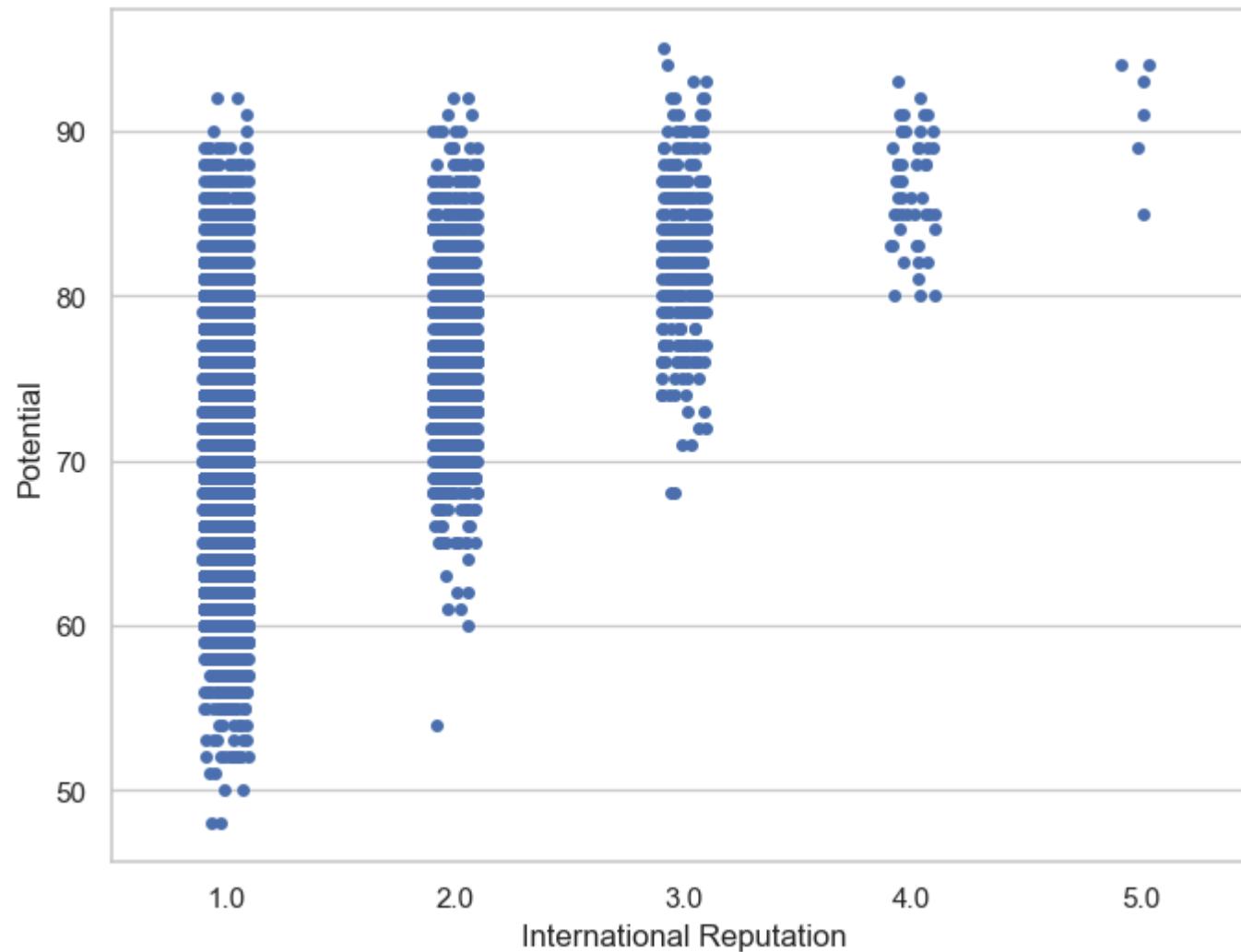
```
In [24]: fifa['International Reputation'].value_counts()
```

```
Out[24]: International Reputation
1.0    16532
2.0    1261
3.0     309
4.0      51
5.0       6
Name: count, dtype: int64
```

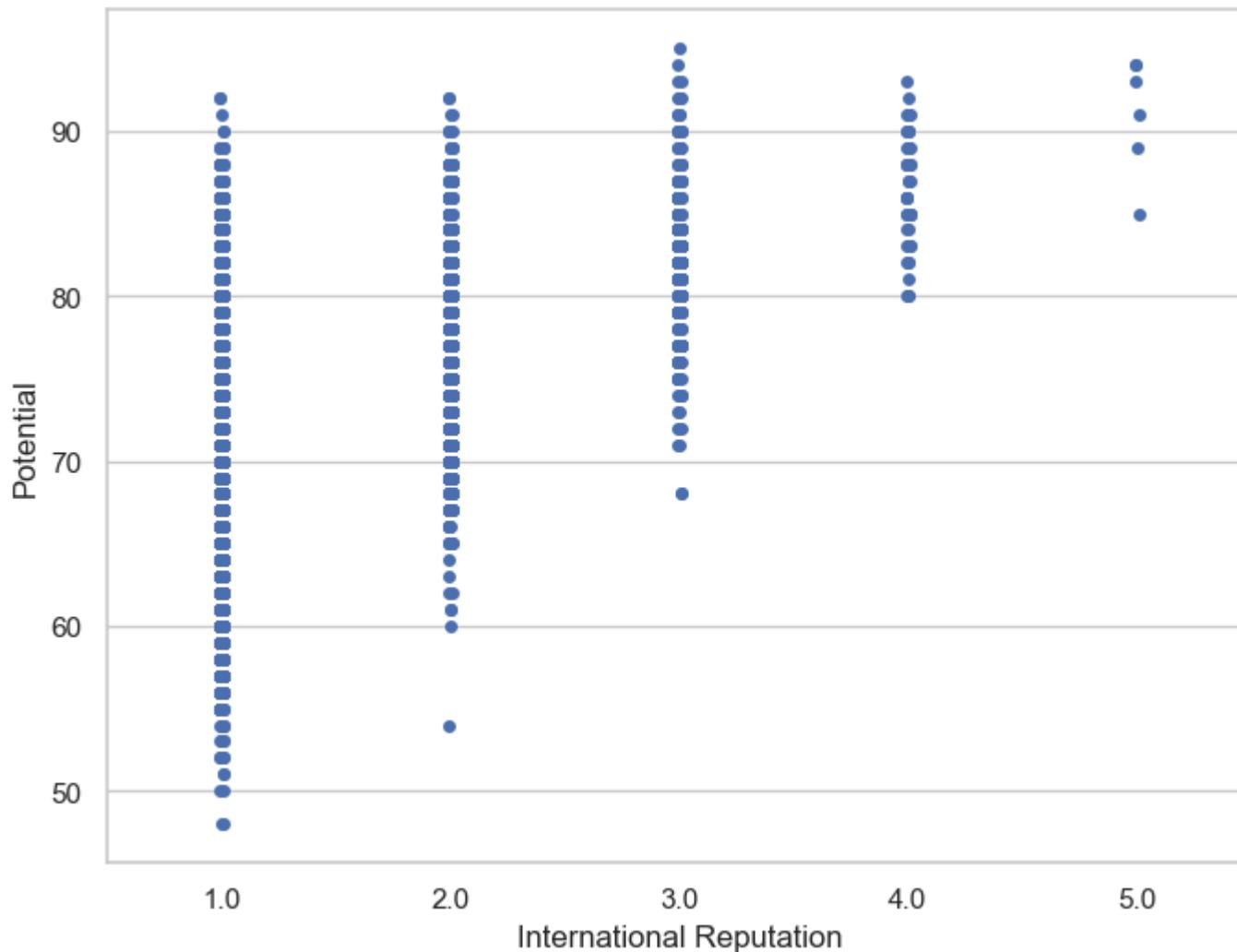
Seaborn Stripplot() function

- This function draws a scatterplot where one variable is categorical.
- A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where we want to show all observations along with some representation of the underlying distribution.
- I will plot a stripplot with `International Reputation` as categorical variable and `Potential` as the other variable.

```
In [25]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", data=fifa)
plt.show()
```

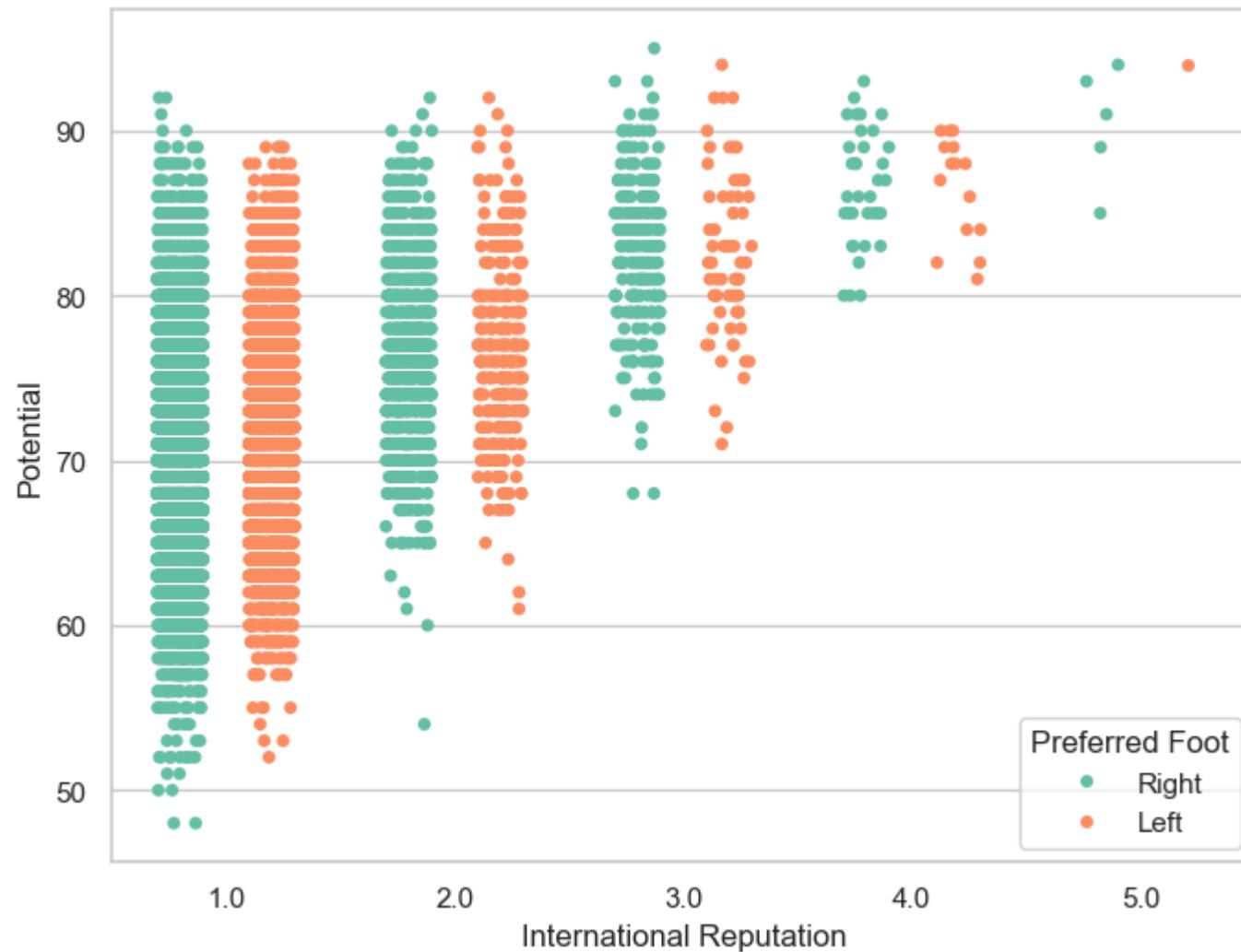


```
In [26]: f, ax = plt.subplots(figsize=(8,6))
sns.stripplot(x="International Reputation",y="Potential",data=fifa,jitter=0.01)
plt.show()
```



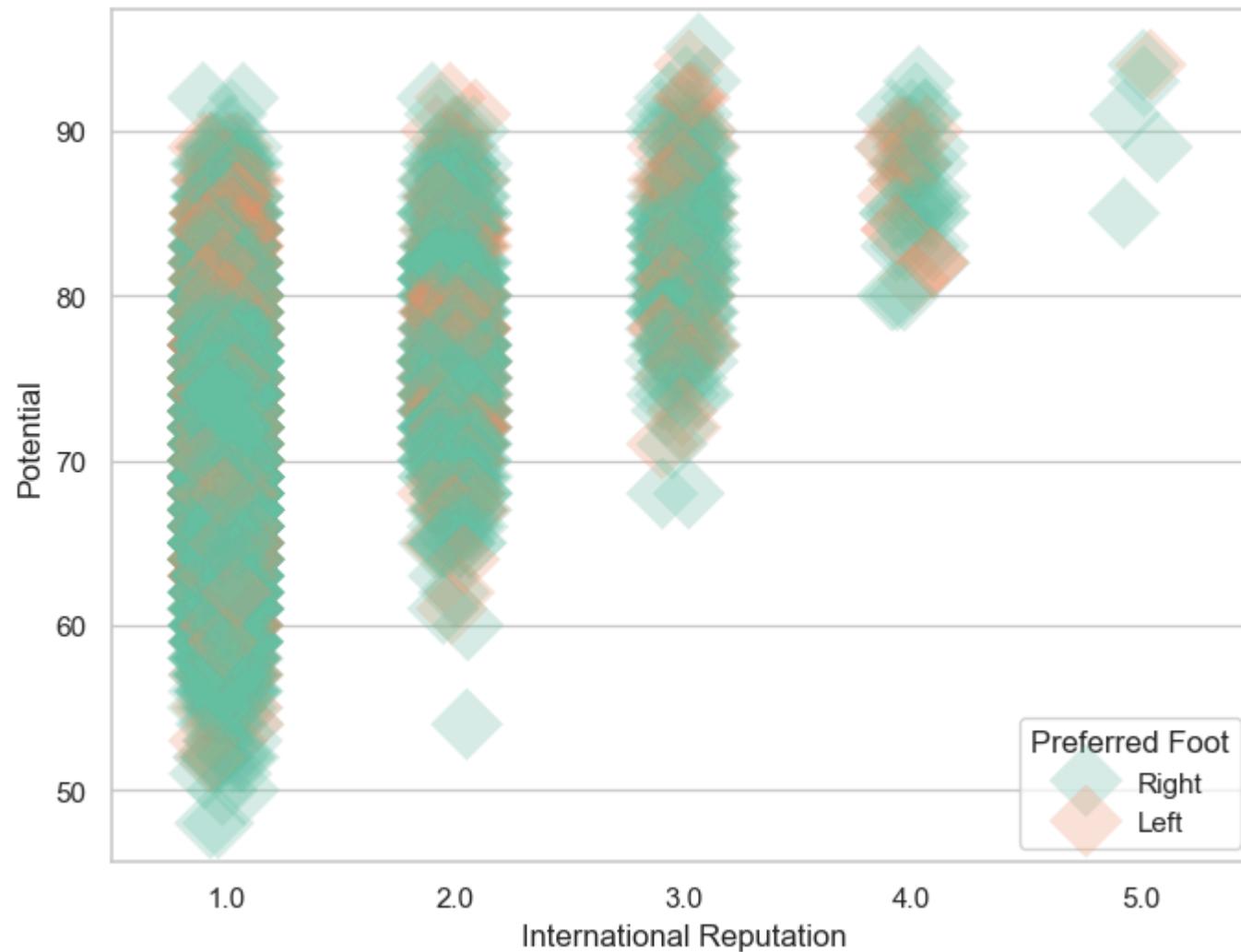
We can nest the strips within a second categorical variable - Preferred Foot as follows-

```
In [27]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", hue="Preferred Foot",
               data=fifa, jitter=0.2, palette="Set2", dodge=True)
plt.show()
```



We can draw strips with large points and different aesthetics as follows-

```
In [28]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", hue="Preferred Foot",
               data=fifa, palette="Set2", size=20, marker="D",
               edgecolor="gray", alpha=.25)
plt.show()
```

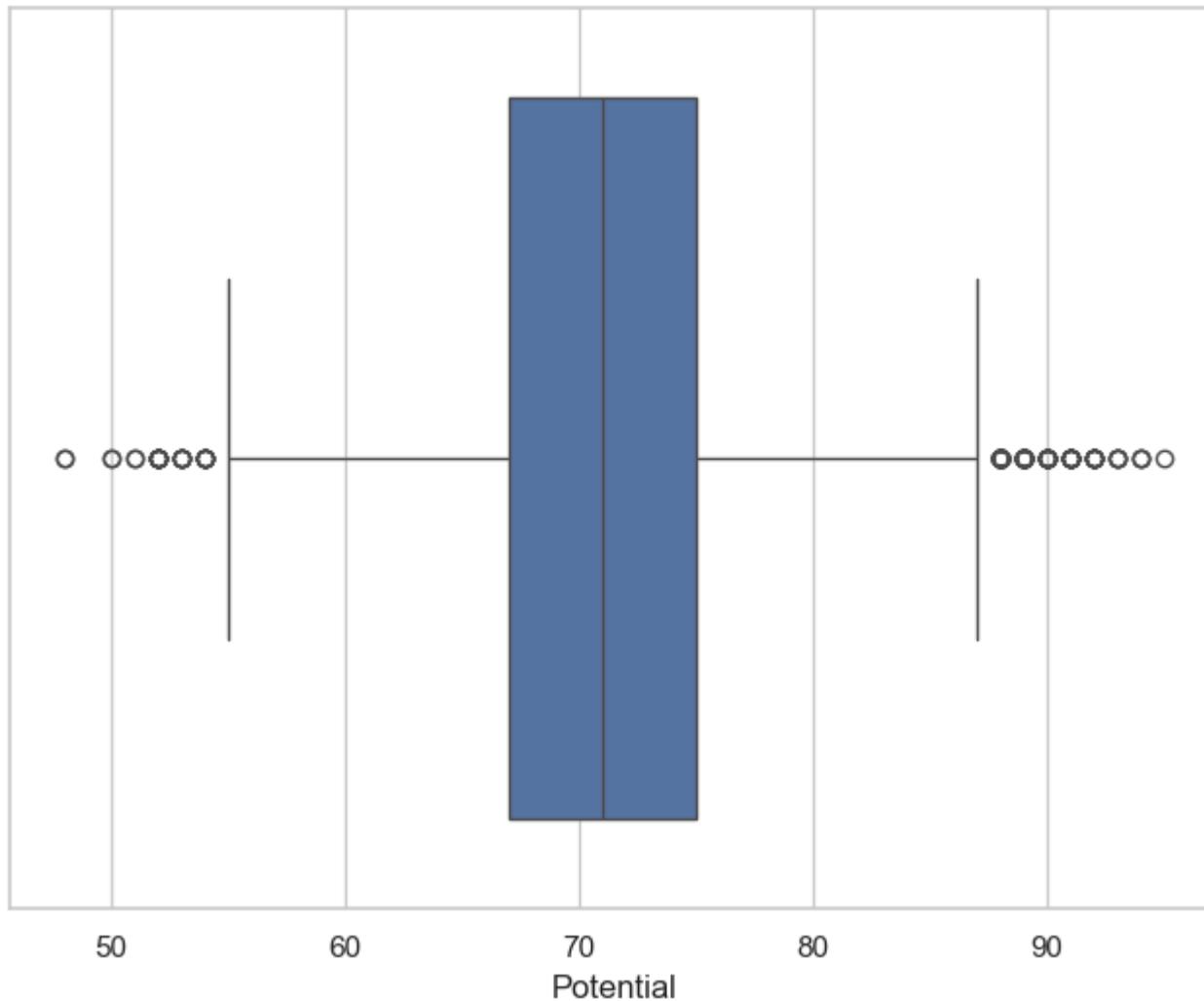


Seaborn `boxplot()` function

- This function draws a box plot to show distributions with respect to categories.
- A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.

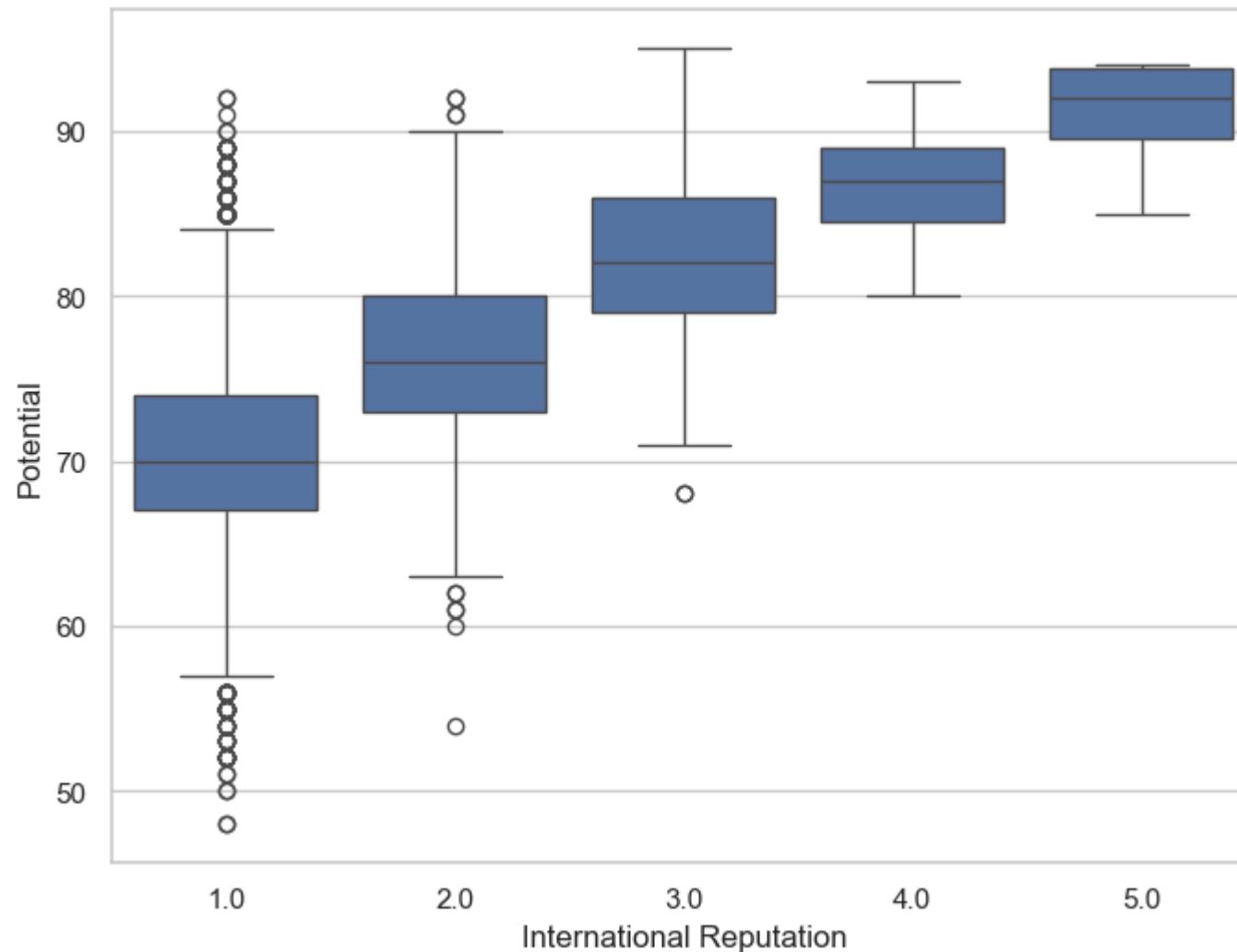
- The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the inter-quartile range.
- I will plot the boxplot of the `Potential` variable as follows-

```
In [29]: f, ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=fifa['Potential'])
plt.show()
```



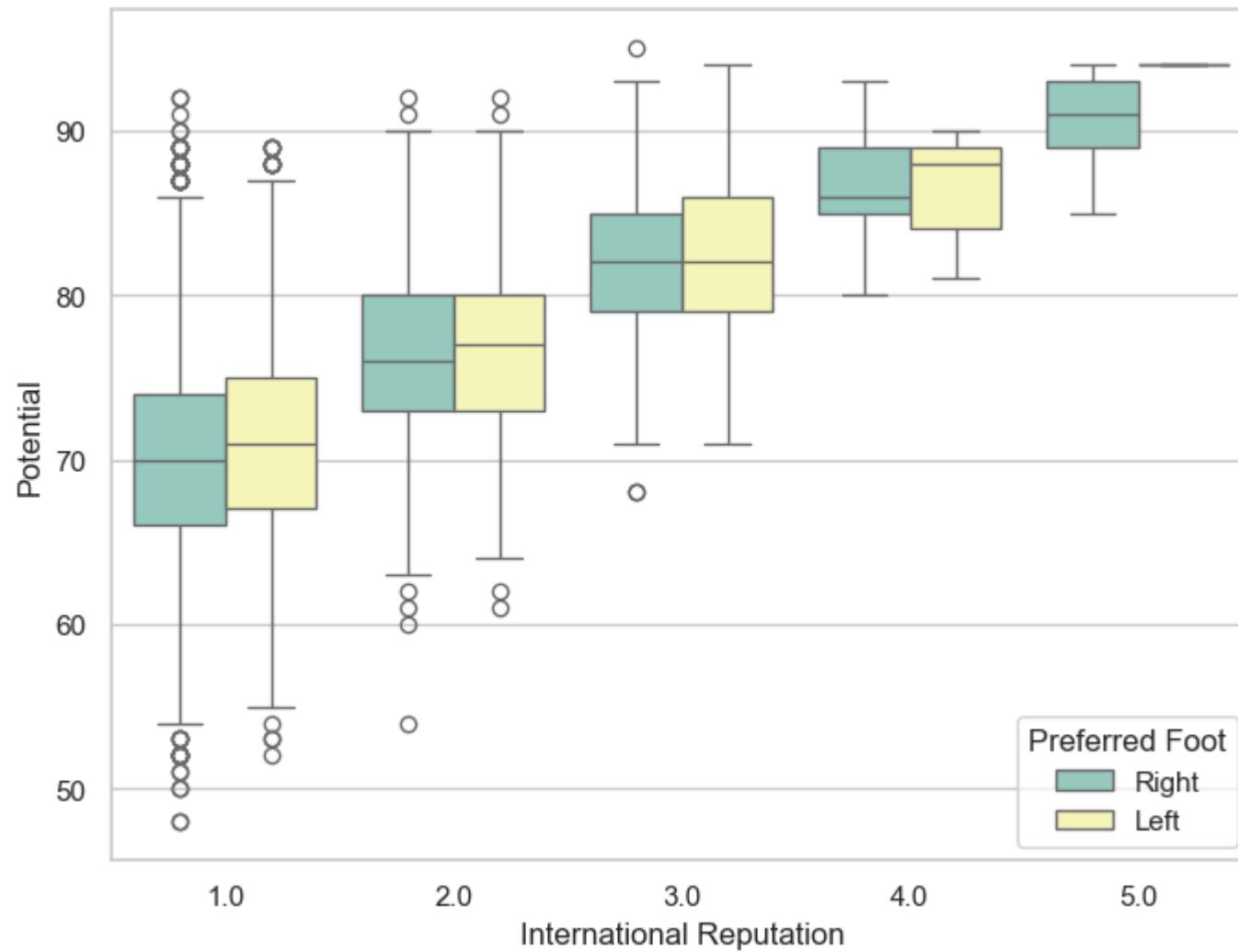
We can draw the vertical boxplot grouped by the categorical variable International Reputation as follows-

```
In [30]: f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="International Reputation", y="Potential", data=fifa)
plt.show()
```



We can draw a boxplot with nested grouping by two categorical variables as follows-

```
In [31]: f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="International Reputation", y="Potential", hue="Preferred Foot", data=fifa, palette="Set3")
plt.show()
```



Seaborn `violinplot()` function

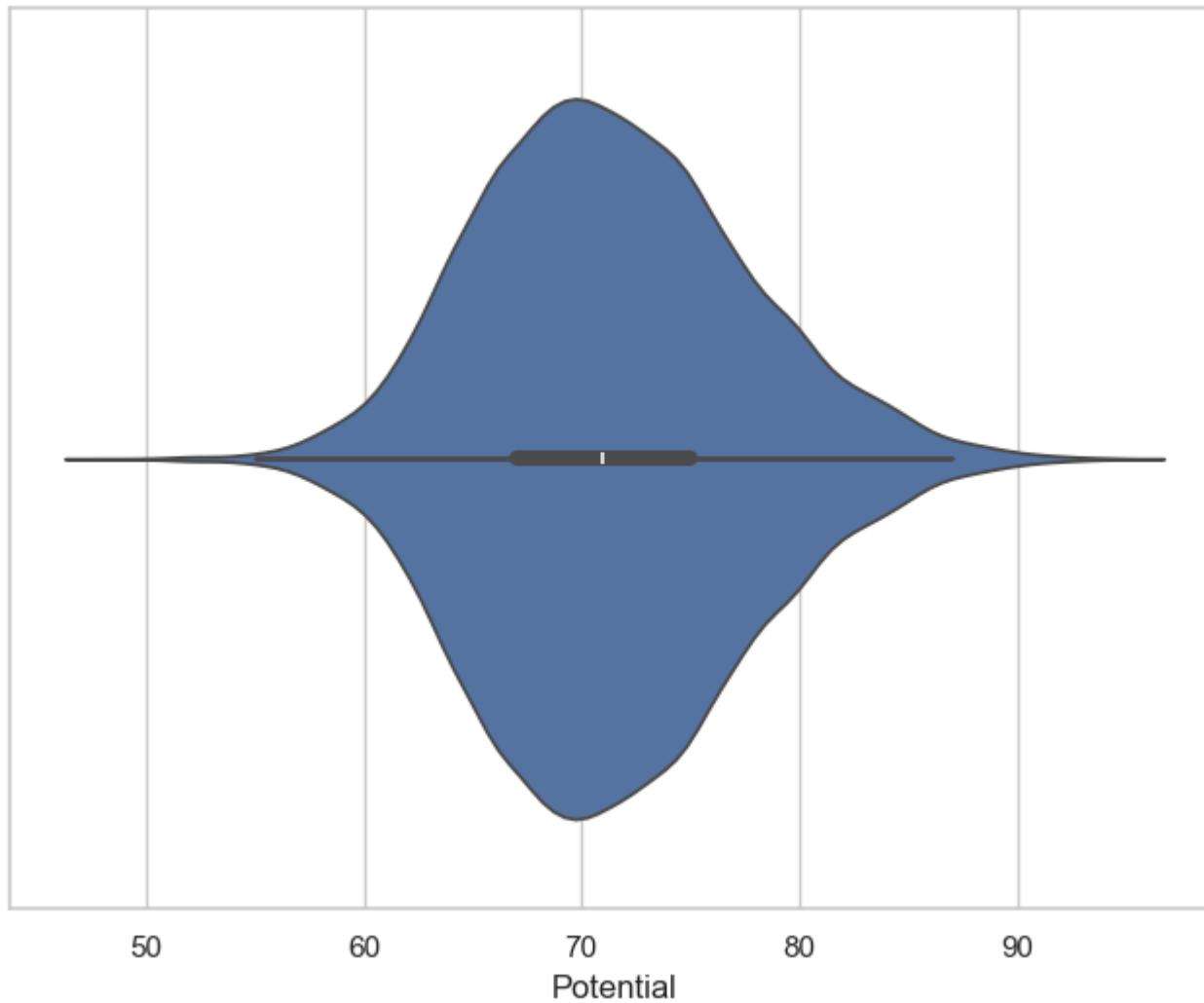
- This function draws a combination of boxplot and kernel density estimate.
- A violin plot plays a similar role as a box and whisker plot.

- It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.
- Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.
- I will plot the violinplot of `Potential` variable as follows-

In [32]:

```
f,ax = plt.subplots(figsize=(8,6))
sns.violinplot(x=fifa['Potential'])
plt.show
```

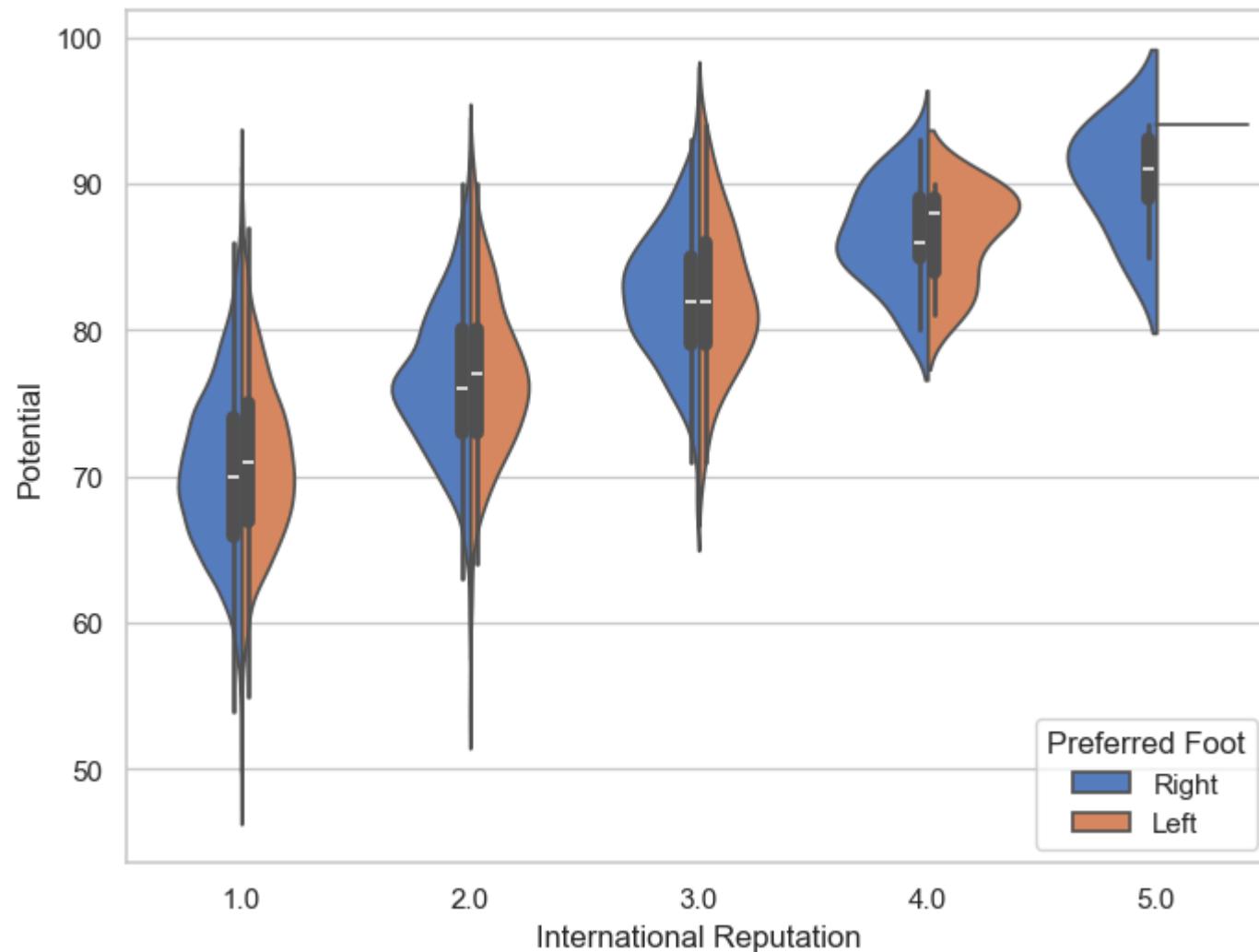
Out[32]: <function matplotlib.pyplot.show(close=None, block=None)>



We can draw the vertical violinplot grouped by the categorical variable International Reputation as follows-

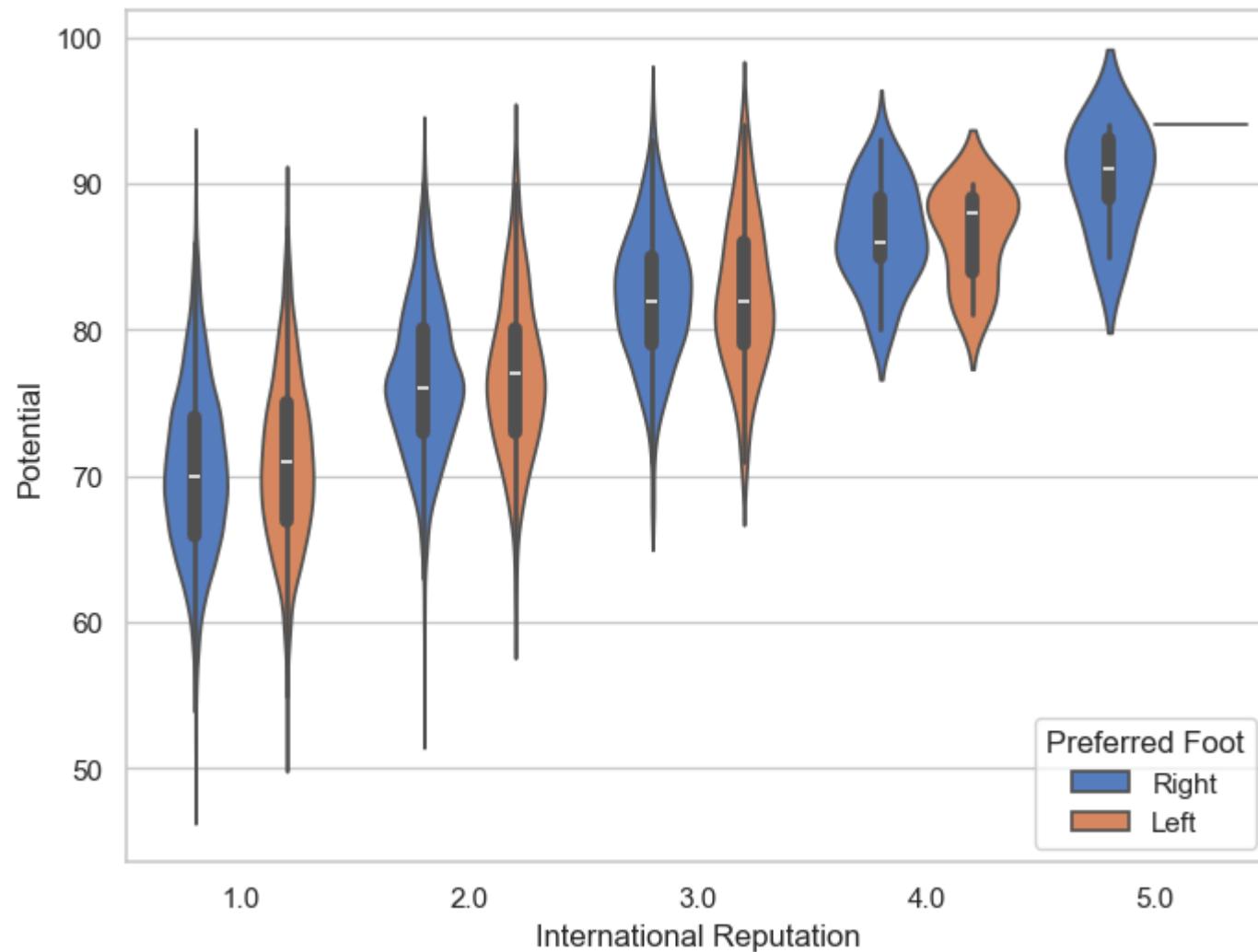
```
In [33]: f,ax = plt.subplots(figsize=(8,6))
sns.violinplot(x='International Reputation',y='Potential',hue="Preferred Foot",data=fifa,palette="muted",split=True)
plt.show()
```

```
Out[33]: <function matplotlib.pyplot.show(close=None, block=None)>
```



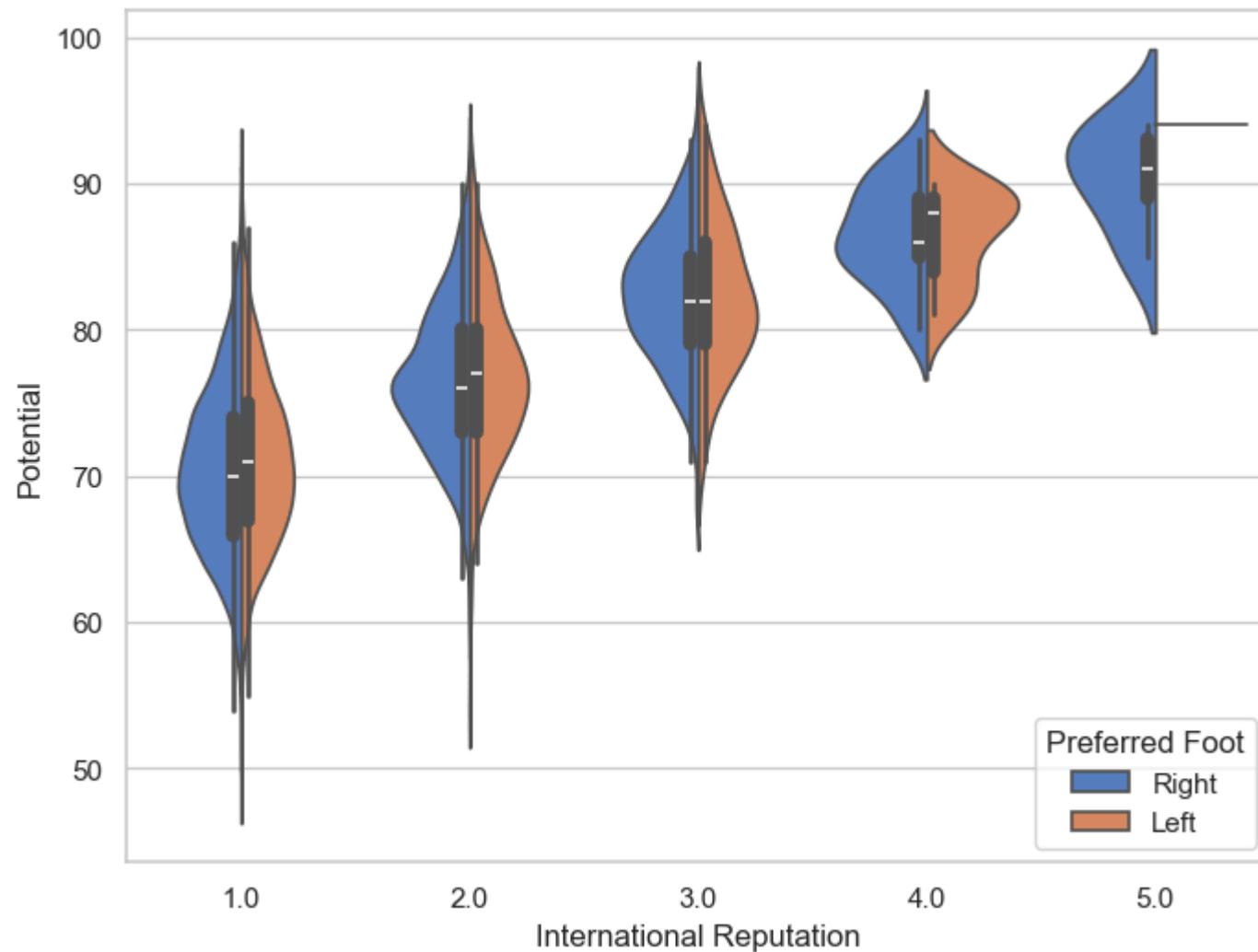
We can draw a violinplot with nested grouping by two categorical variables as follows-

```
In [34]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x="International Reputation", y="Potential", hue="Preferred Foot", data=fifa, palette="muted")
plt.show()
```



We can draw split violins to compare across the hue variable as follows-

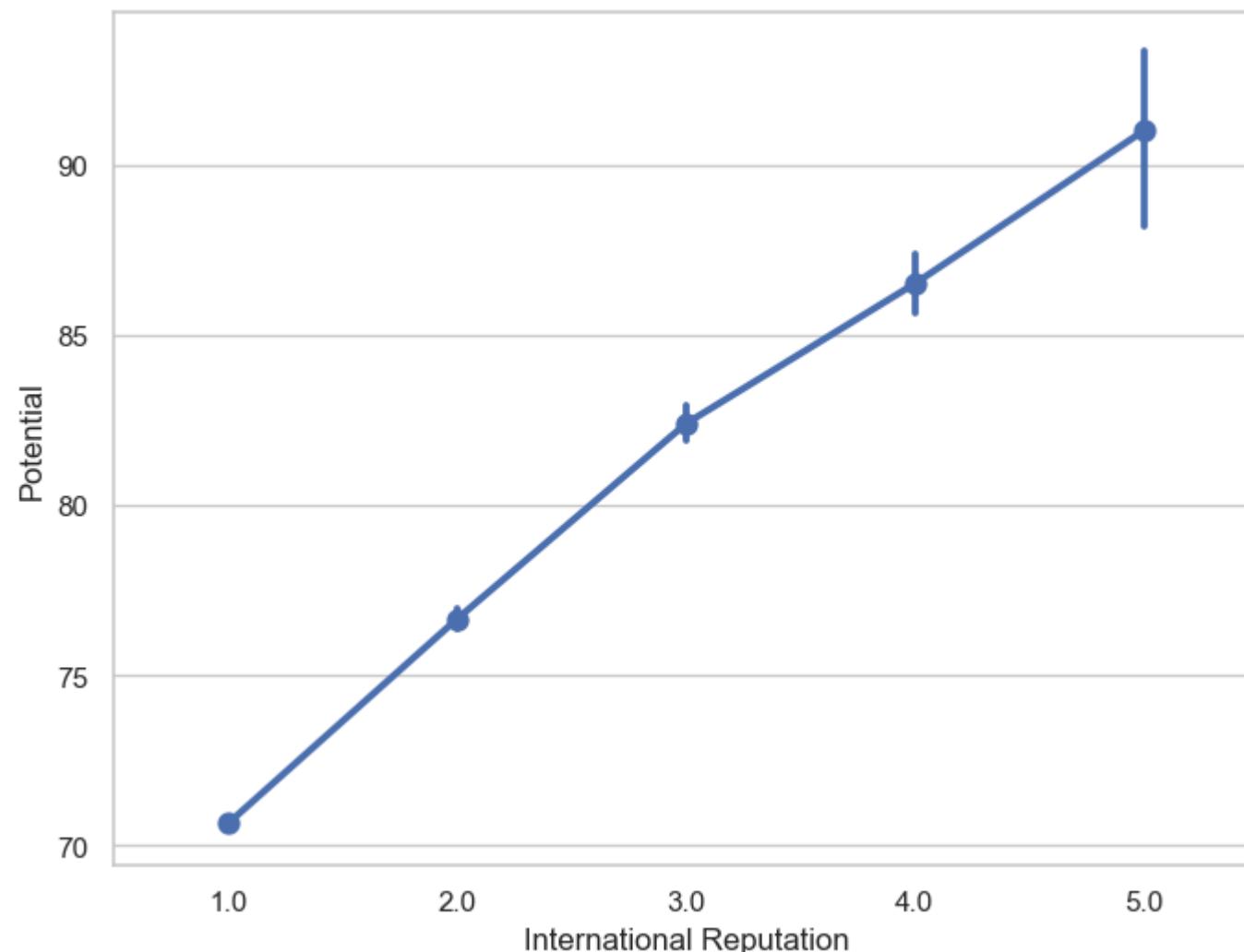
```
In [35]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x="International Reputation", y="Potential", hue="Preferred Foot",
                 data=fifa, palette="muted", split=True)
plt.show()
```



Seaborn `pointplot()` function

- This function shows point estimates and confidence intervals using scatter plot glyphs.
- A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.

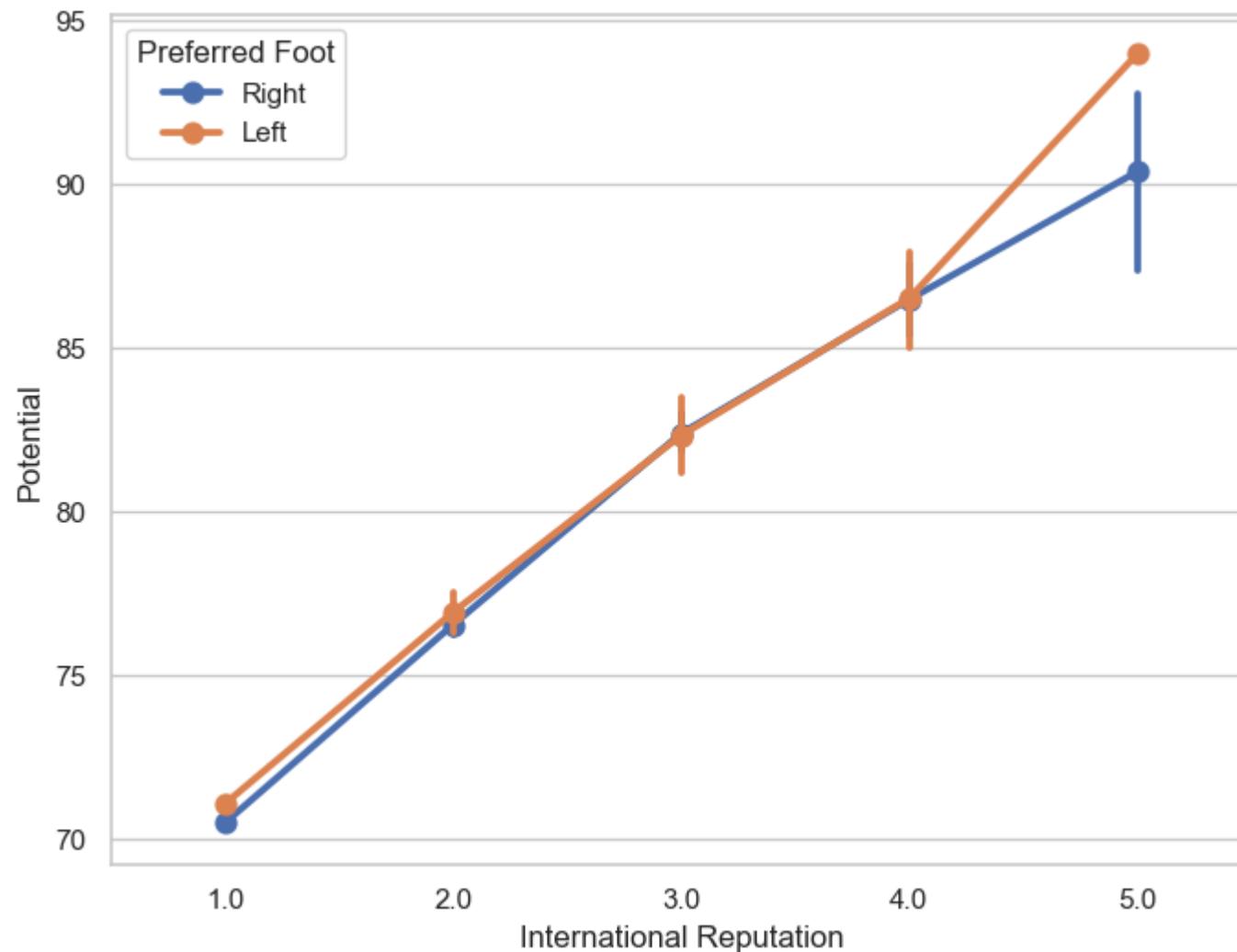
```
In [39]: f, aax = plt.subplots(figsize=(8, 6))
sns.pointplot(x='International Reputation',y='Potential',data=fifa)
plt.show()
```



We can draw a set of vertical points with nested grouping by a two variables as follows-

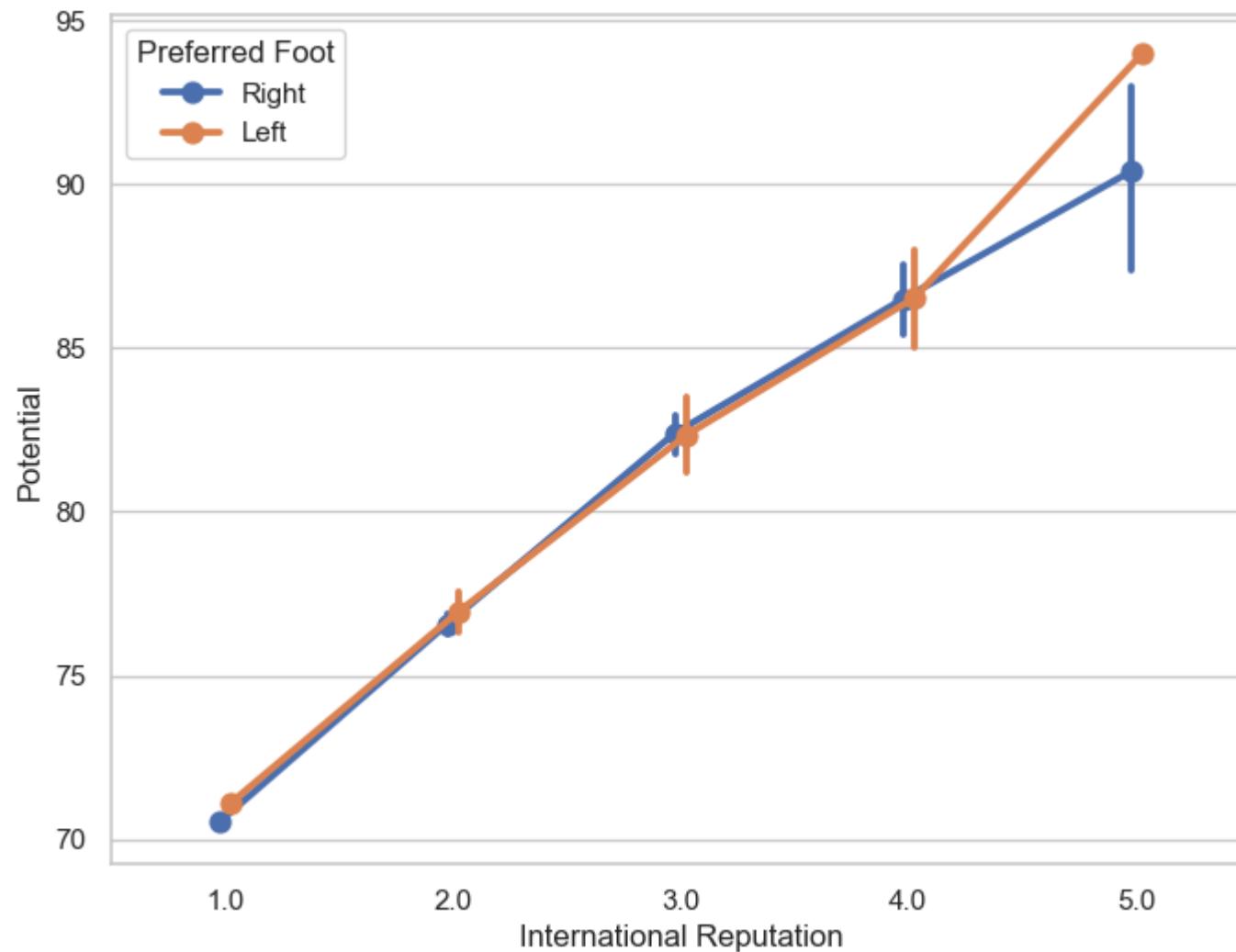
```
In [40]: f, aax = plt.subplots(figsize=(8, 6))
sns.pointplot(x='International Reputation',y='Potential',hue='Preferred Foot',data=fifa)
```

```
plt.show()
```

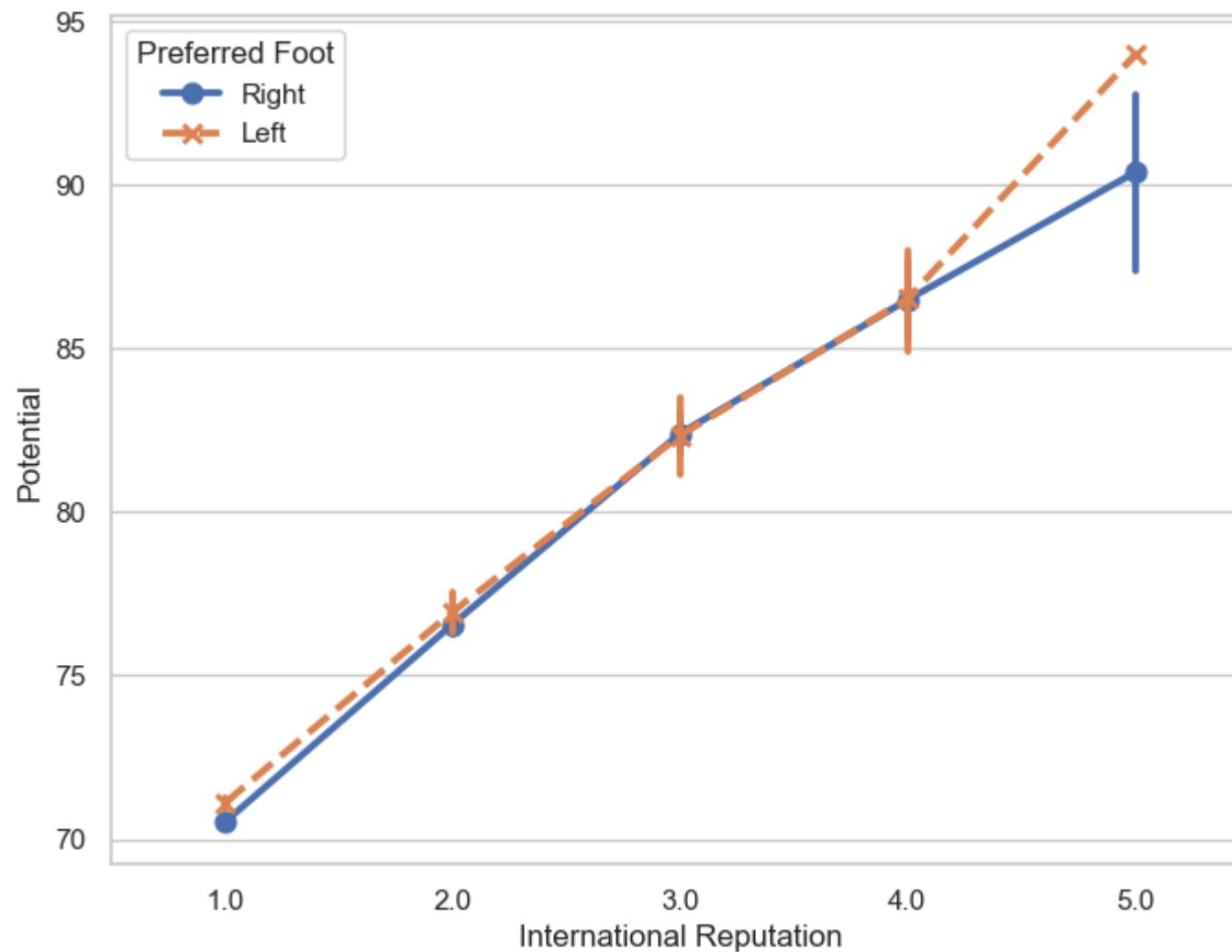


We can separate the points for different hue levels along the categorical axis as follows-

```
In [42]: f, aax = plt.subplots(figsize=(8, 6))
sns.pointplot(x='International Reputation', y='Potential', hue='Preferred Foot', data=fifa, dodge=True)
plt.show()
```



```
In [44]: f, ax = plt.subplots(figsize=(8, 6))
sns.pointplot(x="International Reputation", y="Potential", hue="Preferred Foot",
               data=fifa, markers=["o", "x"], linestyles=[ "-", "--"])
plt.show()
```

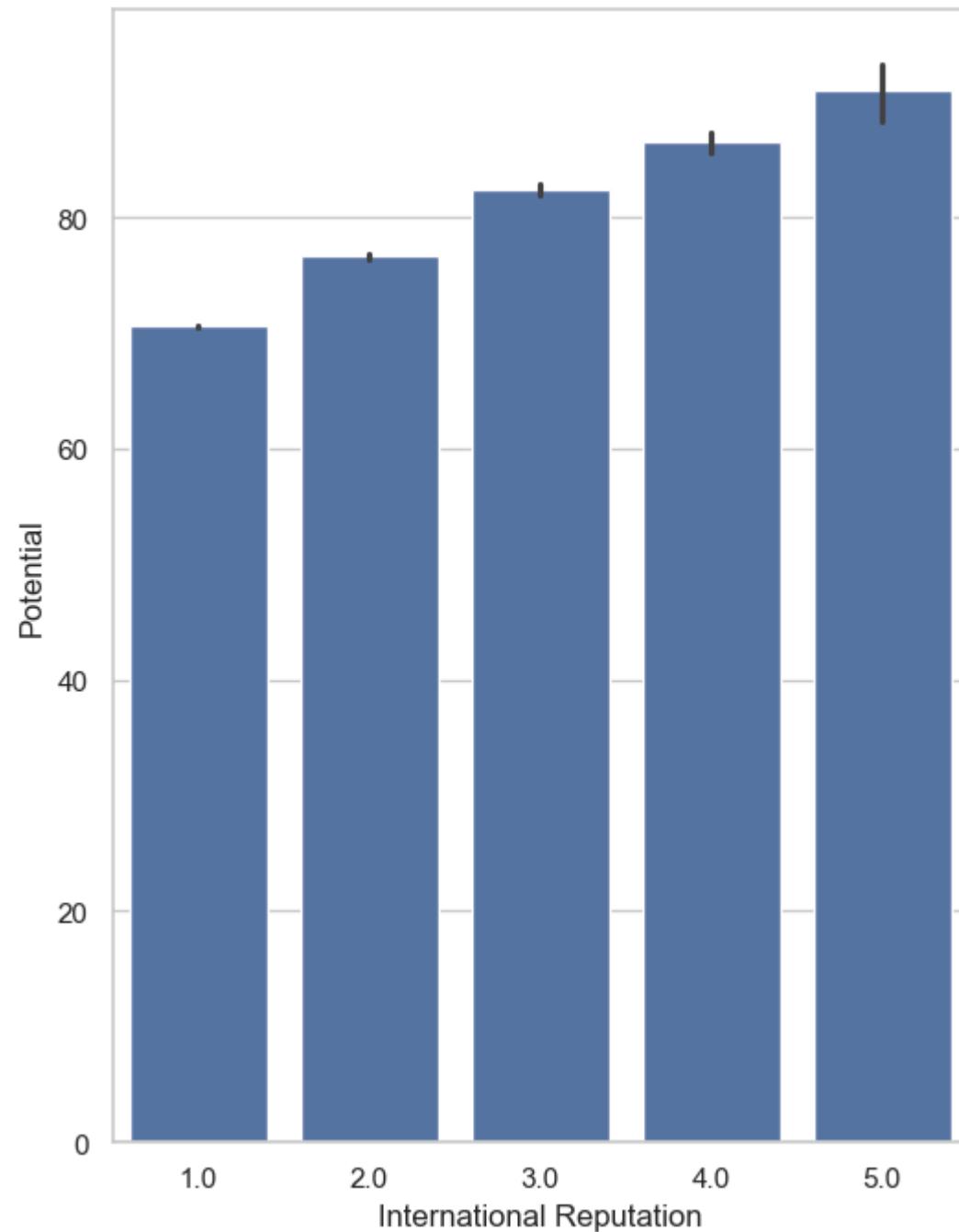


Seaborn `barplot()` function

- This function shows point estimates and confidence intervals as rectangular bars.
- A bar plot represents an estimate of central tendency for a numeric variable with the height of each rectangle and provides some indication of the uncertainty around that estimate using error bars.

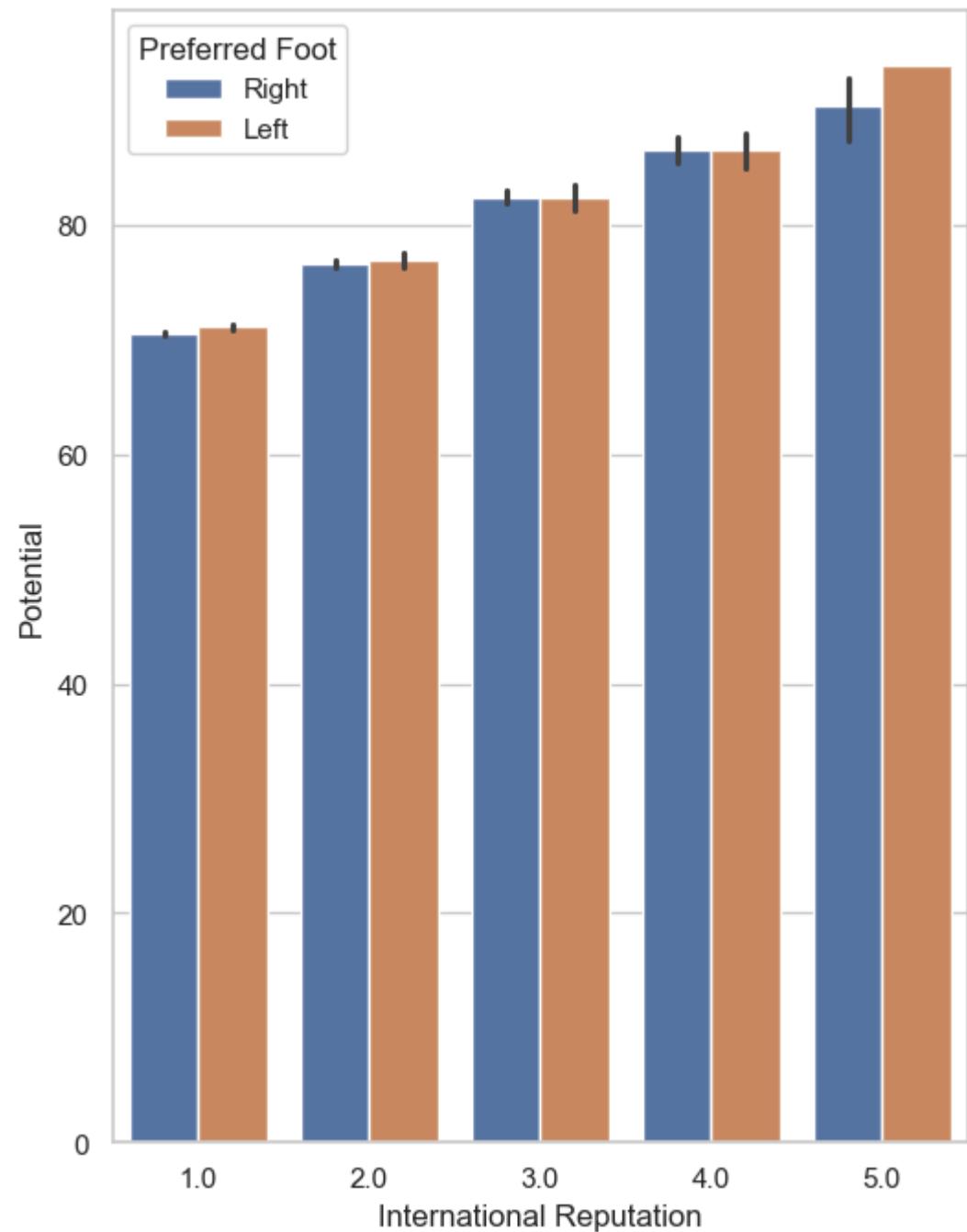
- Bar plots include 0 in the quantitative axis range, and they are a good choice when 0 is a meaningful value for the quantitative variable, and you want to make comparisons against it.
- We can plot a barplot as follows-

```
In [46]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", data=fifa)
plt.show()
```



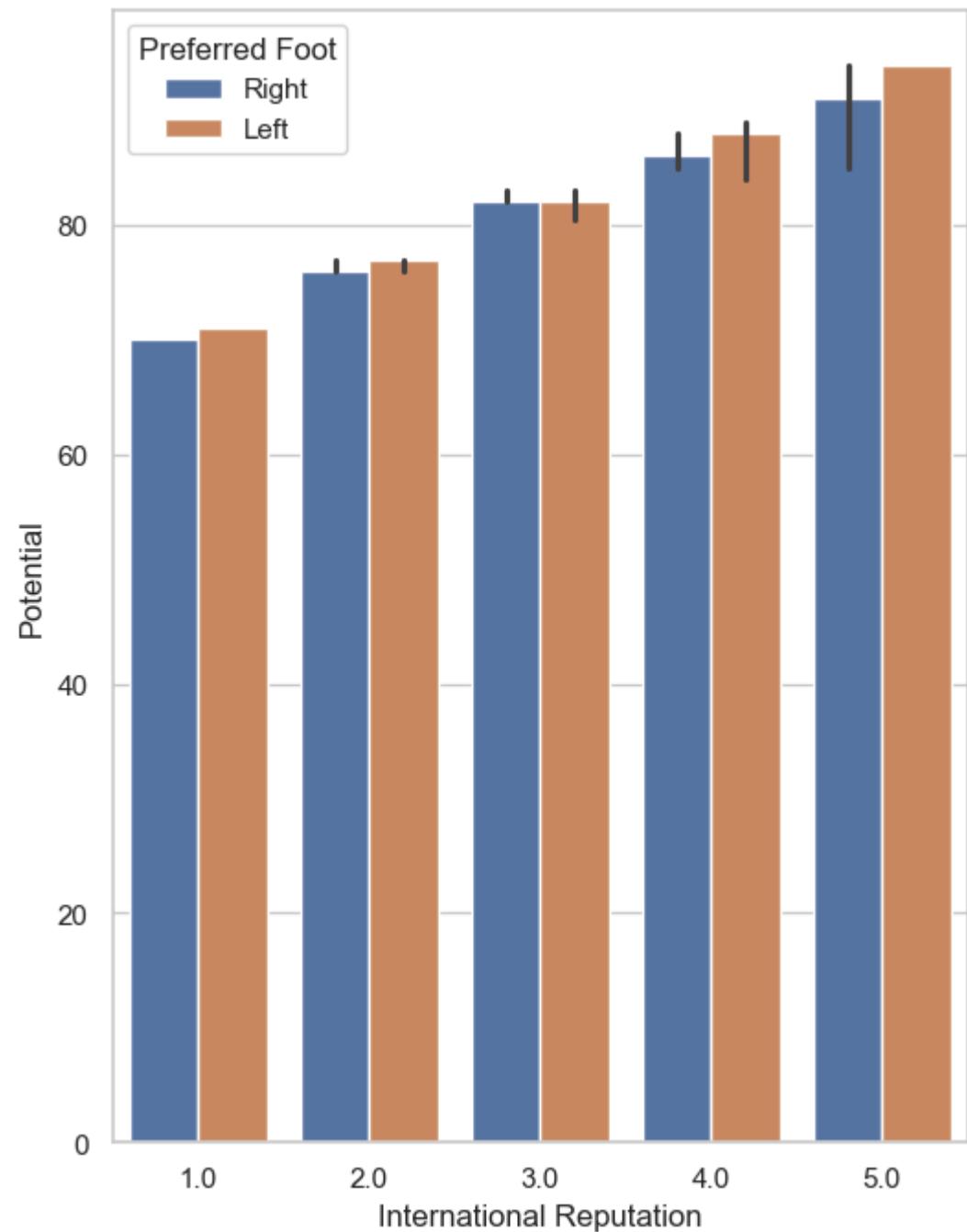
We can draw a set of vertical bars with nested grouping by a two variables as follows-

```
In [47]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", hue='Preferred Foot', data=fifa)
plt.show()
```

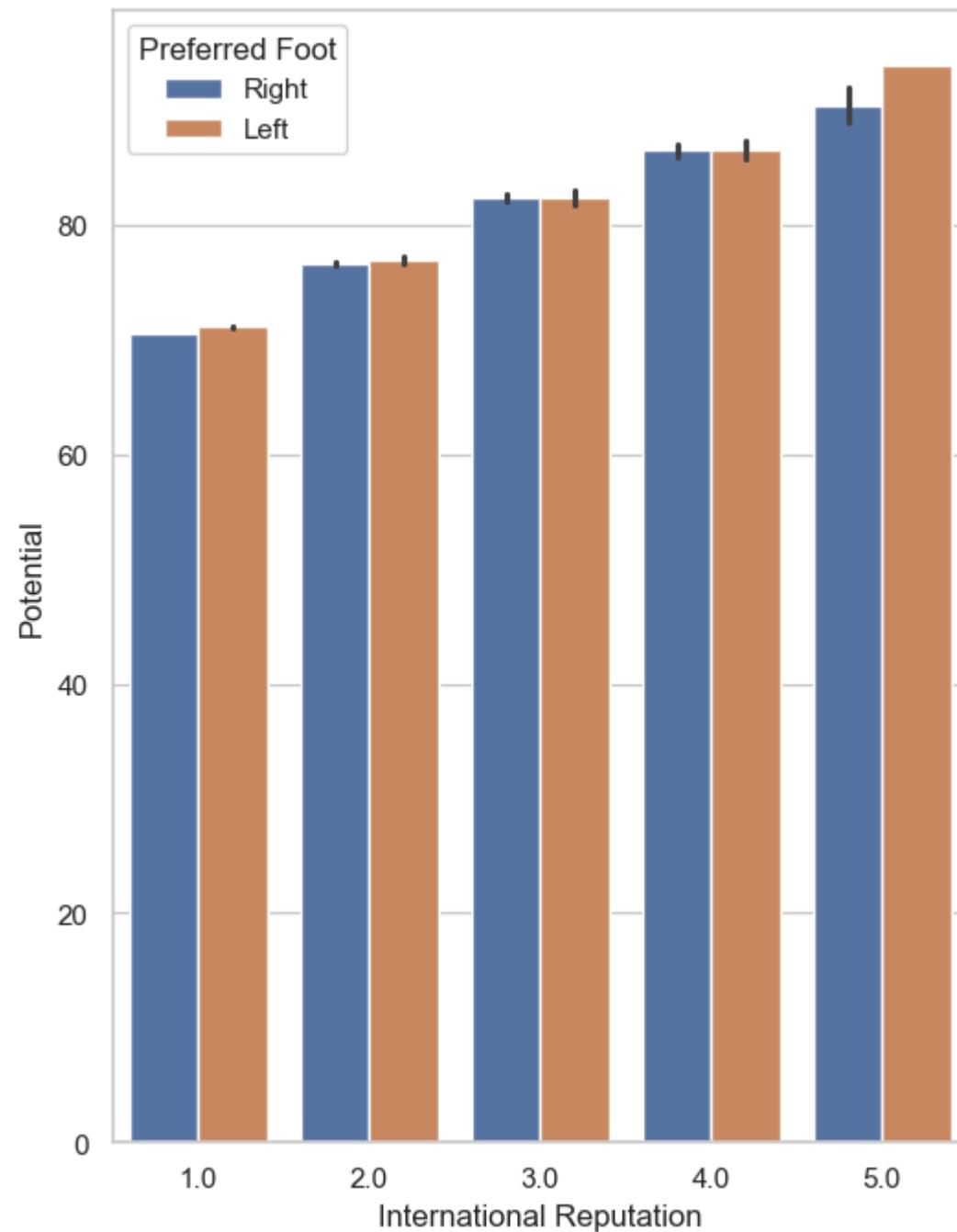


We can use median as the estimate of central tendency as follows-

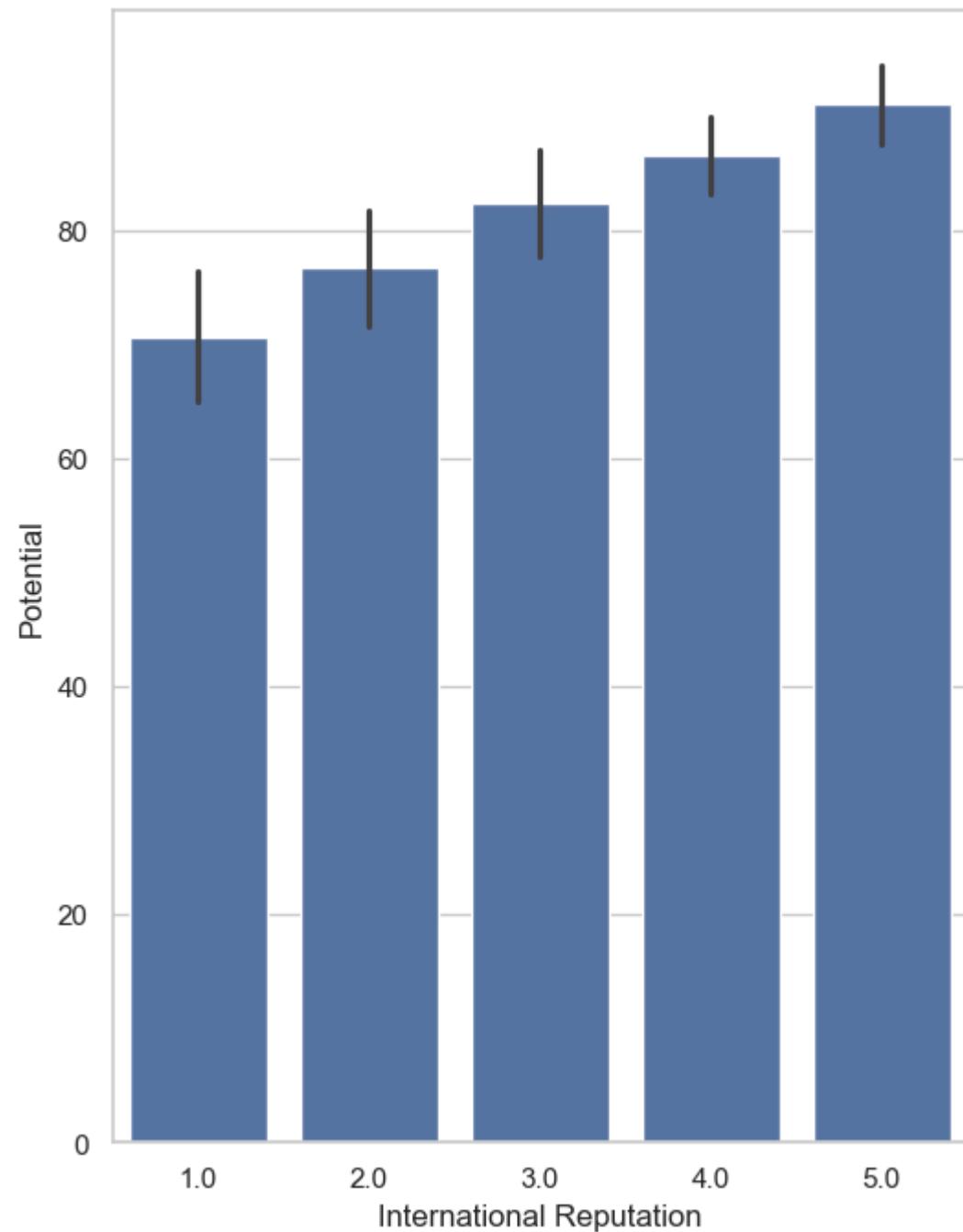
```
In [50]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", hue='Preferred Foot', data=fifa, estimator='median'
            )
plt.show()
```



```
In [52]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", hue='Preferred Foot', data=fifa, ci=68)
plt.show()
```

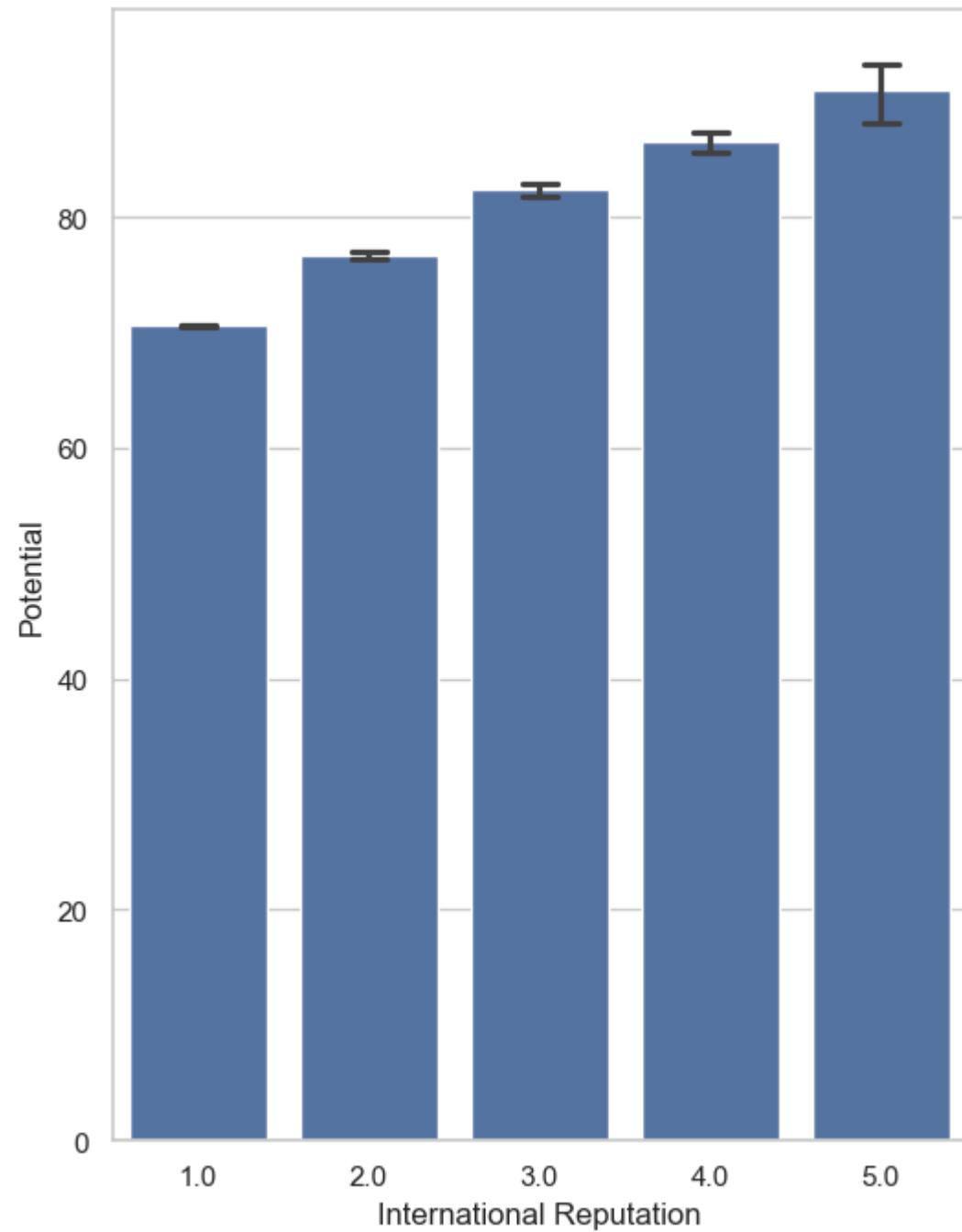


```
In [54]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", data=fifa, ci="sd")
plt.show()
```



We can add "caps" to the error bars as follows-

```
In [56]: f, ax=plt.subplots(figsize=(6,8))
sns.barplot(x="International Reputation", y="Potential", data=fifa, capsize=0.2)
plt.show()
```



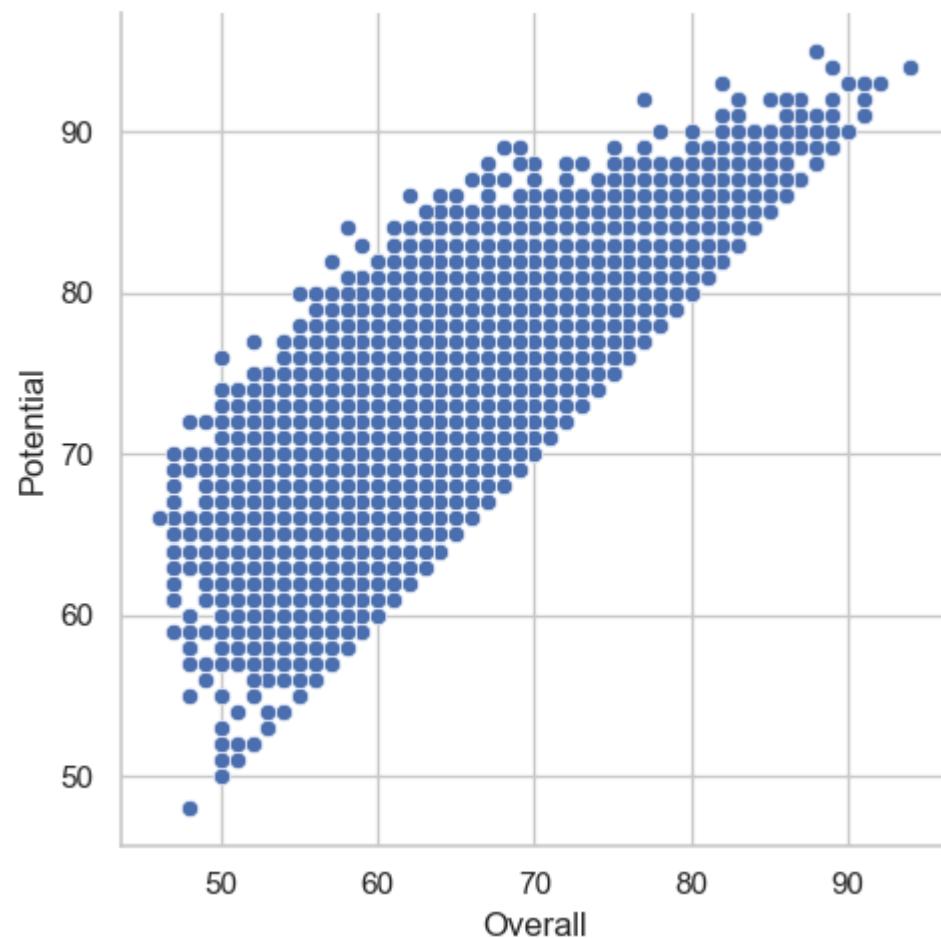
Visualizing statistical relationship with Seaborn `relplot()` function

Seaborn `relplot()` function

- Seaborn `relplot()` function helps us to draw figure-level interface for drawing relational plots onto a FacetGrid.
- This function provides access to several different axes-level functions that show the relationship between two variables with semantic mappings of subsets.
- The `kind` parameter selects the underlying axes-level function to use-
- `scatterplot()` (with `kind="scatter"`; the default)
- `lineplot()` (with `kind="line"`)

We can plot a scatterplot with variables Height and Weight with Seaborn `relplot()` function as follows-

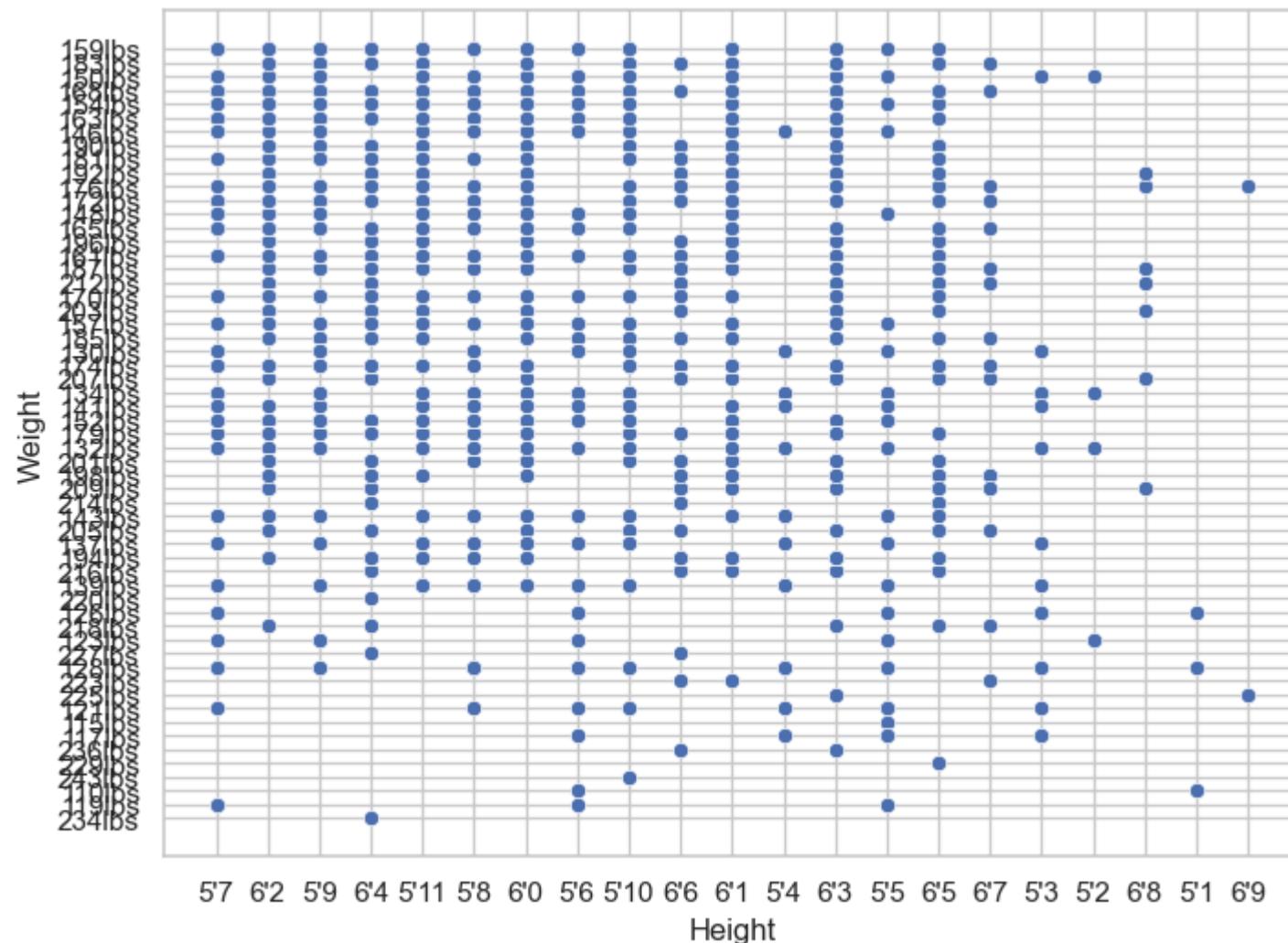
```
In [57]: g = sns.relplot(x='Overall',y='Potential',data=fifa)
```



Seaborn `scatterplot()` function

- This function draws a scatter plot with possibility of several semantic groups.
- The relationship between x and y can be shown for different subsets of the data using the `hue`, `size` and `style` parameters.
- These parameters control what visual semantics are used to identify the different subsets.

```
In [59]: f, ax = plt.subplots(figsize=(8, 6))
sns.scatterplot(x="Height", y="Weight", data=fifa)
plt.show()
```

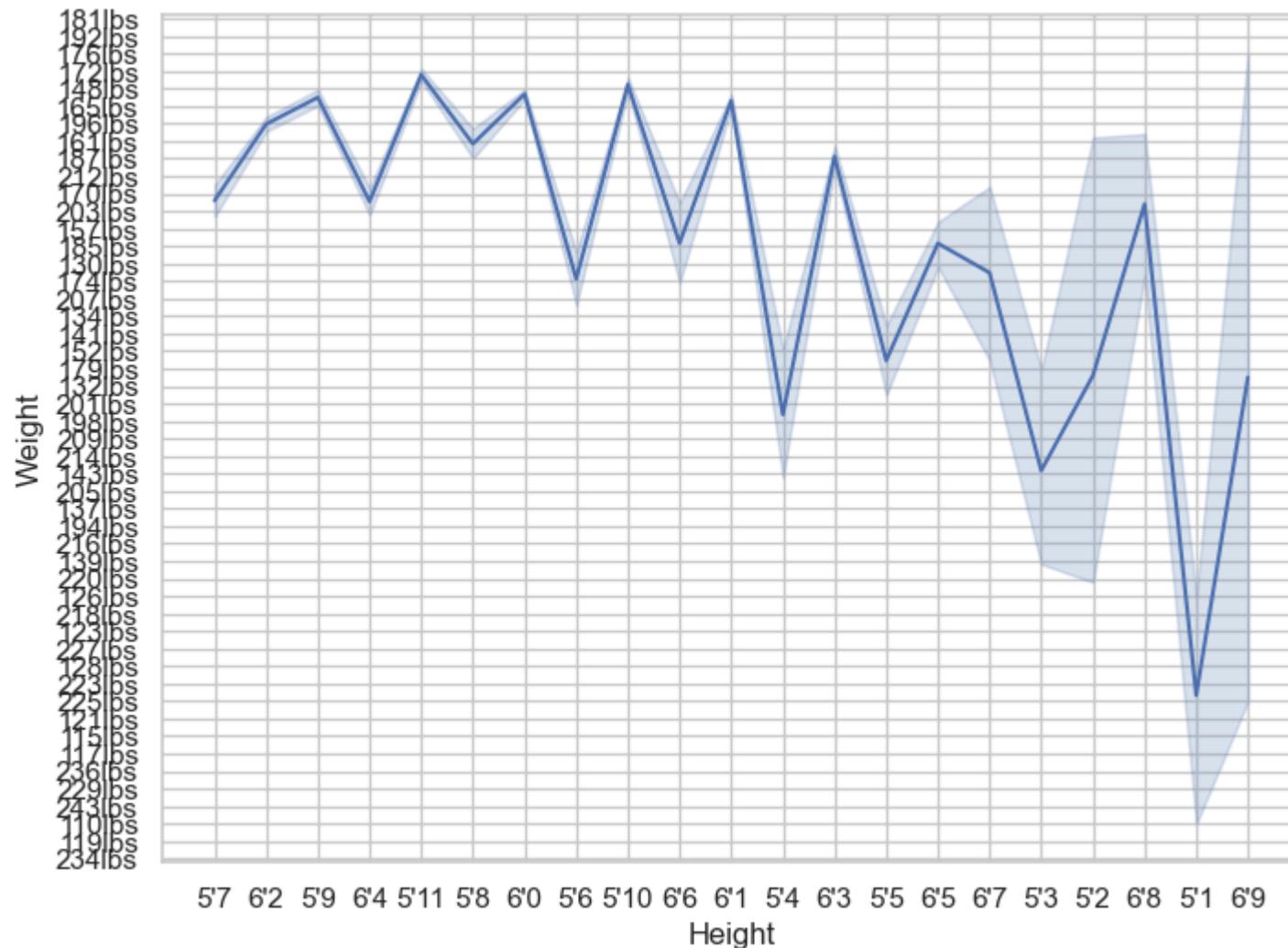


Seaborn lineplot() function

- This function draws a line plot with possibility of several semantic groupings.

- The relationship between x and y can be shown for different subsets of the data using the `hue`, `size` and `style` parameters.
- These parameters control what visual semantics are used to identify the different subsets.

```
In [61]: f, ax = plt.subplots(figsize=(8, 6))
sns.lineplot(x="Height", y="Weight", data=fifa)
plt.show()
```



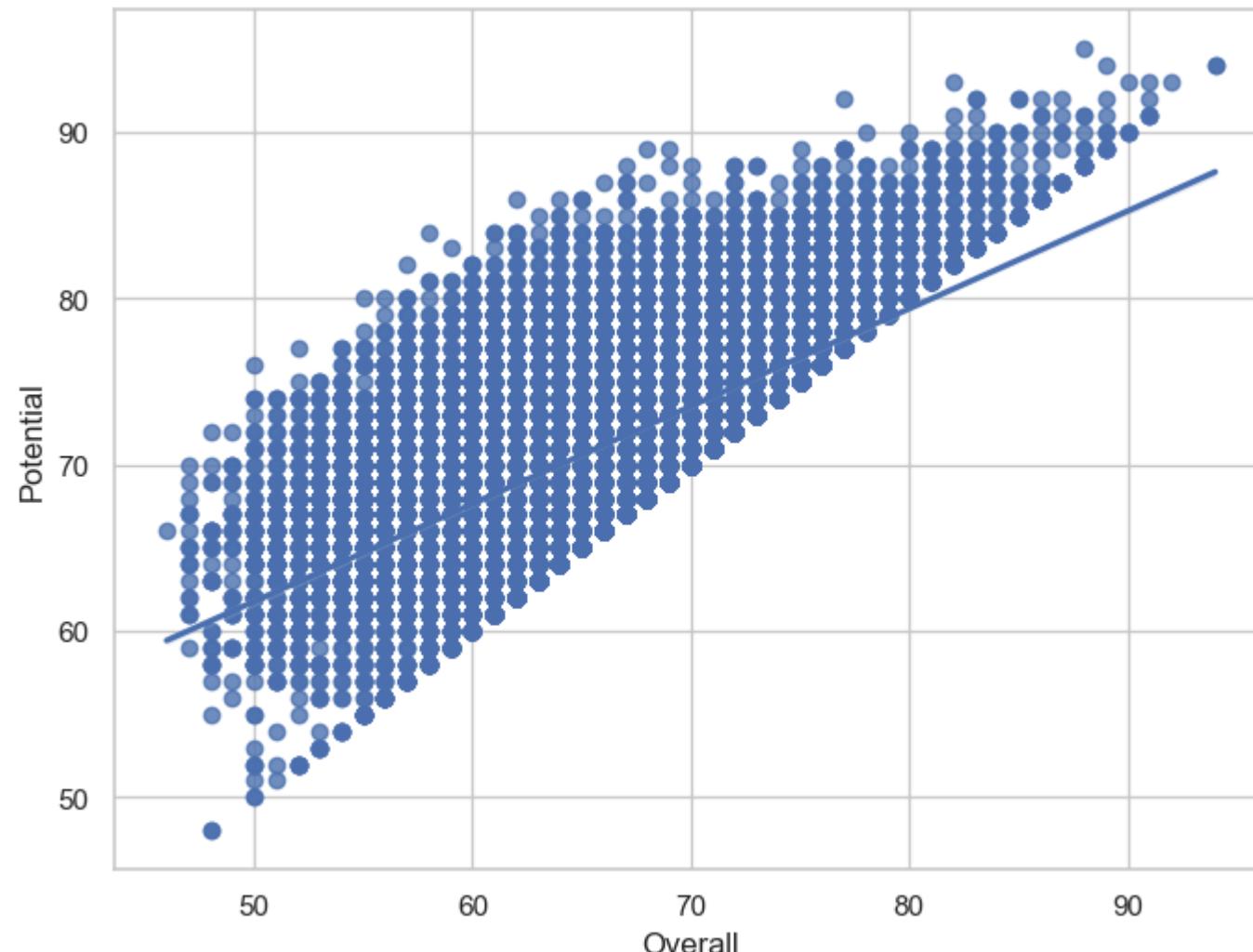
Visualize linear relationship with Seaborn regplot() function

Seaborn regplot() function

- This function plots data and a linear regression model fit.

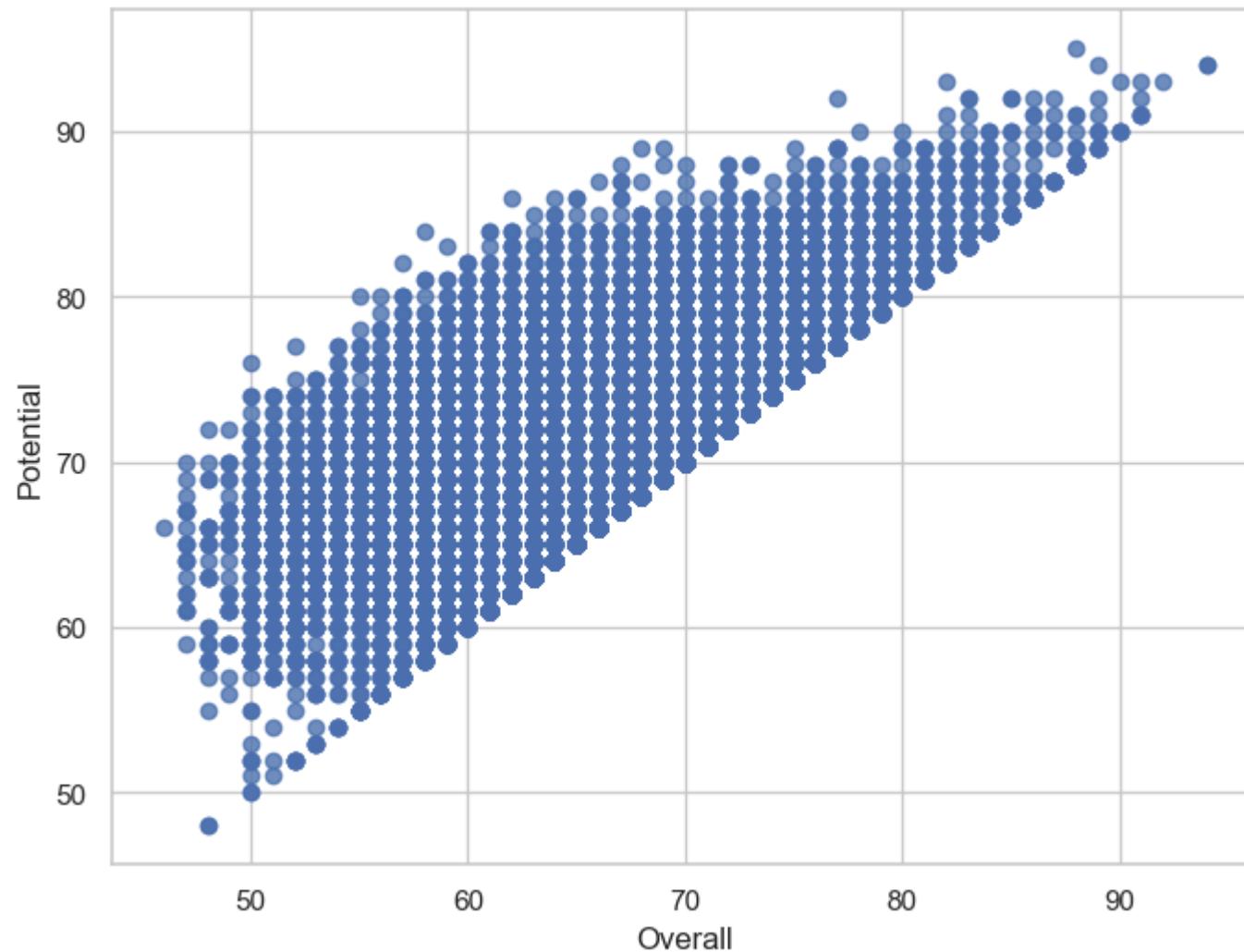
- We can plot a linear regression model between Overall and Potential variable with regplot() function as follows-

```
In [65]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=fifa, fit_reg=True)
plt.show()
```



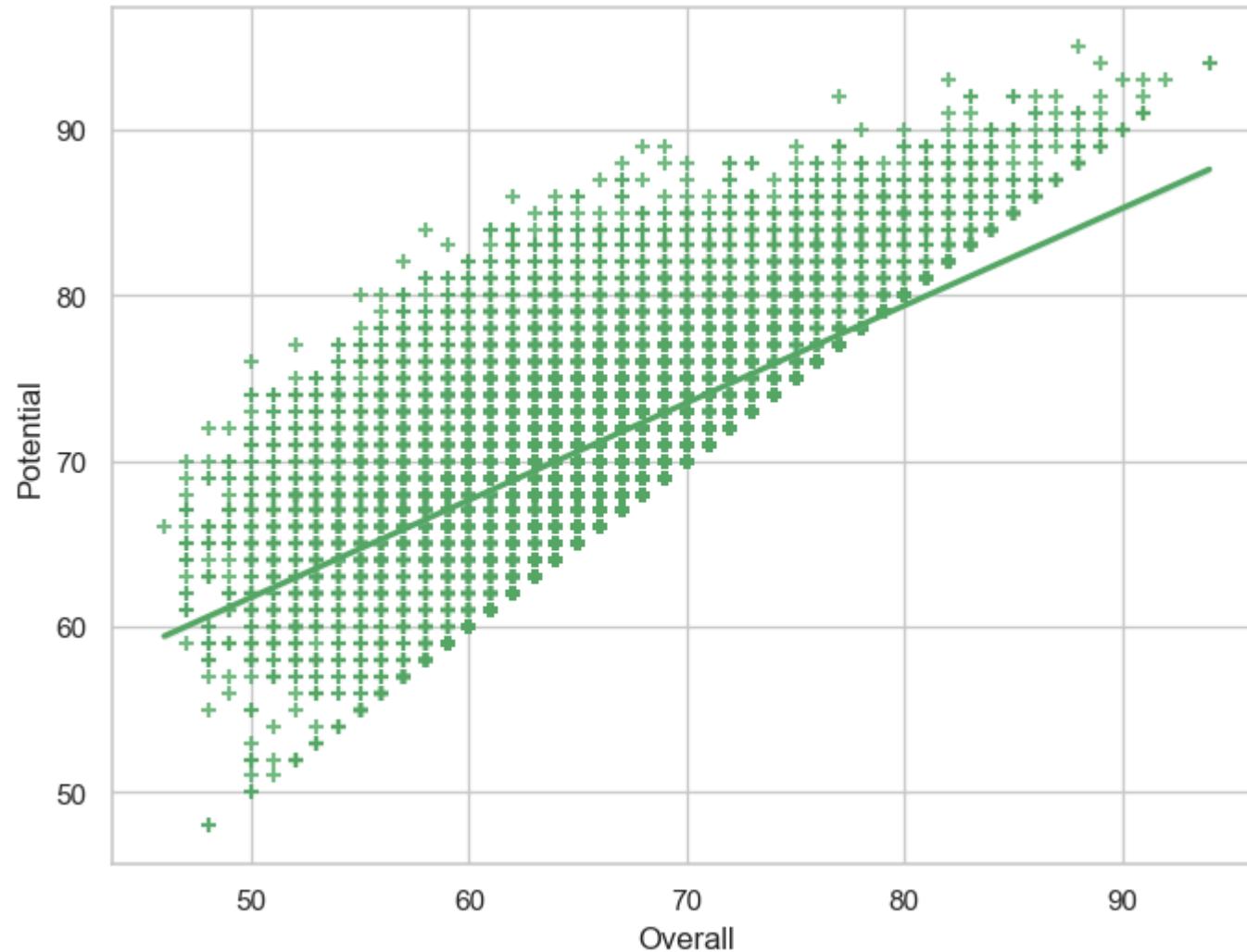
```
In [66]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=fifa, fit_reg=False)
```

```
plt.show()
```



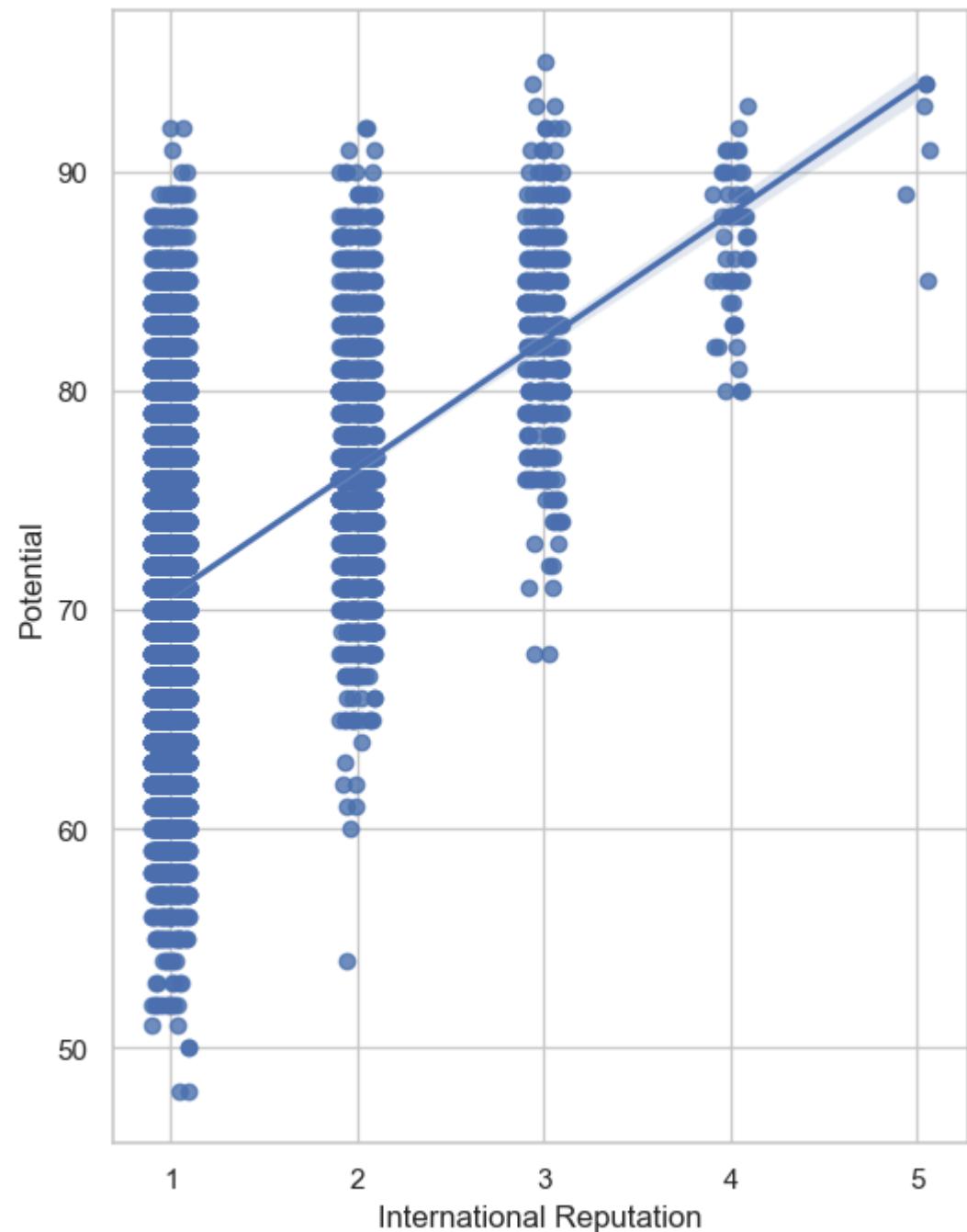
We can use a different color and marker as follows-

```
In [69]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=fifa, color="g", marker="+")
plt.show()
```



We can plot with a discrete variable and add some jitter as follows-

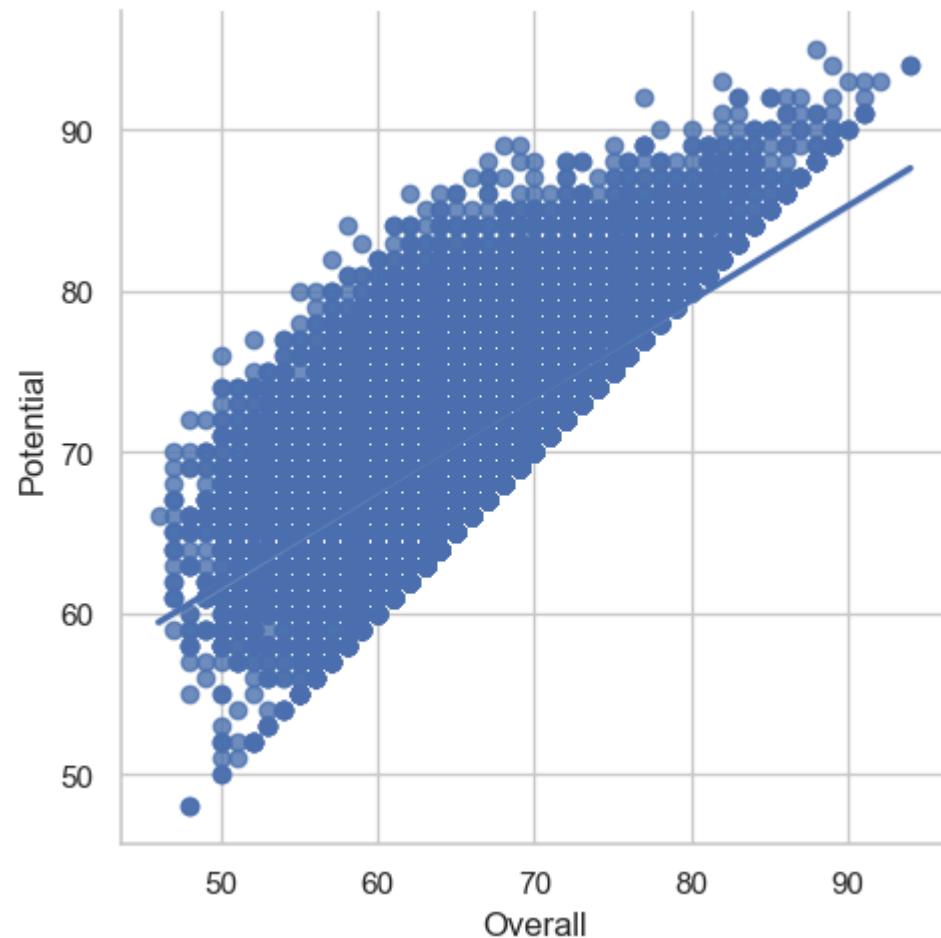
```
In [70]: f, ax=plt.subplots(figsize=(6,8))
ax= sns.regplot(x='International Reputation',y="Potential",data=fifa,x_jitter=0.1)
plt.show()
```



Seaborn lmplot() function

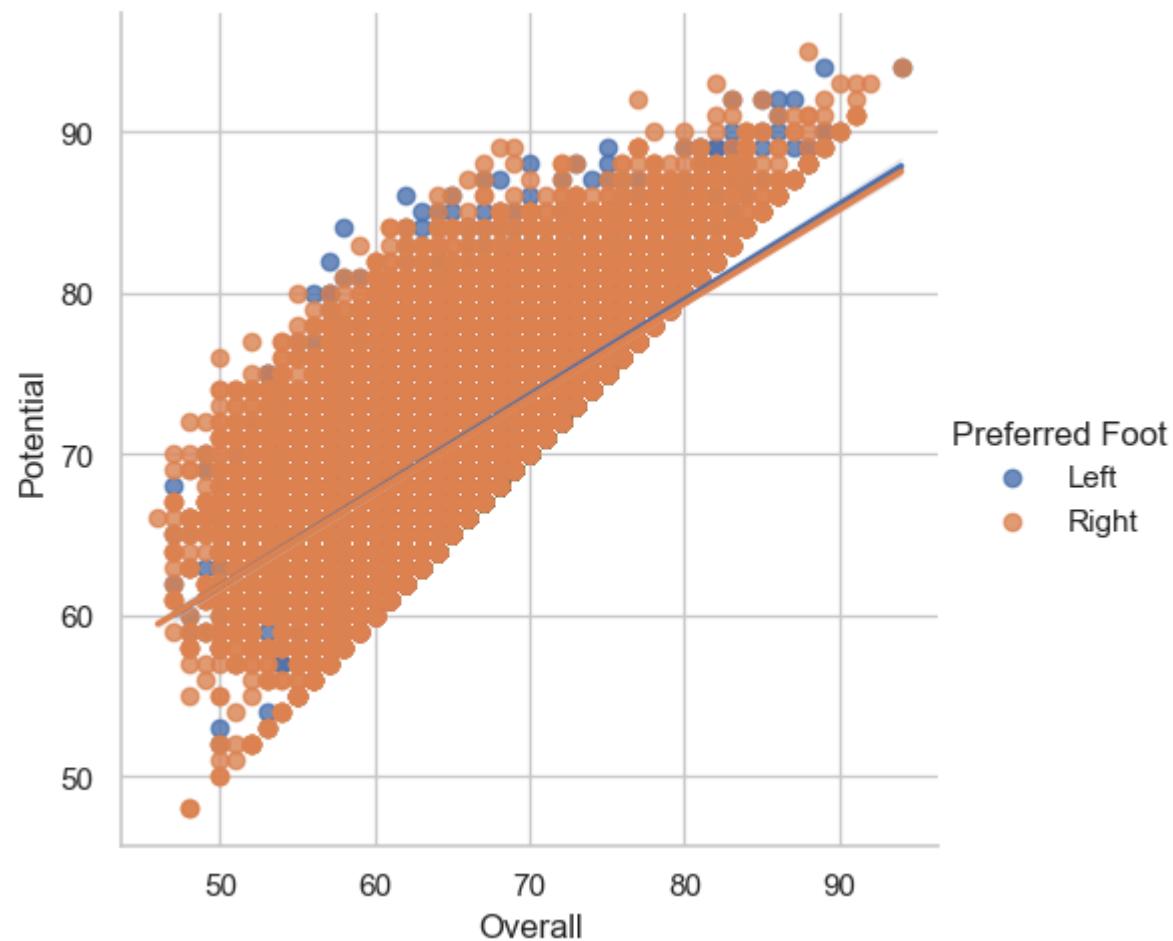
- This function plots data and regression model fits across a FacetGrid.
- This function combines `regplot()` and `FacetGrid`.
- It is intended as a convenient interface to fit regression models across conditional subsets of a dataset.
- We can plot a linear regression model between `Overall` and `Potential` variable with `lmplot()` function as follows-

```
In [72]: g= sns.lmplot(x="Overall",y="Potential",data=fifa)
```



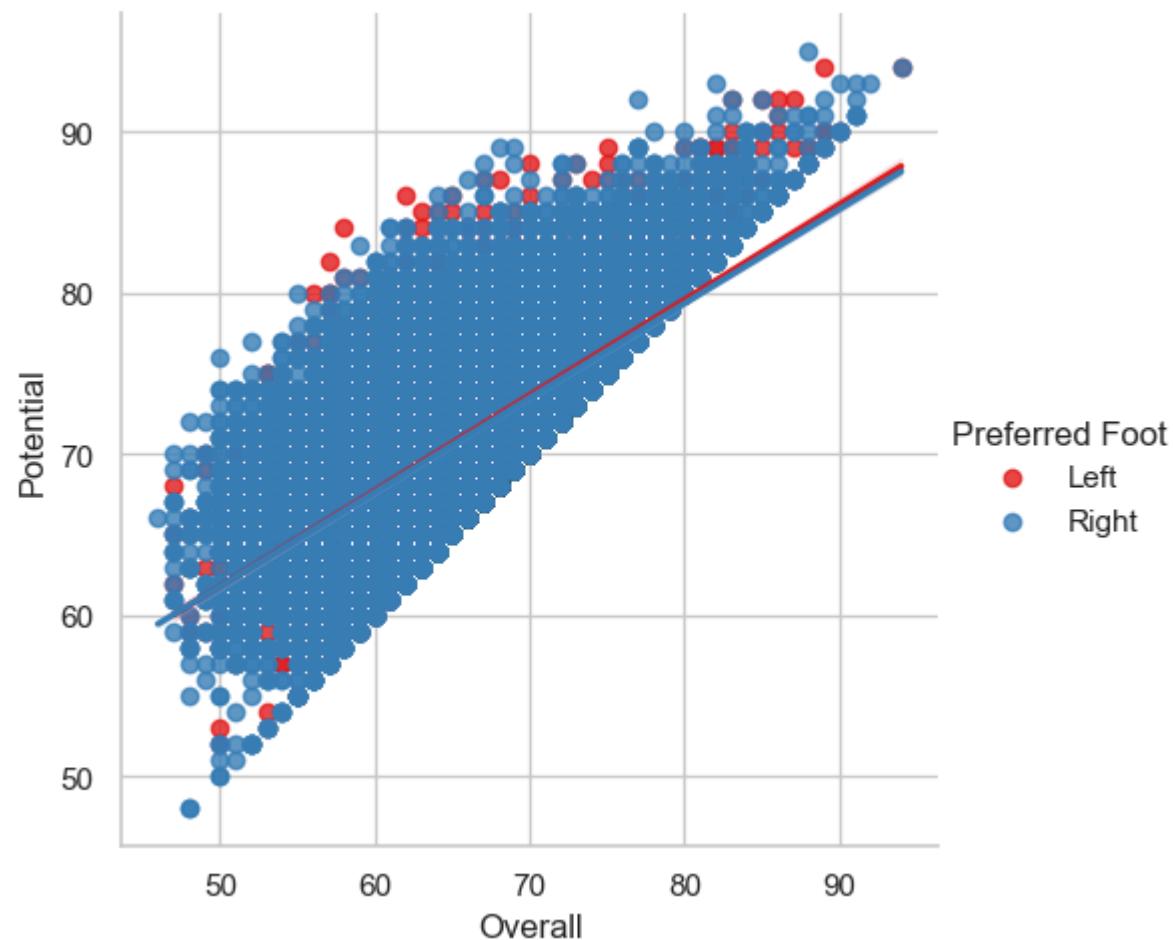
We can condition on a third variable and plot the levels in different colors as follows-

```
In [75]: g = sns.lmplot(x="Overall",y="Potential",hue="Preferred Foot",data=fifa)
```



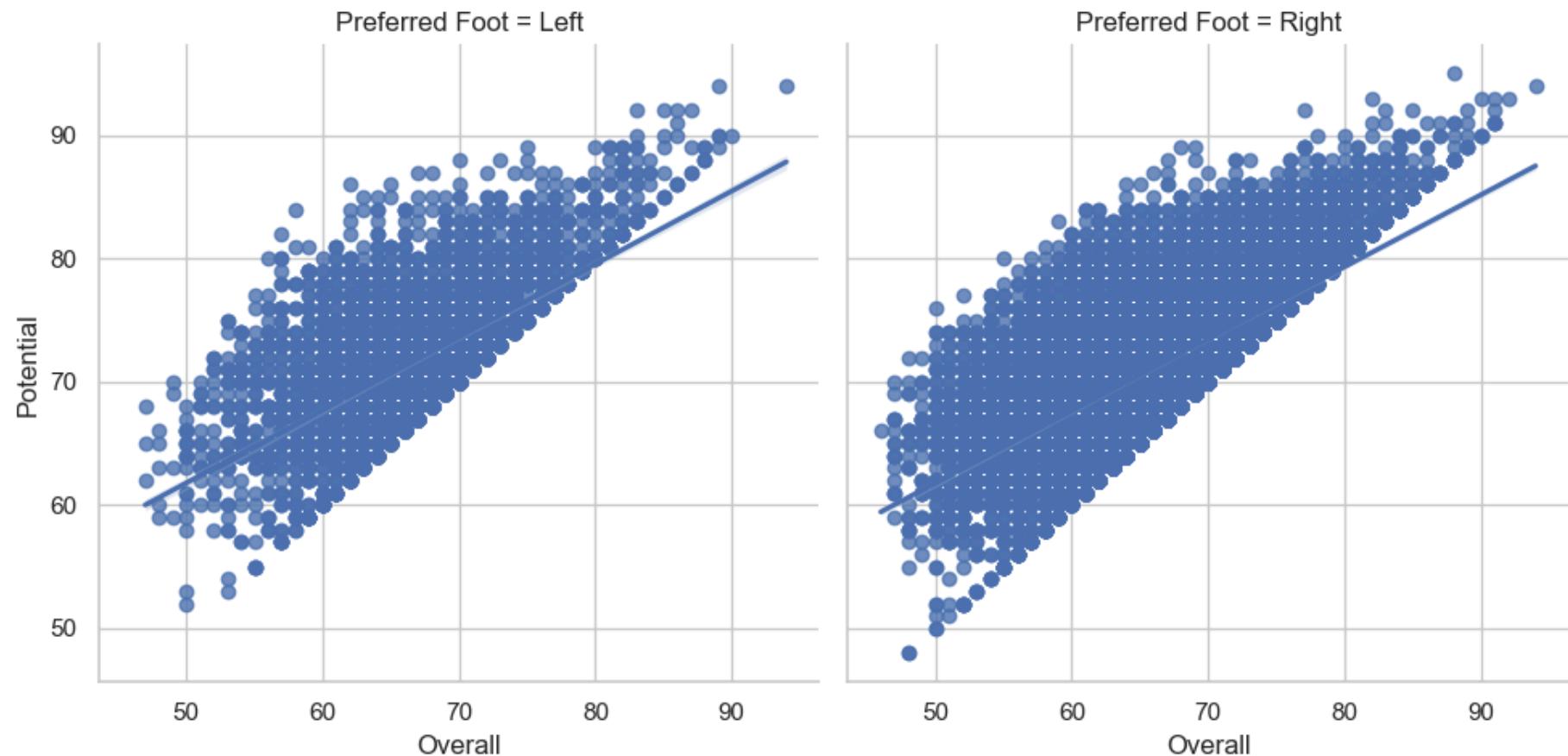
We can use a different color palette as follows-

```
In [79]: g = sns.lmplot(x="Overall",y="Potential",hue="Preferred Foot",data=fifa, palette="Set1")
```



We can plot the levels of the third variable across different columns as follows-

```
In [80]: g = sns.lmplot(x="Overall", y="Potential", col="Preferred Foot", data=fifa)
```



Multi-plot grids

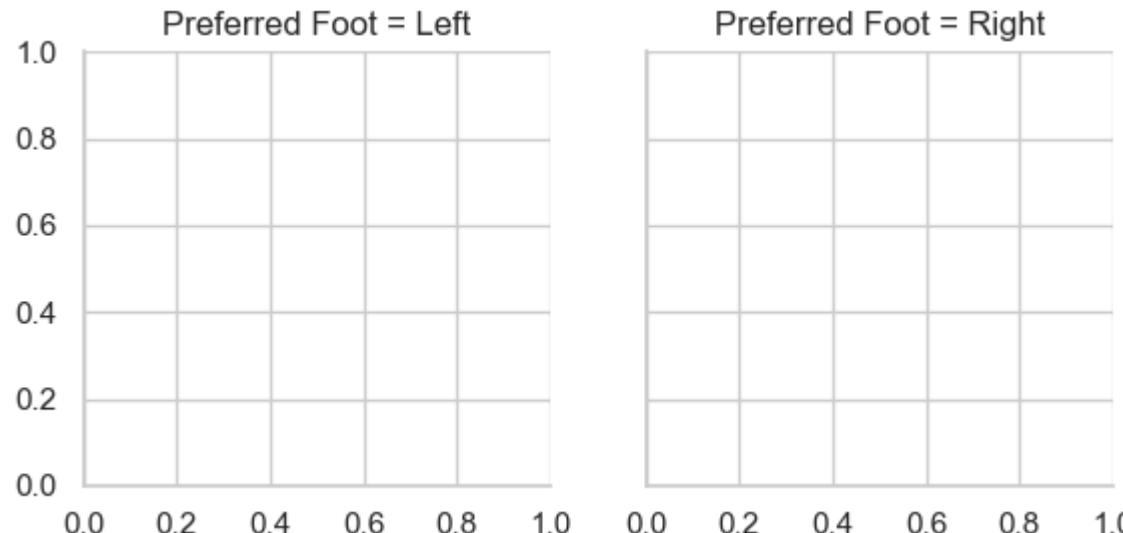
Seaborn `FacetGrid()` function

- The `FacetGrid` class is useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset.
- A `FacetGrid` can be drawn with up to three dimensions - `row`, `col` and `hue`. The first two have obvious correspondence with the resulting array of axes - the `hue` variable is a third dimension along a depth axis, where different levels are plotted with different colors.

- The class is used by initializing a FacetGrid object with a dataframe and the names of the variables that will form the `row`, `column` or `hue` dimensions of the grid.
- These variables should be categorical or discrete, and then the data at each level of the variable will be used for a facet along that axis.

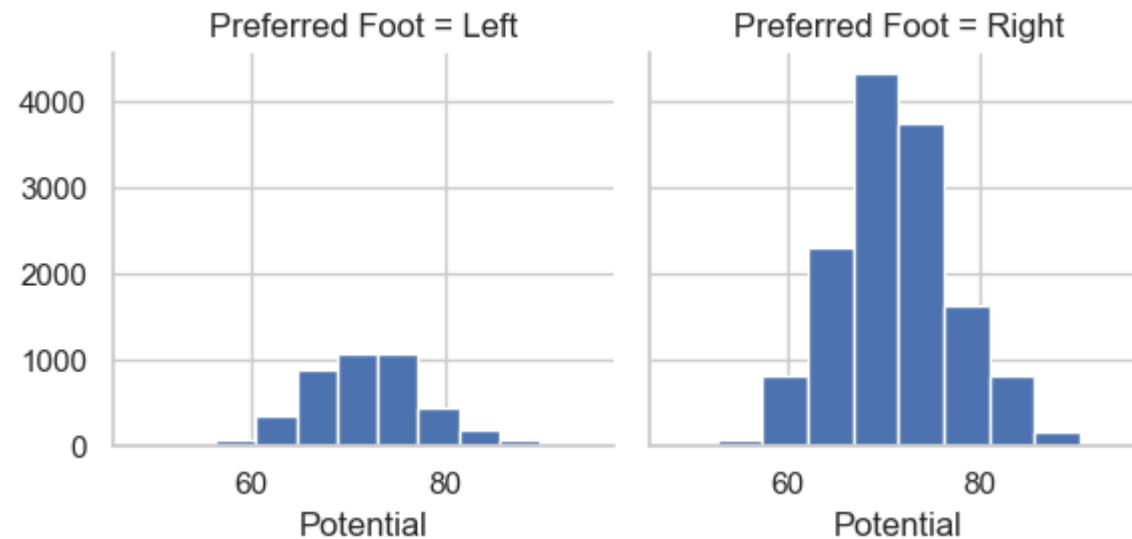
We can initialize a 1x2 grid of facets using the fifa19 dataset.

```
In [83]: g = sns.FacetGrid(fifa, col="Preferred Foot")
```

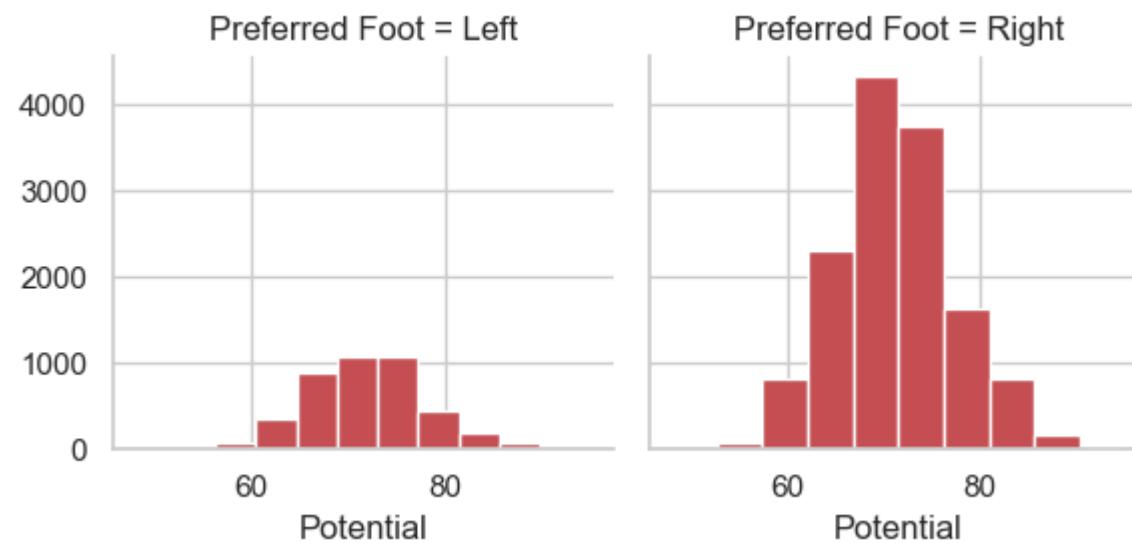


We can draw a univariate plot of Potential variable on each facet as follows-

```
In [85]: g = sns.FacetGrid(fifa, col="Preferred Foot")
g=g.map(plt.hist,"Potential")
```

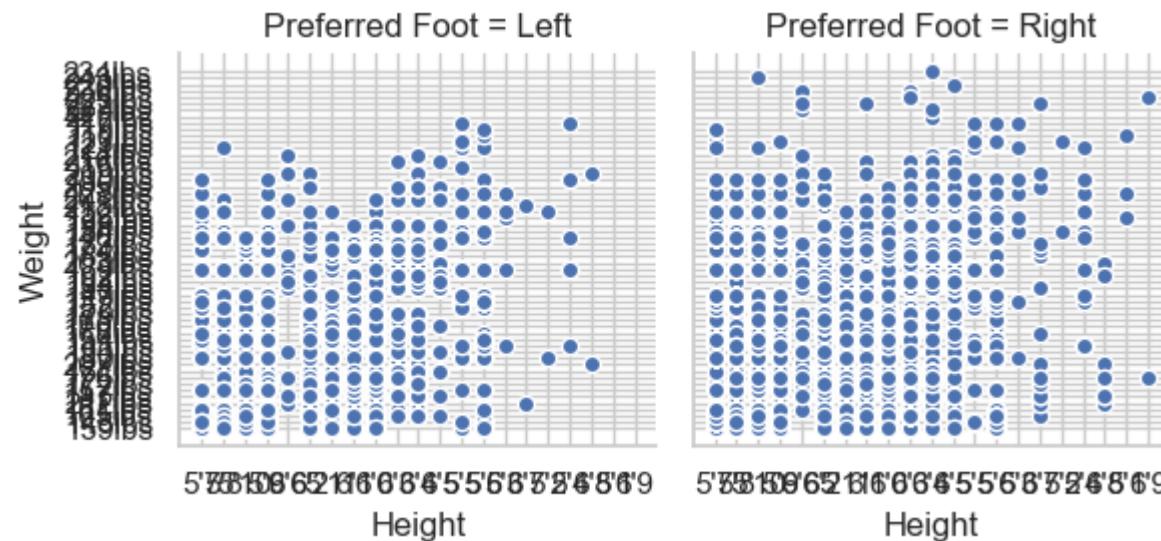


```
In [86]: g = sns.FacetGrid(fifa, col="Preferred Foot")
g=g.map(plt.hist,"Potential",bins=10,color='r')
```



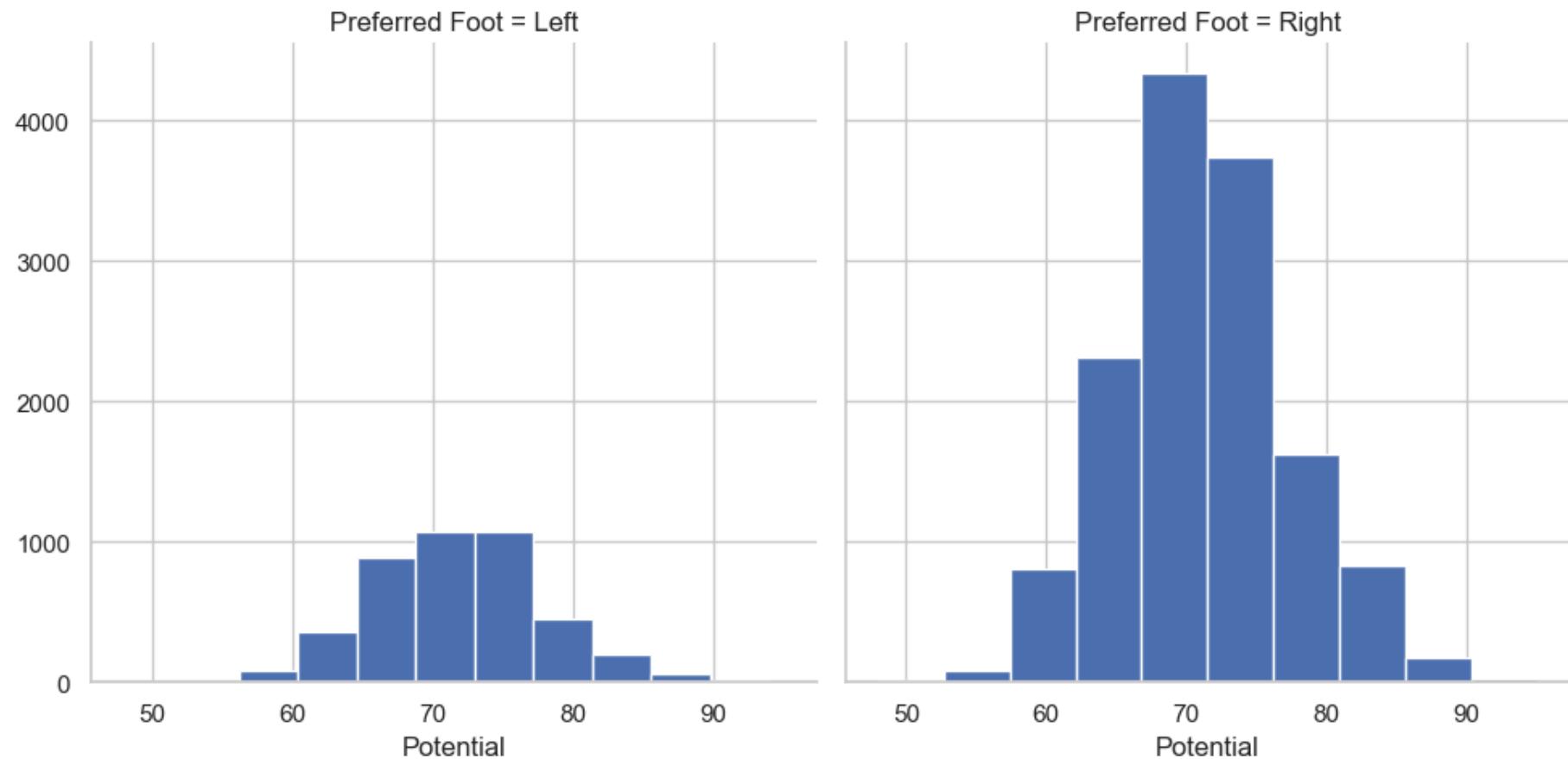
We can plot a bivariate function on each facet as follows-

```
In [88]: g = sns.FacetGrid(fifa, col="Preferred Foot")
g=g.map(plt.scatter,"Height","Weight",edgecolor="w").add_legend()
```



The size of the figure is set by providing the height of each facet, along with the aspect ratio:

```
In [90]: g = sns.FacetGrid(fifa, col="Preferred Foot", height=5, aspect=1)
g = g.map(plt.hist, "Potential")
```



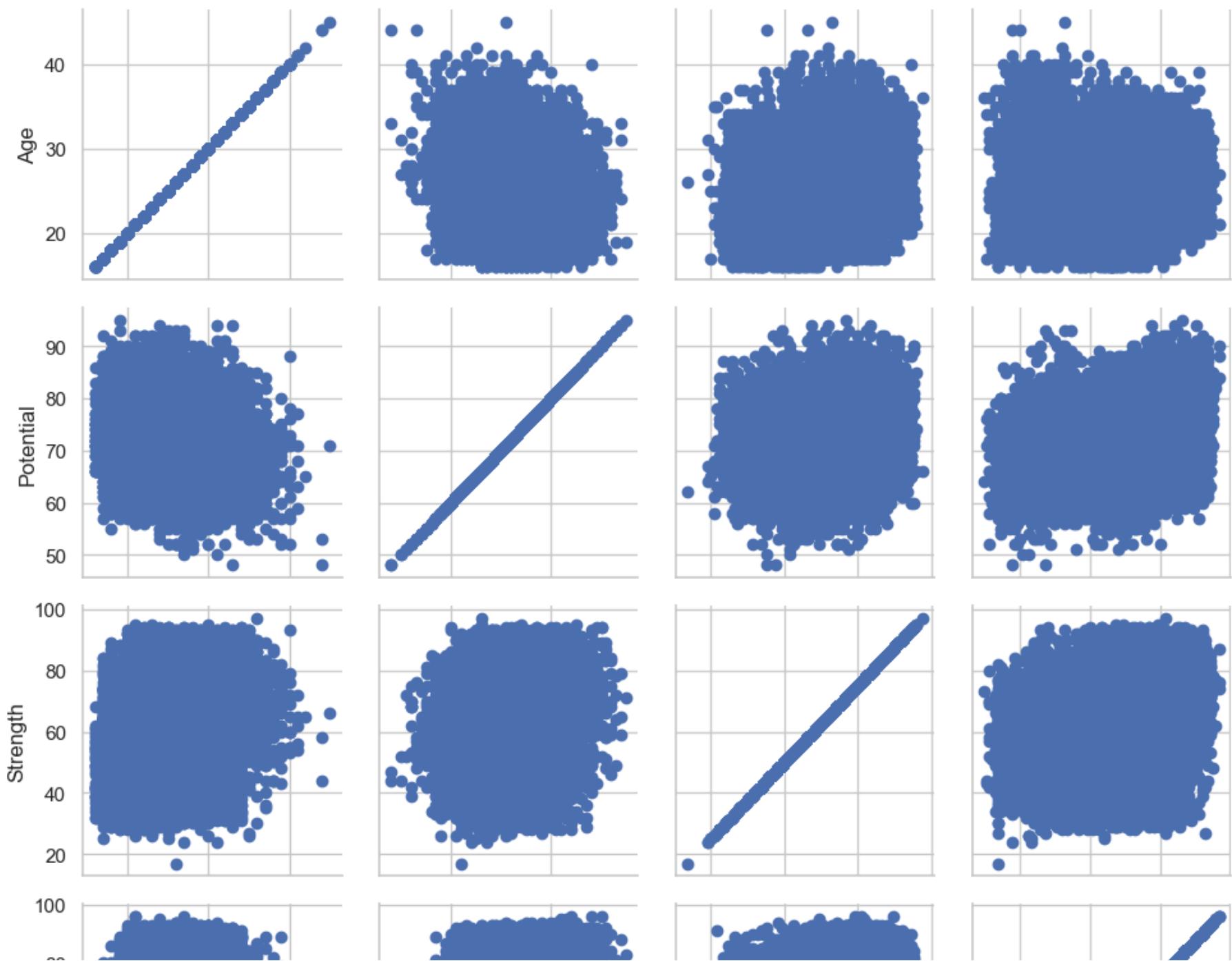
Seaborn `Pairgrid()` function

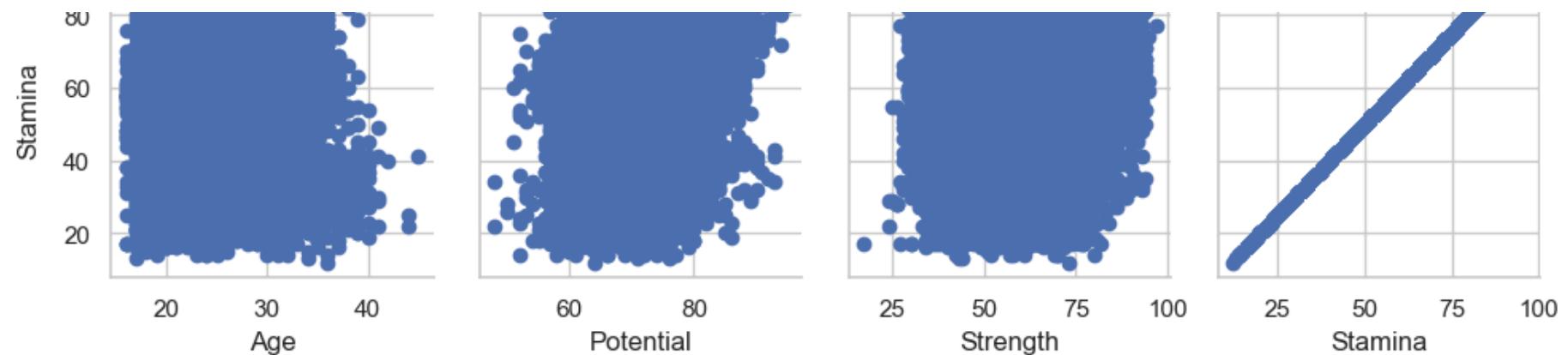
- This function plots subplot grid for plotting pairwise relationships in a dataset.
- This class maps each variable in a dataset onto a column and row in a grid of multiple axes.
- Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the marginal distribution of each variable can be shown on the diagonal.
- It can also represent an additional level of conditionalization with the hue parameter, which plots different subsets of data in different colors.

- This uses color to resolve elements on a third dimension, but only draws subsets on top of each other and will not tailor the hue parameter for the specific visualization the way that axes-level functions that accept hue will.

```
In [91]: fifa_new = fifa[['Age', "Potential", "Strength", "Stamina", "Preferred Foot"]]
```

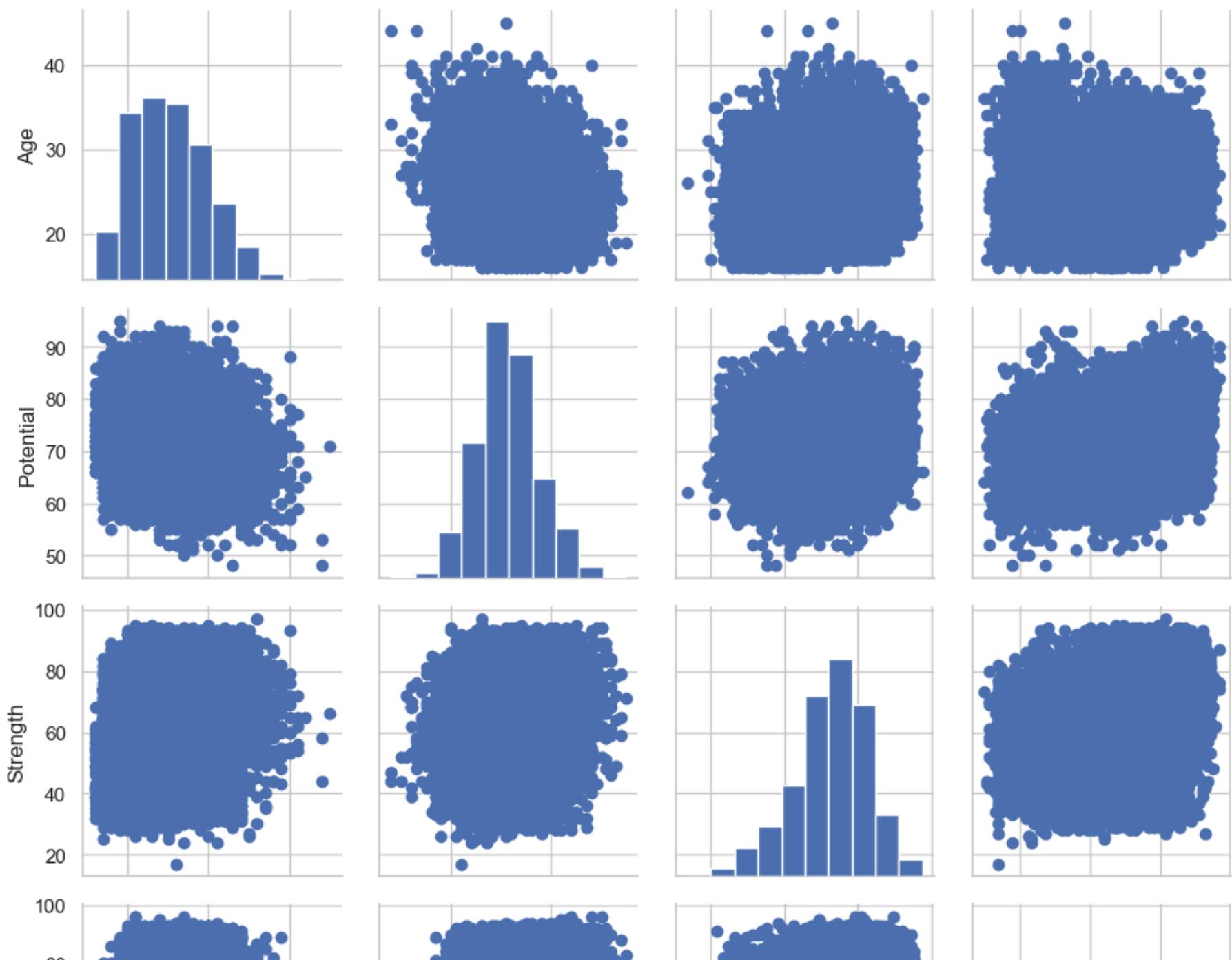
```
In [94]: g = sns.PairGrid(fifa_new)
g = g.map(plt.scatter)
```

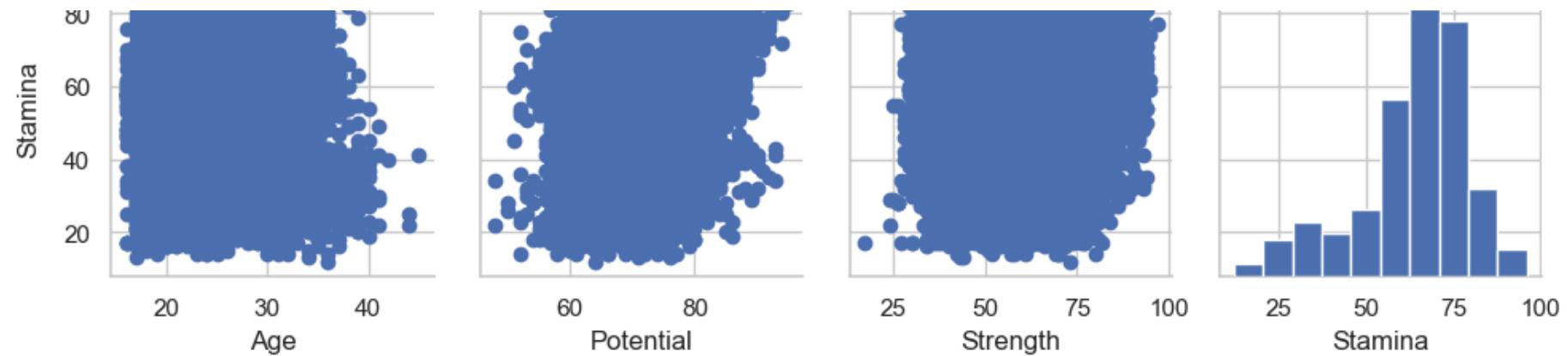




We can show a univariate distribution on the diagonal as follows-

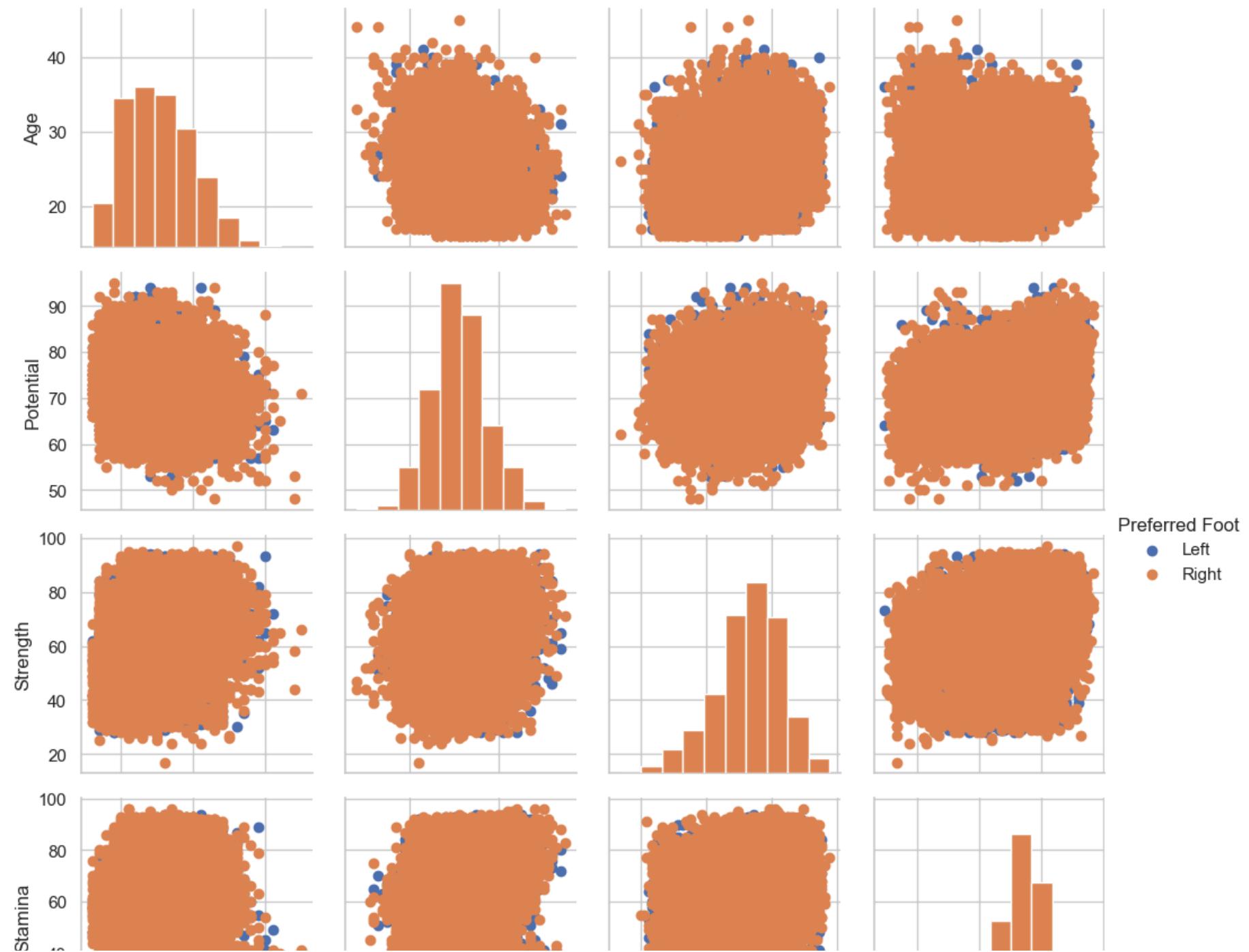
```
In [96]: g = sns.PairGrid(fifa_new)
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
```

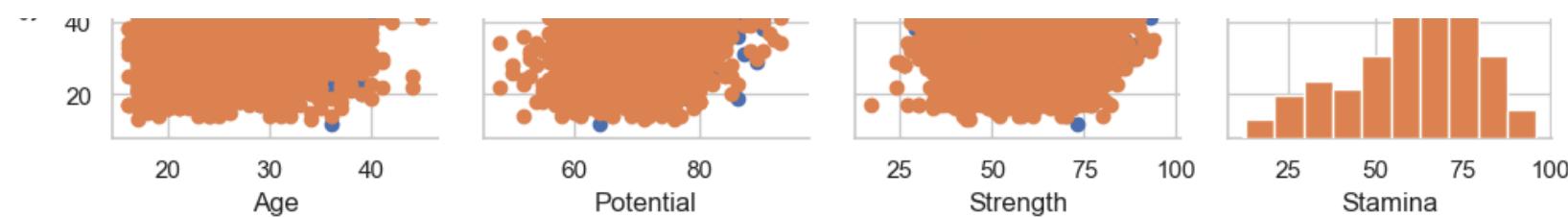




We can color the points using the categorical variable Preferred Foot as follows -

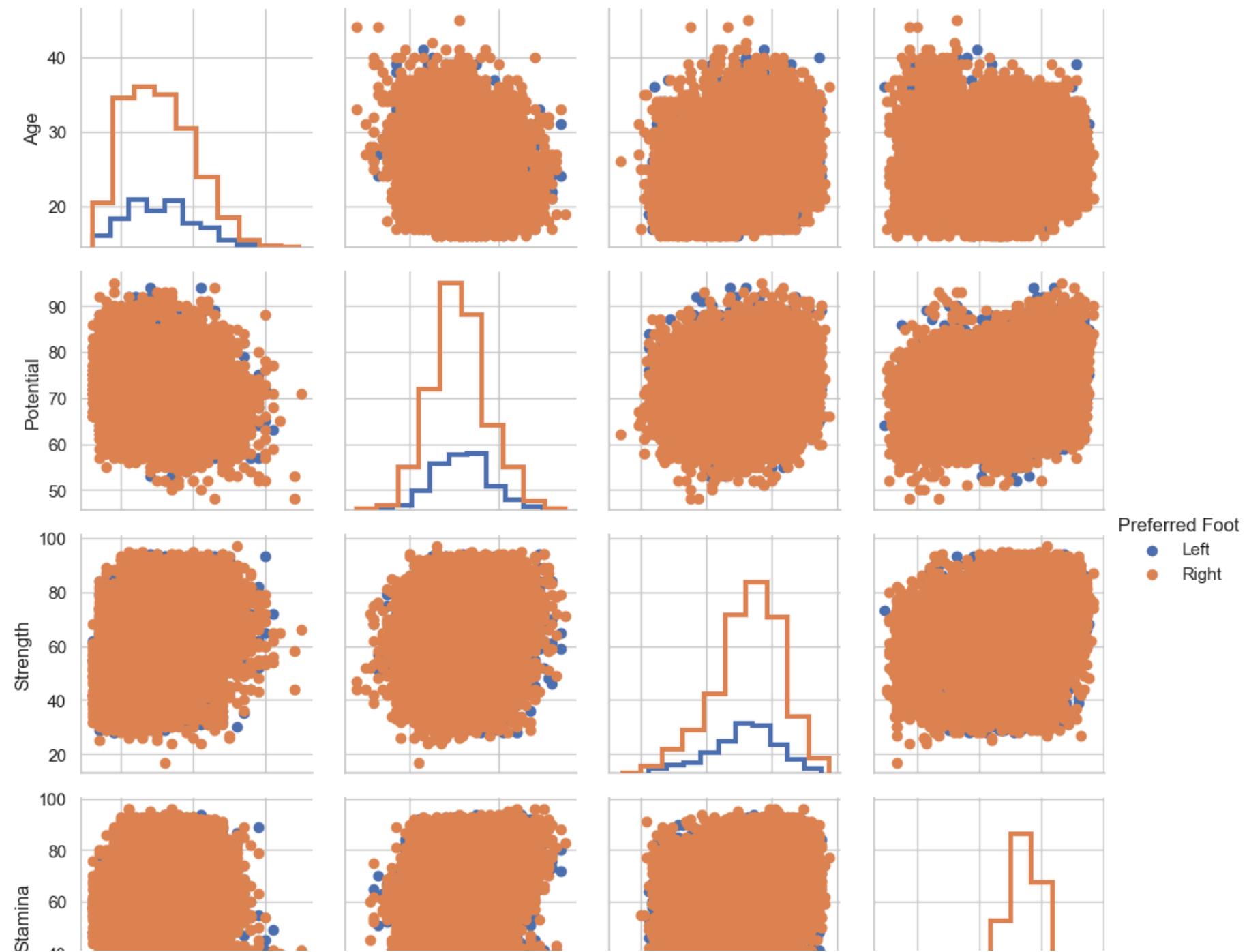
```
In [98]: g = sns.PairGrid(fifa_new,hue="Preferred Foot")
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
g=g.add_legend()
```

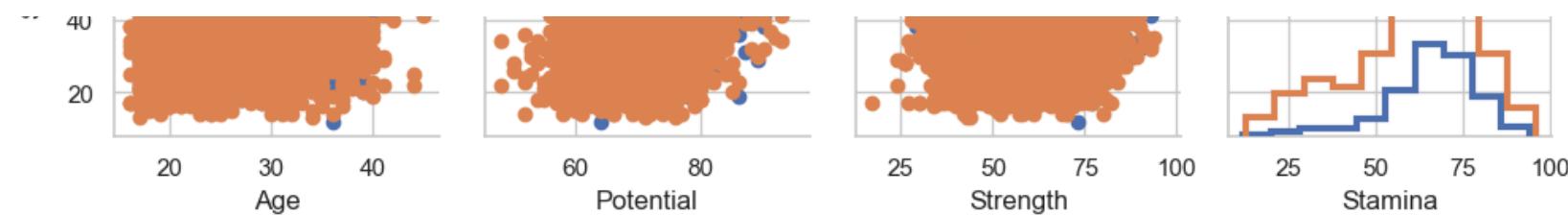




We can use a different style to show multiple histograms as follows-

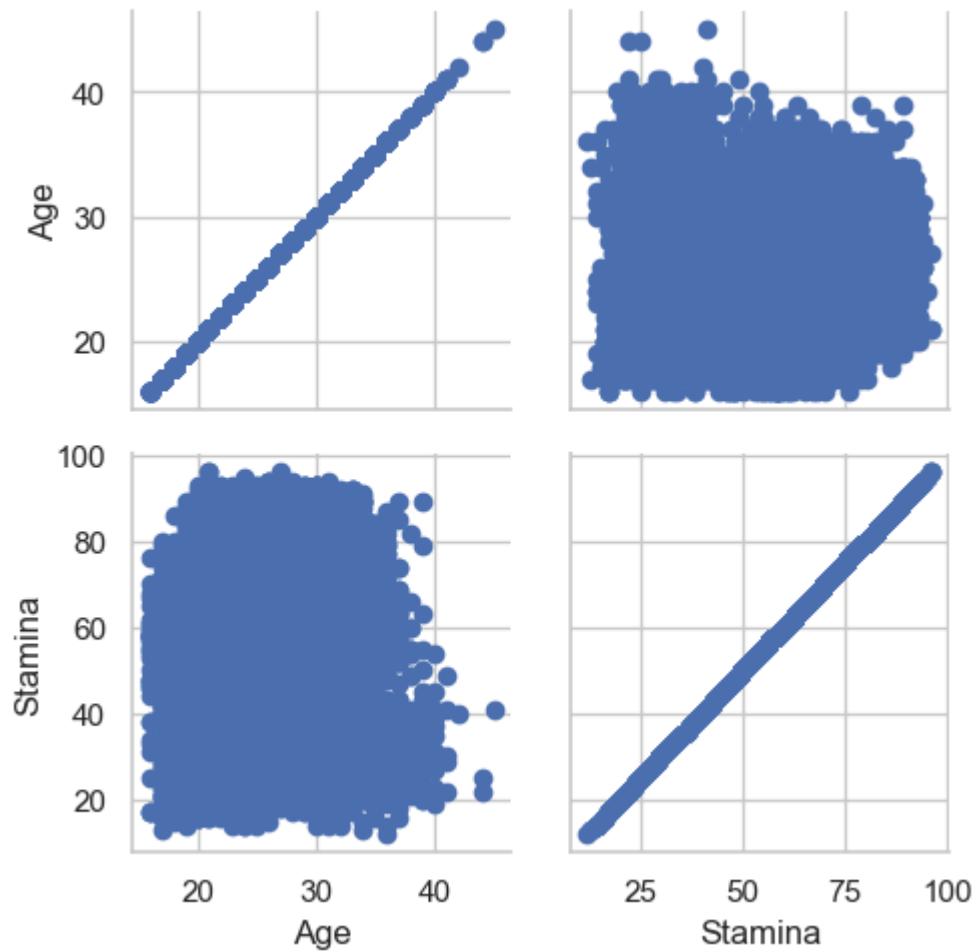
```
In [99]:  
g = sns.PairGrid(fifa_new,hue="Preferred Foot")  
g = g.map_diag(plt.hist,histtype="step",linewidth=3)  
g = g.map_offdiag(plt.scatter)  
g=g.add_legend()
```





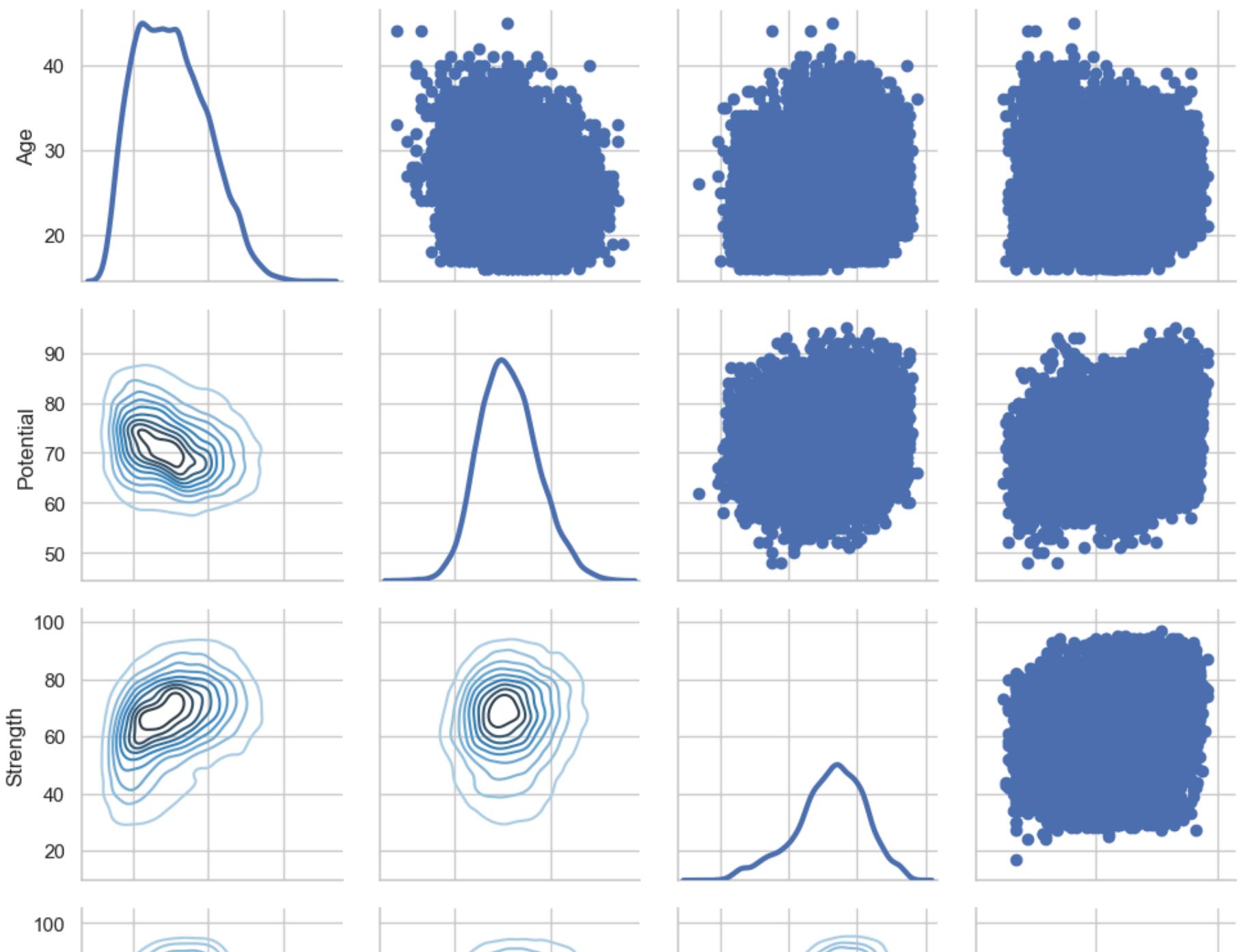
We can plot a subset of variables as follows-

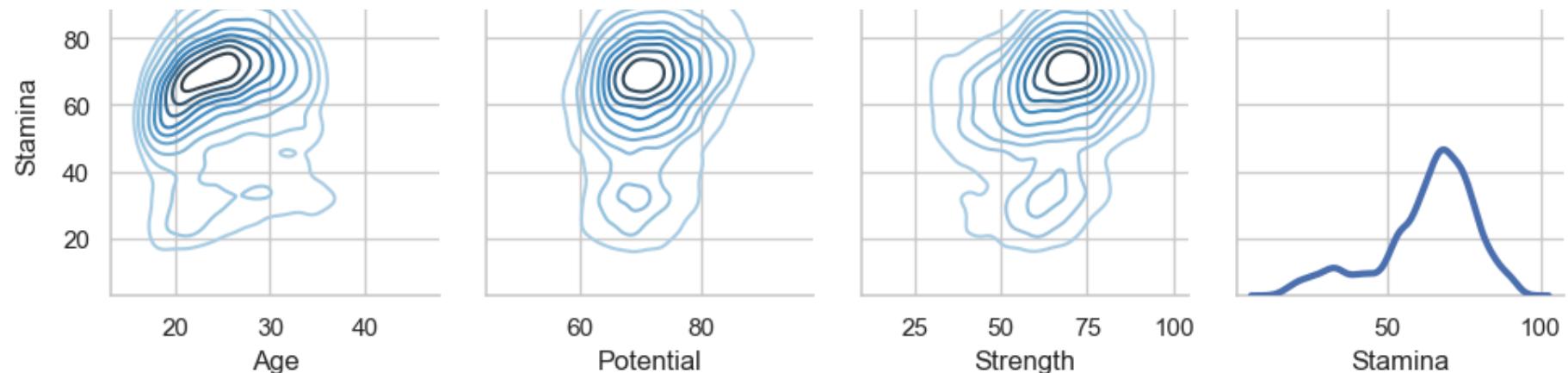
```
In [101...]:  
g = sns.PairGrid(fifa_new, vars=['Age', 'Stamina'])  
g.map(plt.scatter)
```



We can use different functions on the upper and lower triangles as follows-

```
In [103...]:  
g = sns.PairGrid(fifa_new)  
g = g.map_upper(plt.scatter)  
g = g.map_lower(sns.kdeplot, cmap="Blues_d")  
g = g.map_diag(sns.kdeplot, lw=3, legend=False)
```



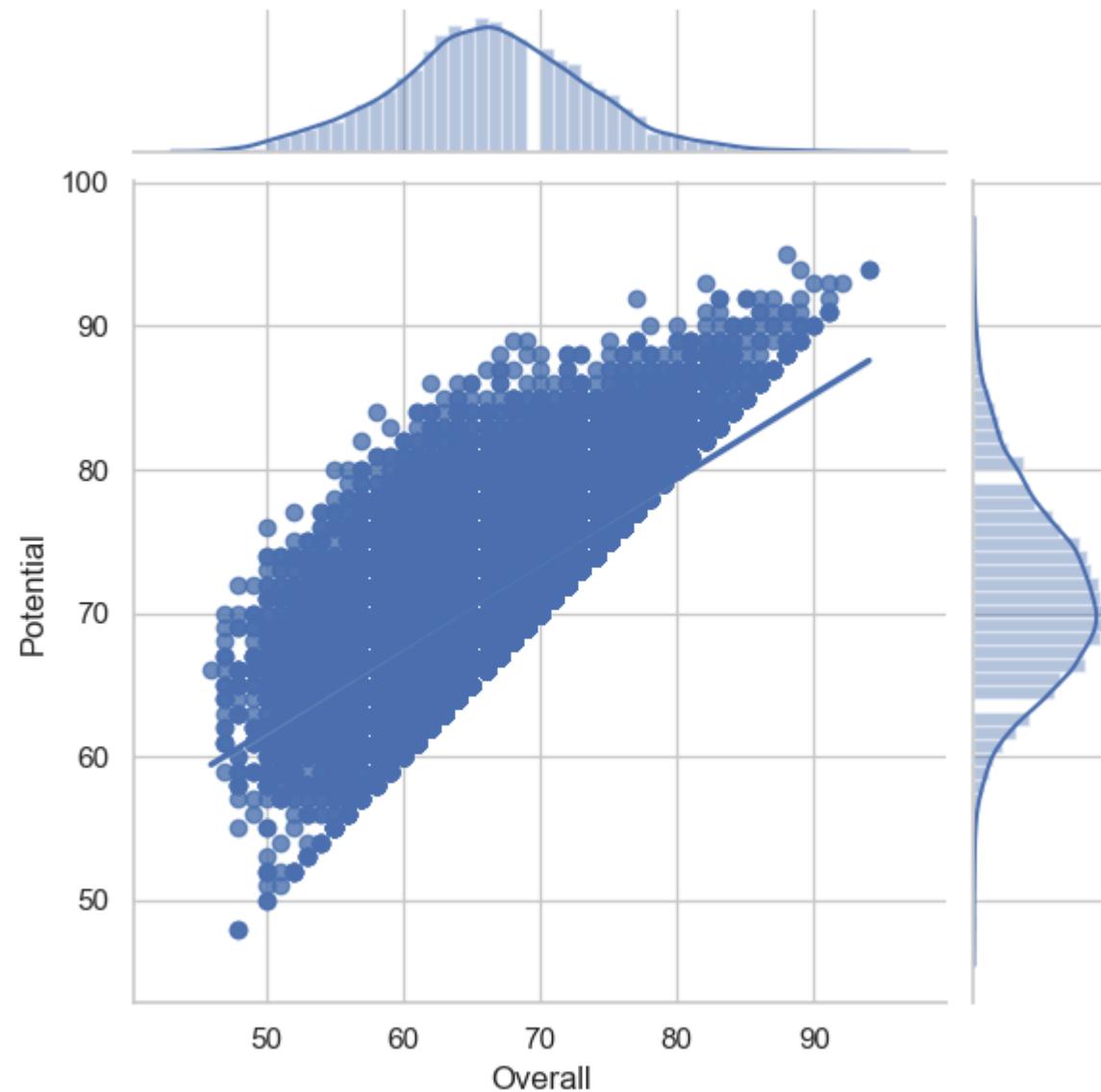


Seaborn `Jointgrid()` function

- This function provides a grid for drawing a bivariate plot with marginal univariate plots.
- It set up the grid of subplots.

We can initialize the figure and add plots using default parameters as follows-

```
In [105...]: g = sns.JointGrid(x="Overall", y="Potential", data=fifa)
g = g.plot(sns.regplot, sns.distplot)
```

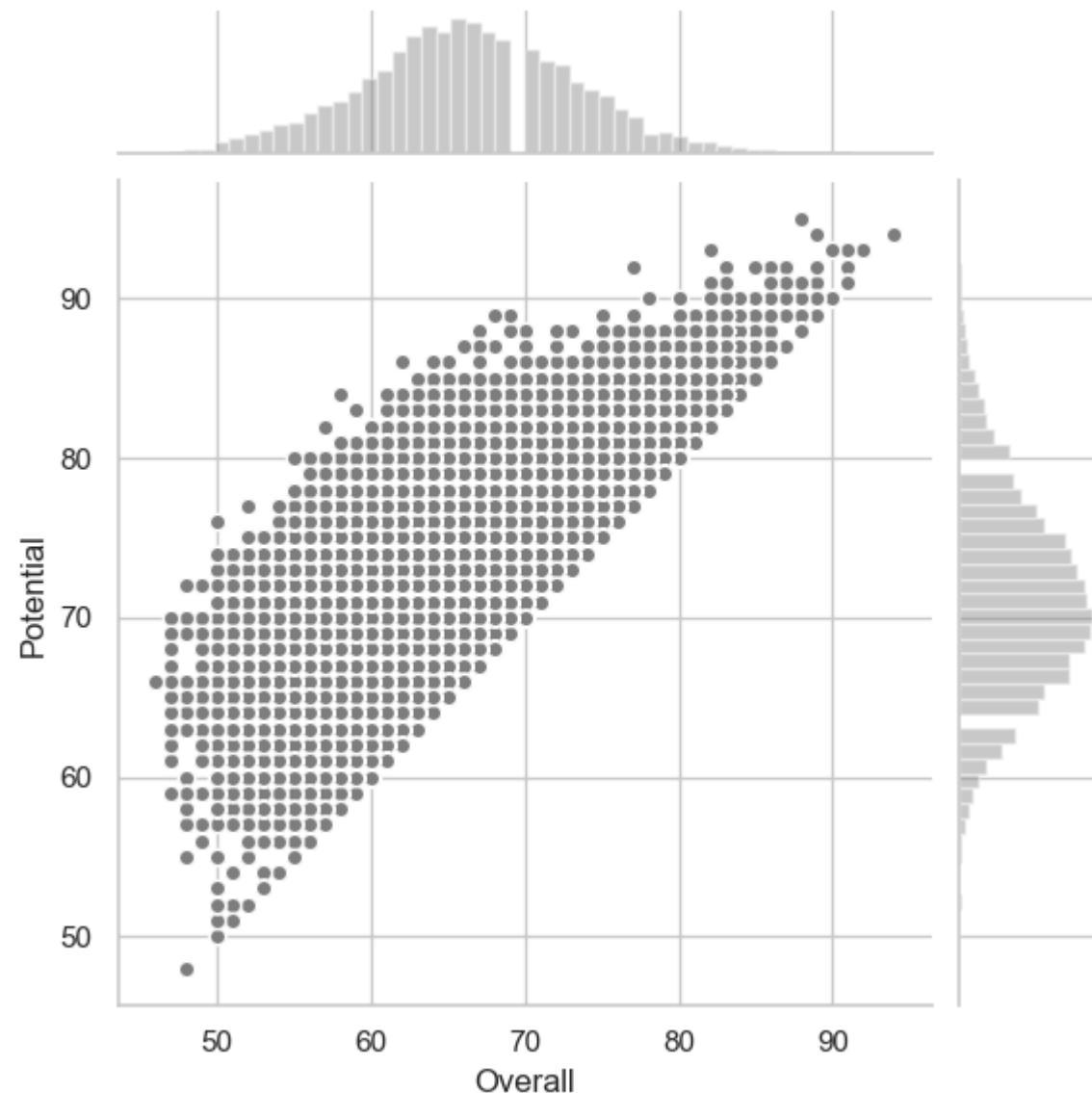


We can draw the joint and marginal plots separately, which allows finer-level control other parameters as follows -

In [108...]

```
g = sns.JointGrid(x="Overall", y="Potential", data=fifa)
g = g.plot_joint(plt.scatter, color=".5", edgecolor="white")
```

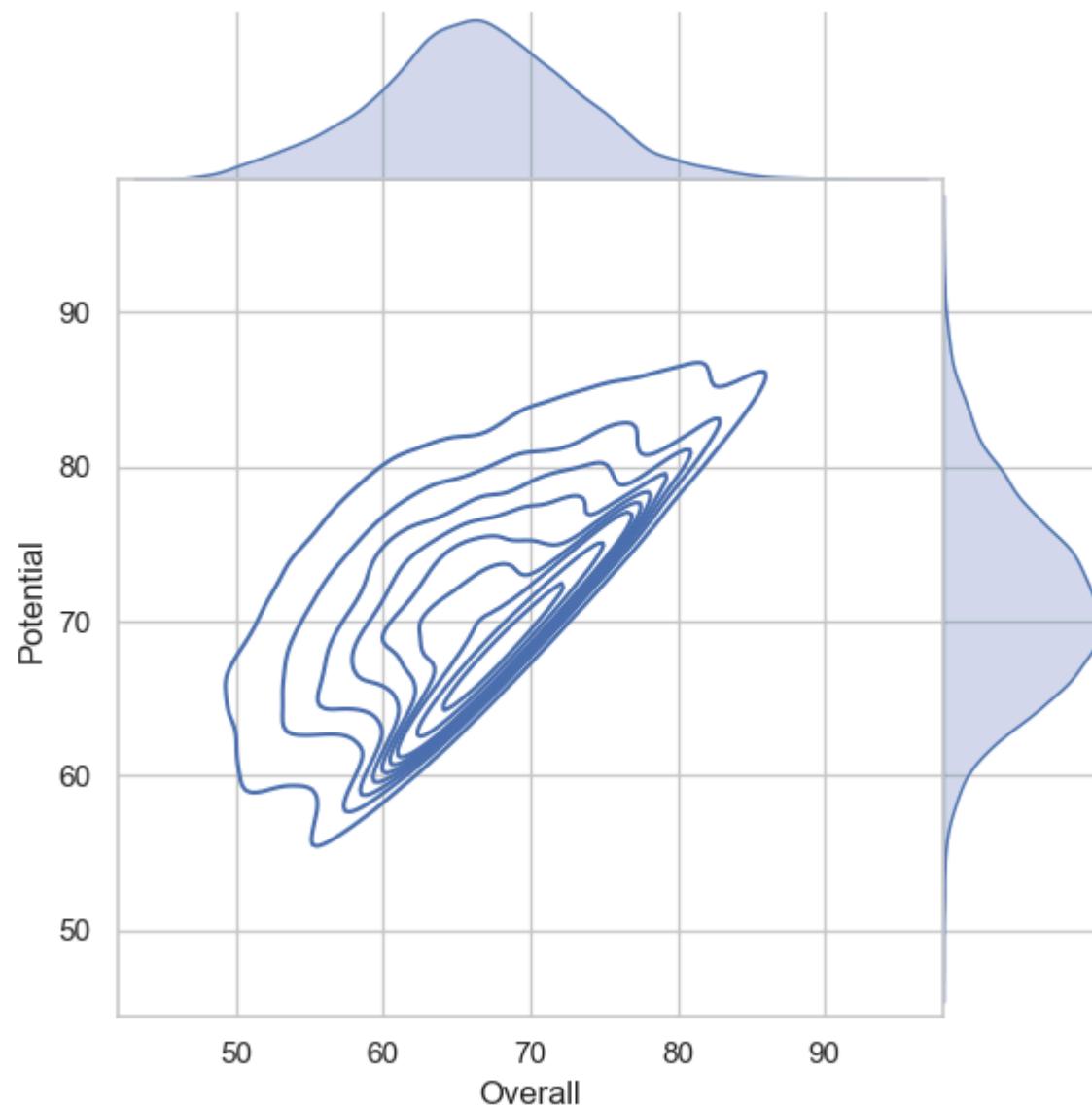
```
g = g.plot_marginals(sns.distplot, kde=False, color=".5")
```



We can remove the space between the joint and marginal axes as follows -

In [112...]

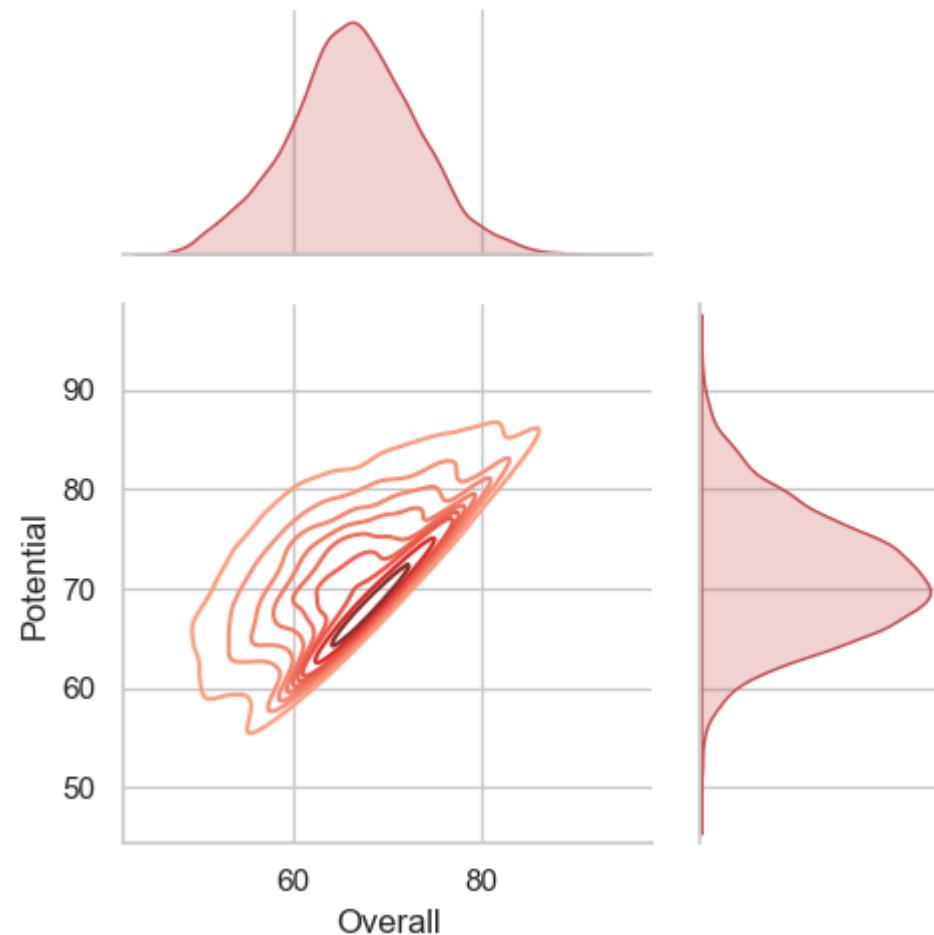
```
g = sns.JointGrid(x="Overall", y="Potential", data=fifa, space=0)
g = g.plot_joint(sns.kdeplot, cmap='Blues_d')
g = g.plot_marginals(sns.kdeplot, shade=True)
```



We can draw a smaller plot with relatively larger marginal axes as follows -

In [114...]

```
g = sns.JointGrid(x="Overall", y="Potential", data=fifa, height=5, ratio=2)
g = g.plot_joint(sns.kdeplot, cmap="Reds_d")
g = g.plot_marginals(sns.kdeplot, color="r", shade=True)
```



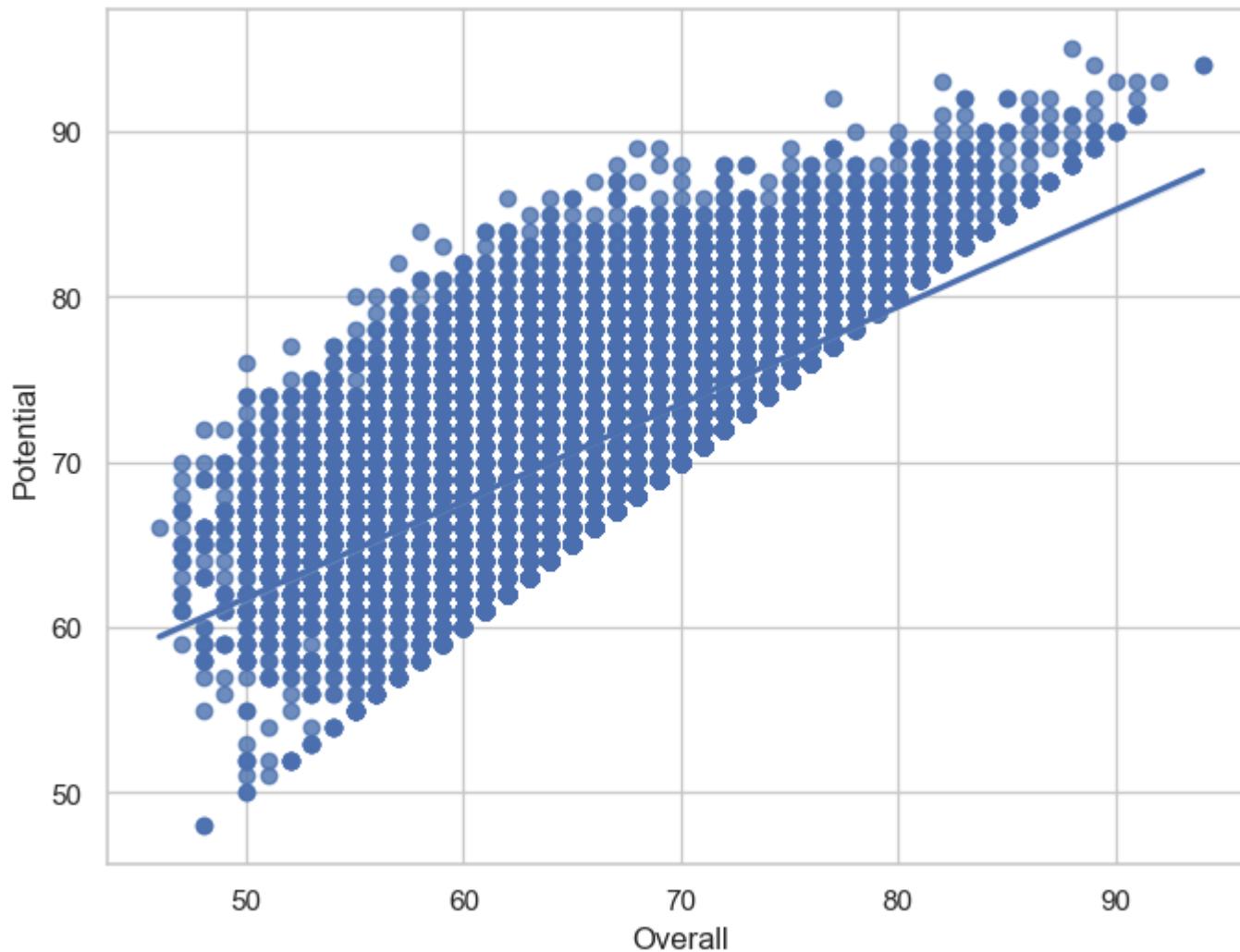
Controlling the size and shape of the plot

- The default plots made by `regplot()` and `lmplot()` look the same but on axes that have a different size and shape.
- This is because `regplot()` is an “axes-level” function draws onto a specific axes.

- This means that you can make multi-panel figures yourself and control exactly where the regression plot goes.
- If no axes object is explicitly provided, it simply uses the “currently active” axes, which is why the default plot has the same size and shape as most other matplotlib functions.
- To control the size, we need to create a figure object ourself as follows-

In [115...]

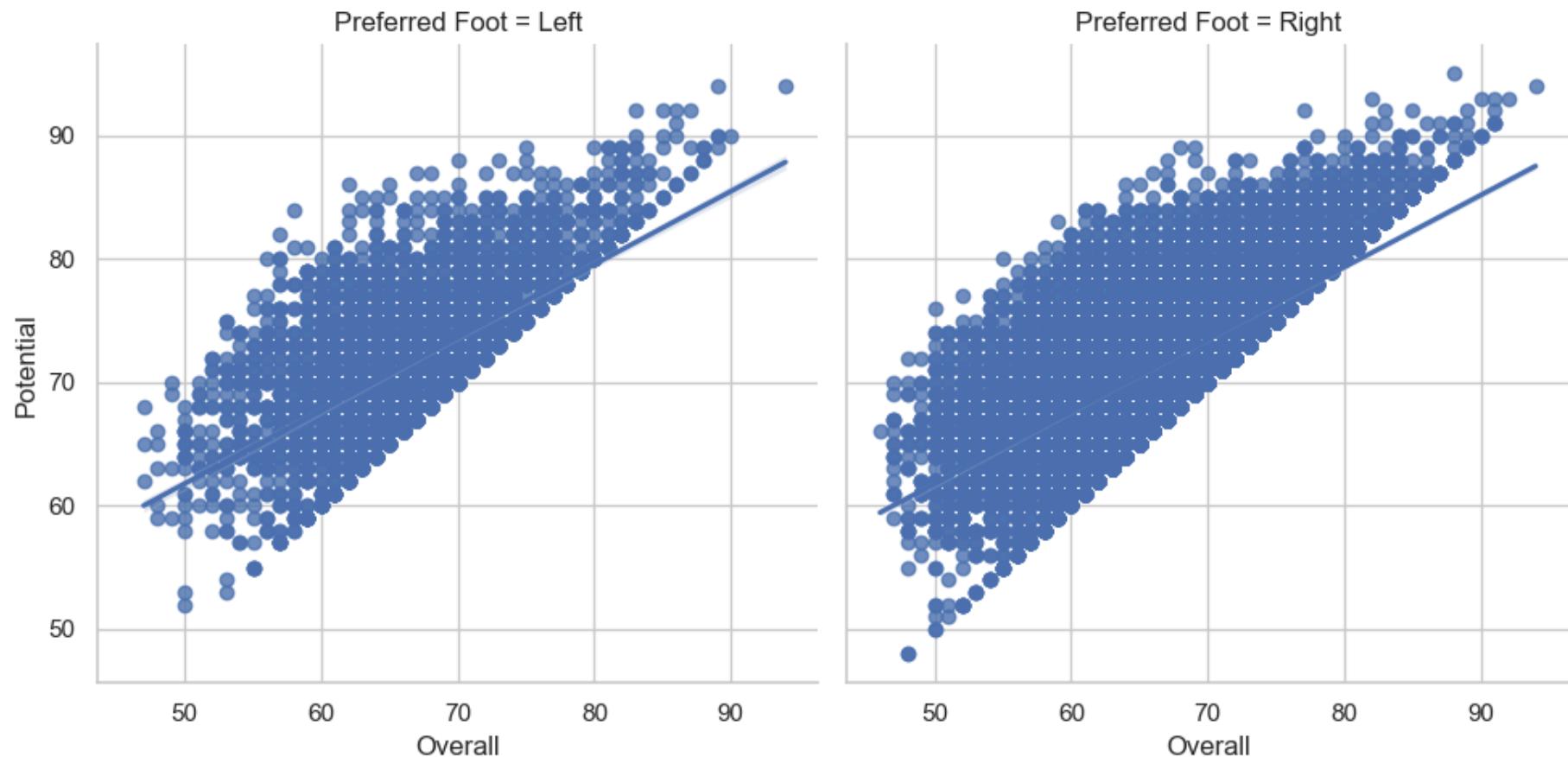
```
f, ax = plt.subplots(figsize=(8,6))
ax= sns.regplot(x="Overall",y="Potential",data=fifa);
```



In contrast, the size and shape of the lmplot() figure is controlled through the FacetGrid interface using the size and aspect parameters, which apply to each facet in the plot, not to the overall figure itself.

```
In [118]: sns.lmplot(x="Overall", y="Potential", col="Preferred Foot", data=fifa, col_wrap=2, height=5, aspect=1)
```

```
Out[118]: <seaborn.axisgrid.FacetGrid at 0x1b596157490>
```



Seaborn figure styles

- There are five preset seaborn themes: `darkgrid`, `whitegrid`, `dark`, `white` and `ticks`.
- They are each suited to different applications and personal preferences.
- The default theme is `darkgrid`.
- The grid helps the plot serve as a lookup table for quantitative information, and the white-on grey helps to keep the grid from competing with lines that represent data.

- The whitegrid theme is similar, but it is better suited to plots with heavy data elements:

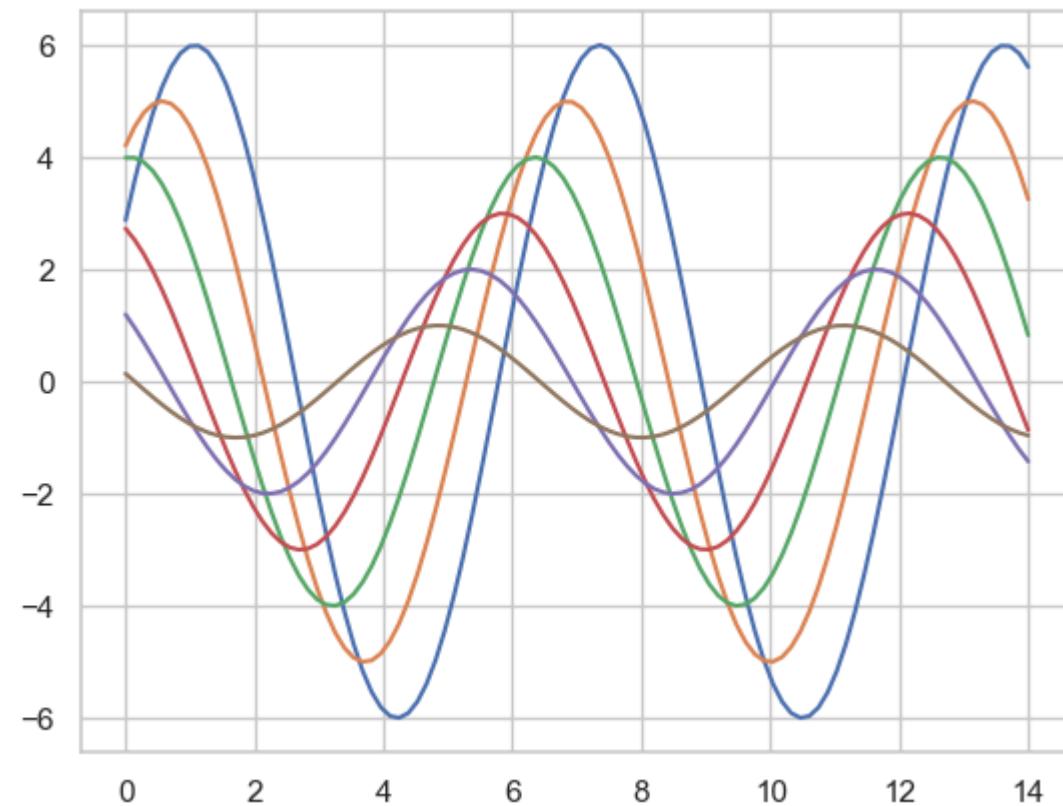
I will define a simple function to plot some offset sine waves, which will help us see the different stylistic parameters as follows -

In [123...]

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
```

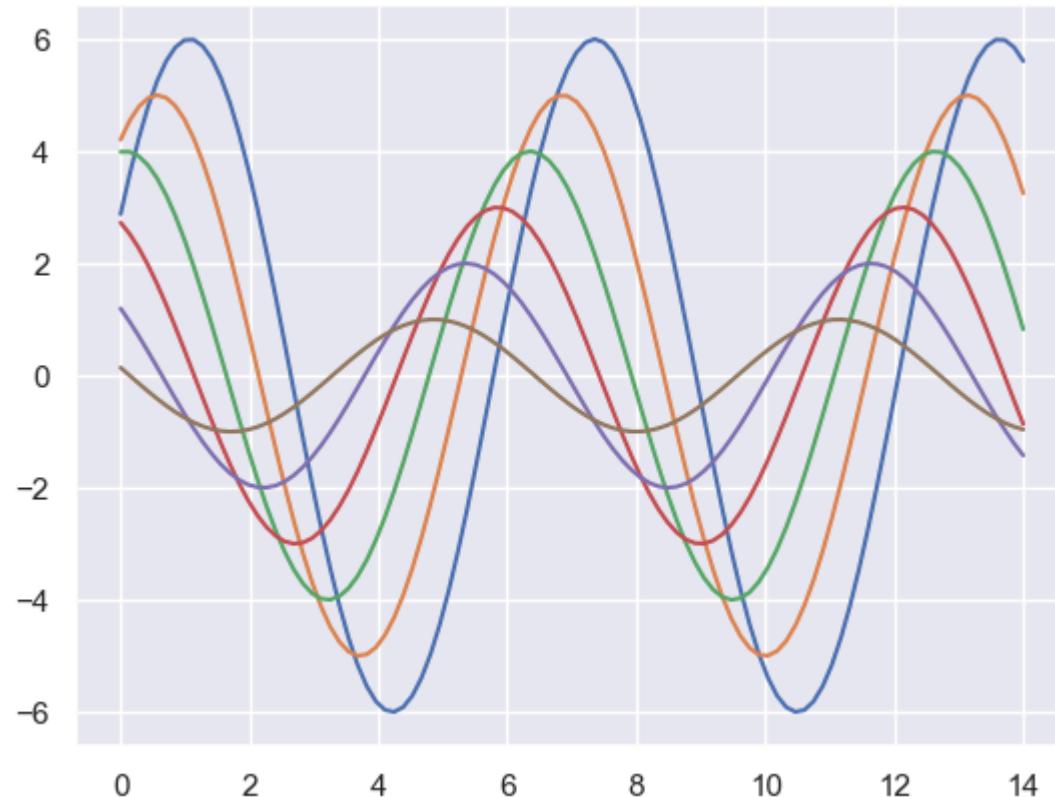
In [124...]

```
sinplot()
```



In [125...]

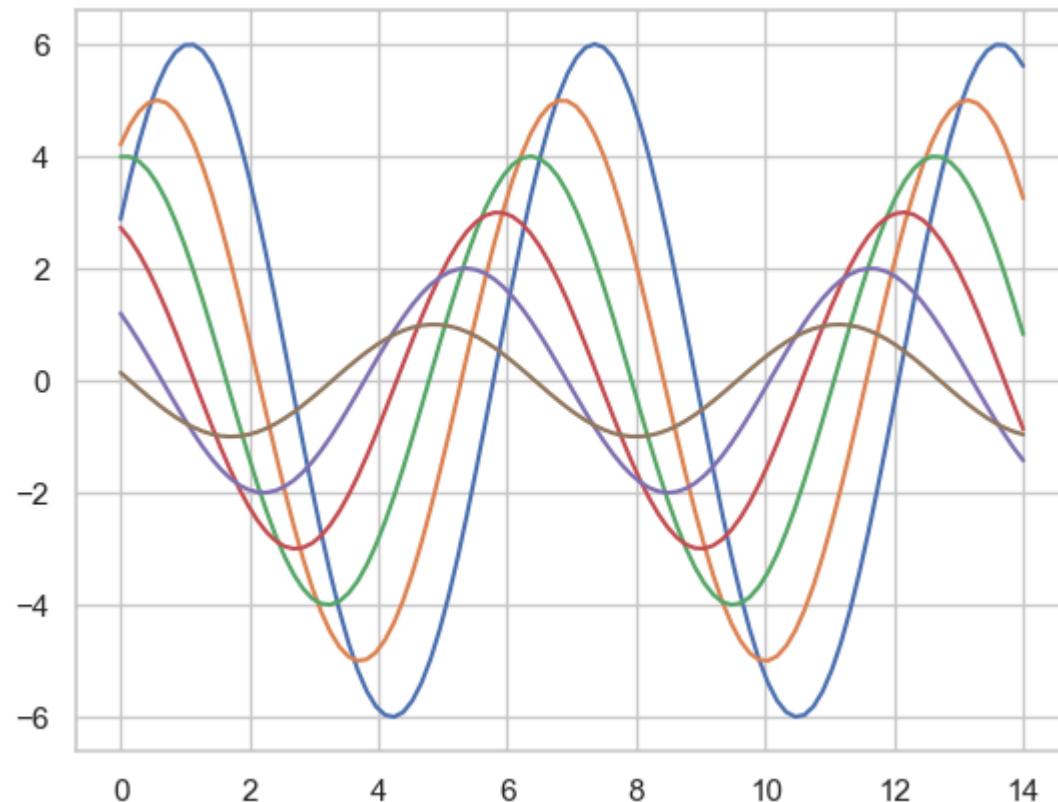
```
sns.set()
sinplot()
```



- We can set different styles as follows -

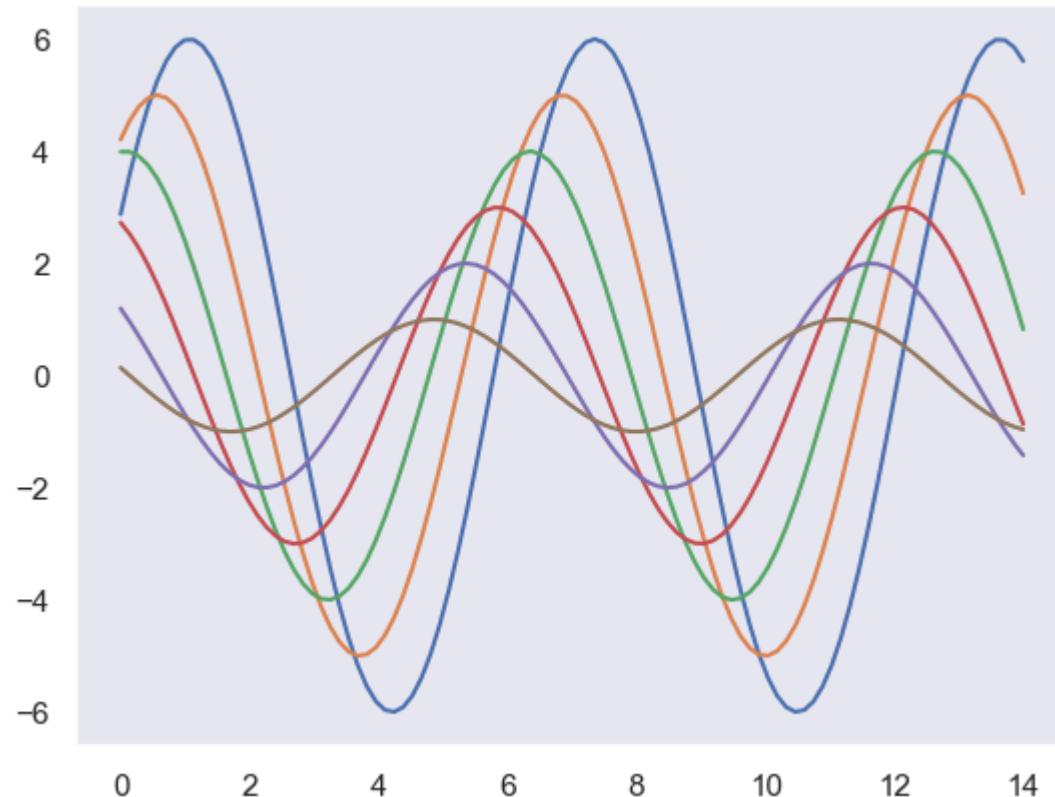
In [127...]

```
sns.set_style("whitegrid")
sinplot()
```



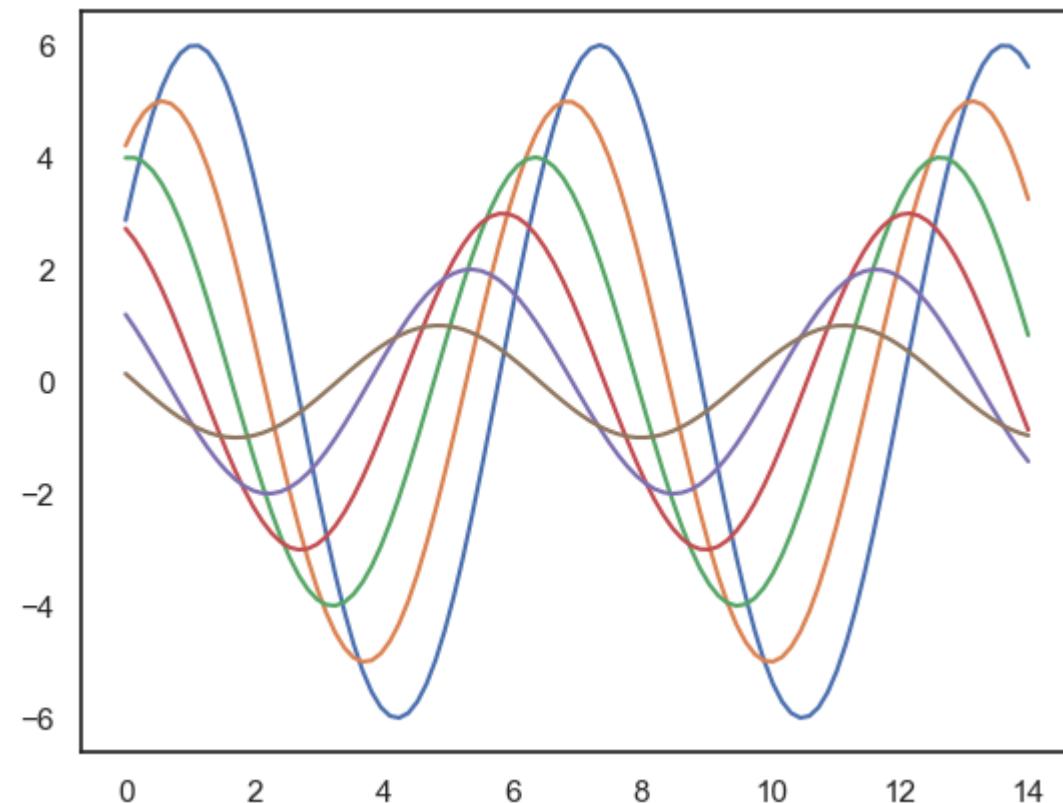
In [128]:

```
sns.set_style('dark')
sinplot()
```



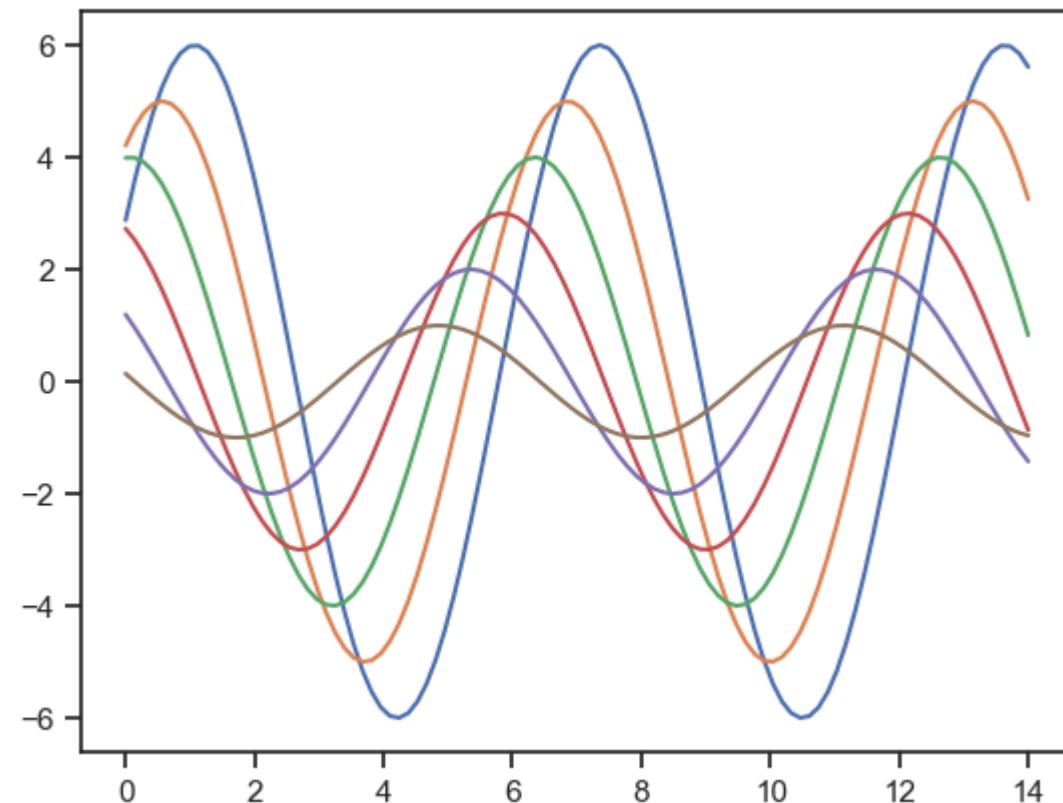
In [129]:

```
sns.set_style("white")
sinplot()
```



In [130]:

```
sns.set_style("ticks")
sinplot()
```



In []: