

## Table of Contents

The table of contents for this tutorial is as follows -

- Import libraries
- Read dataset
- Exploratory data analysis
- Visualize distribution of Age variable with Seaborn distplot() function
- Seaborn Kernel Density Estimation (KDE) plot
- Histograms
- Visualize distribution of values in Preferred Foot variable with Seaborn countplot() function
- Seaborn catplot() function
- Seaborn stripplot() function
- Seaborn boxplot() function
- Seaborn violinplot() function
- Seaborn pointplot() function
- Seaborn barplot() function
- Visualizing statistical relationship with Seaborn relplot() function
- Seaborn scatterplot() function
- Seaborn lineplot() function
- Seaborn regplot() function
- Seaborn lmplot() function
- Multi-plot grids
- Seaborn Facetgrid() function
- Seaborn Pairgrid() function
- Seaborn Jointgrid() function
- Controlling the size and shape of the plot
- Seaborn figure styles

## Import libraries

```
In [1]: import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
sns.set(style="whitegrid")
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline
```

## import ignore warnings

```
In [2]: # ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

## read dataset

```
In [3]: df = pd.read_csv(r'C:\Users\ADMIN\Desktop\DATA SCIENCE NOTES\NOVEMBER MONTH\27TH NOV\25th, 26th- Advanced EDA project
```

```
In [4]: df
```

Out[4]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	
0	0	158023	L. Messi	31	<a href="https://cdn.sofifa.org/players/4/19/158023.png">https://cdn.sofifa.org/players/4/19/158023.png</a>	Argentina	<a href="https://cdn.sofifa.org/flags/ARGN.png">https://cdn.sofifa.org/flags/ARGN.png</a>
1	1	20801	Cristiano Ronaldo	33	<a href="https://cdn.sofifa.org/players/4/19/20801.png">https://cdn.sofifa.org/players/4/19/20801.png</a>	Portugal	<a href="https://cdn.sofifa.org/flags/PORL.png">https://cdn.sofifa.org/flags/PORL.png</a>
2	2	190871	Neymar Jr	26	<a href="https://cdn.sofifa.org/players/4/19/190871.png">https://cdn.sofifa.org/players/4/19/190871.png</a>	Brazil	<a href="https://cdn.sofifa.org/flags/BRAZ.png">https://cdn.sofifa.org/flags/BRAZ.png</a>
3	3	193080	De Gea	27	<a href="https://cdn.sofifa.org/players/4/19/193080.png">https://cdn.sofifa.org/players/4/19/193080.png</a>	Spain	<a href="https://cdn.sofifa.org/flags/SPAN.png">https://cdn.sofifa.org/flags/SPAN.png</a>
4	4	192985	K. De Bruyne	27	<a href="https://cdn.sofifa.org/players/4/19/192985.png">https://cdn.sofifa.org/players/4/19/192985.png</a>	Belgium	<a href="https://cdn.sofifa.org/flags/BELG.png">https://cdn.sofifa.org/flags/BELG.png</a>
...	...	...	...	...	...	...	...
18202	18202	238813	J. Lundstram	19	<a href="https://cdn.sofifa.org/players/4/19/238813.png">https://cdn.sofifa.org/players/4/19/238813.png</a>	England	<a href="https://cdn.sofifa.org/flags/ENGL.png">https://cdn.sofifa.org/flags/ENGL.png</a>
18203	18203	243165	N. Christoffersson	19	<a href="https://cdn.sofifa.org/players/4/19/243165.png">https://cdn.sofifa.org/players/4/19/243165.png</a>	Sweden	<a href="https://cdn.sofifa.org/flags/SWEA.png">https://cdn.sofifa.org/flags/SWEA.png</a>
18204	18204	241638	B. Worman	16	<a href="https://cdn.sofifa.org/players/4/19/241638.png">https://cdn.sofifa.org/players/4/19/241638.png</a>	England	<a href="https://cdn.sofifa.org/flags/ENGL.png">https://cdn.sofifa.org/flags/ENGL.png</a>
18205	18205	246268	D. Walker-Rice	17	<a href="https://cdn.sofifa.org/players/4/19/246268.png">https://cdn.sofifa.org/players/4/19/246268.png</a>	England	<a href="https://cdn.sofifa.org/flags/ENGL.png">https://cdn.sofifa.org/flags/ENGL.png</a>
18206	18206	246269	G. Nugent	16	<a href="https://cdn.sofifa.org/players/4/19/246269.png">https://cdn.sofifa.org/players/4/19/246269.png</a>	England	<a href="https://cdn.sofifa.org/flags/ENGL.png">https://cdn.sofifa.org/flags/ENGL.png</a>

18207 rows × 89 columns



## Exploratory Data Analysis

preview the dataset

```
In [5]: df.head()
```

Out[5]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	C
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	

5 rows × 89 columns



## view summary of dataset

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 89 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Unnamed: 0        18207 non-null  int64  
 1   ID                18207 non-null  int64  
 2   Name               18207 non-null  object  
 3   Age                18207 non-null  int64  
 4   Photo              18207 non-null  object  
 5   Nationality        18207 non-null  object  
 6   Flag               18207 non-null  object  
 7   Overall            18207 non-null  int64  
 8   Potential           18207 non-null  int64  
 9   Club               17966 non-null  object  
 10  Club Logo          18207 non-null  object  
 11  Value              18207 non-null  object  
 12  Wage               18207 non-null  object  
 13  Special             18207 non-null  int64  
 14  Preferred Foot     18159 non-null  object  
 15  International Reputation 18159 non-null  float64 
 16  Weak Foot          18159 non-null  float64 
 17  Skill Moves         18159 non-null  float64 
 18  Work Rate           18159 non-null  object  
 19  Body Type           18159 non-null  object  
 20  Real Face           18159 non-null  object  
 21  Position             18147 non-null  object  
 22  Jersey Number       18147 non-null  float64 
 23  Joined              16654 non-null  object  
 24  Loaned From         1264 non-null   object  
 25  Contract Valid Until 17918 non-null  object  
 26  Height              18159 non-null  object  
 27  Weight              18159 non-null  object  
 28  LS                  16122 non-null  object  
 29  ST                  16122 non-null  object  
 30  RS                  16122 non-null  object  
 31  LW                  16122 non-null  object  
 32  LF                  16122 non-null  object  
 33  CF                  16122 non-null  object  
 34  RF                  16122 non-null  object  
 35  RW                  16122 non-null  object  
 36  LAM                 16122 non-null  object
```

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

```
79 Composure           18159 non-null float64
80 Marking             18159 non-null float64
81 StandingTackle      18159 non-null float64
82 SlidingTackle       18159 non-null float64
83 GKDiving            18159 non-null float64
84 GKHandling           18159 non-null float64
85 GKKicking             18159 non-null float64
86 GKPositioning         18159 non-null float64
87 GKReflexes            18159 non-null float64
88 Release Clause        16643 non-null object
dtypes: float64(38), int64(6), object(45)
memory usage: 12.4+ MB
```

```
In [7]: df['Body Type'].value_counts()
```

```
Out[7]: Body Type
Normal              10595
Lean                 6417
Stocky                140
Messi                  1
C. Ronaldo               1
Neymar                  1
Courtois                  1
PLAYER_BODY_TYPE_25          1
Shaqiri                  1
Akinfenwa                  1
Name: count, dtype: int64
```

## Comment

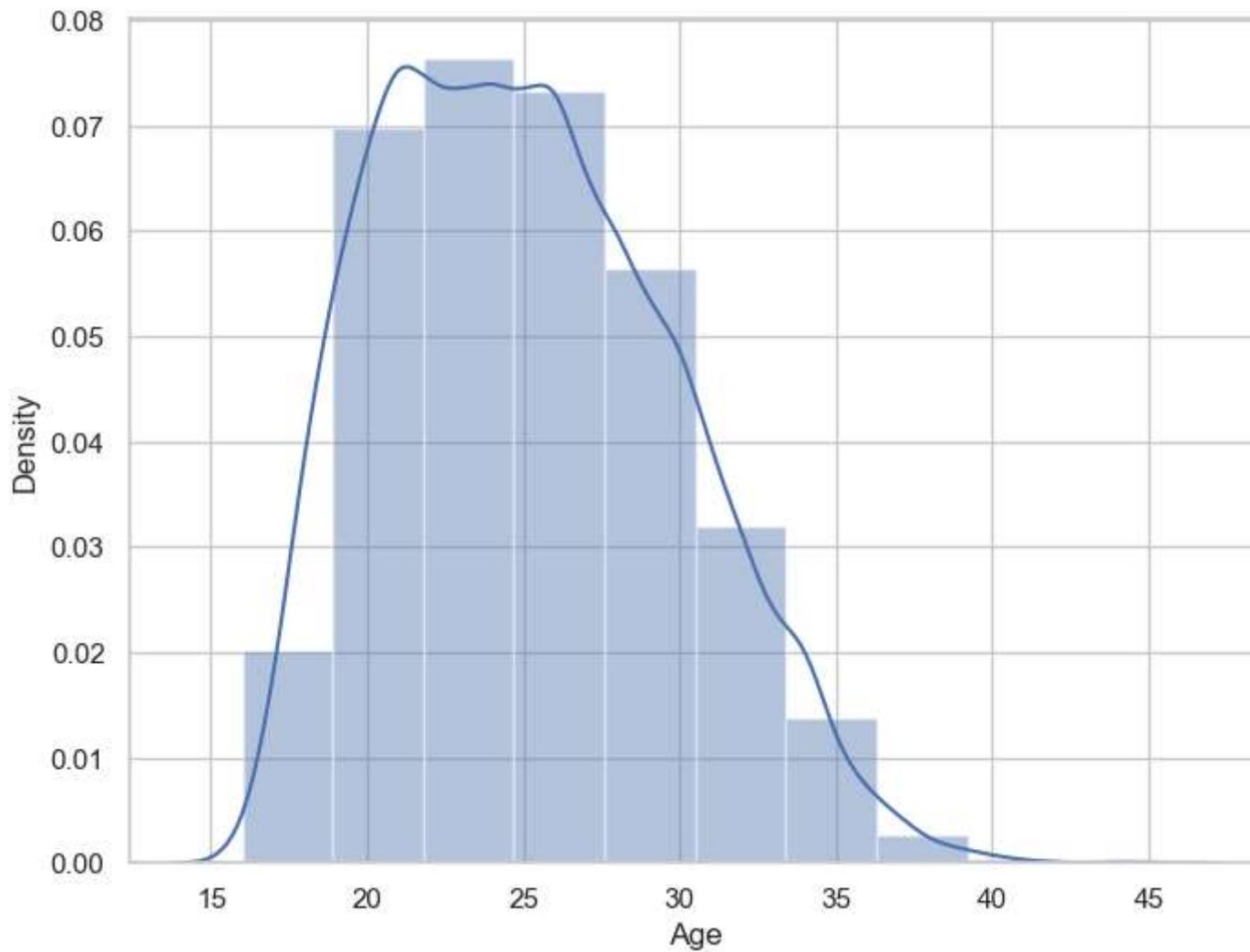
- This dataset contains 89 variables.
- Out of the 89 variables, 44 are numerical variables. 38 are of float64 data type and remaining 6 are of int64 data type.
- The remaining 45 variables are of character data type.

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

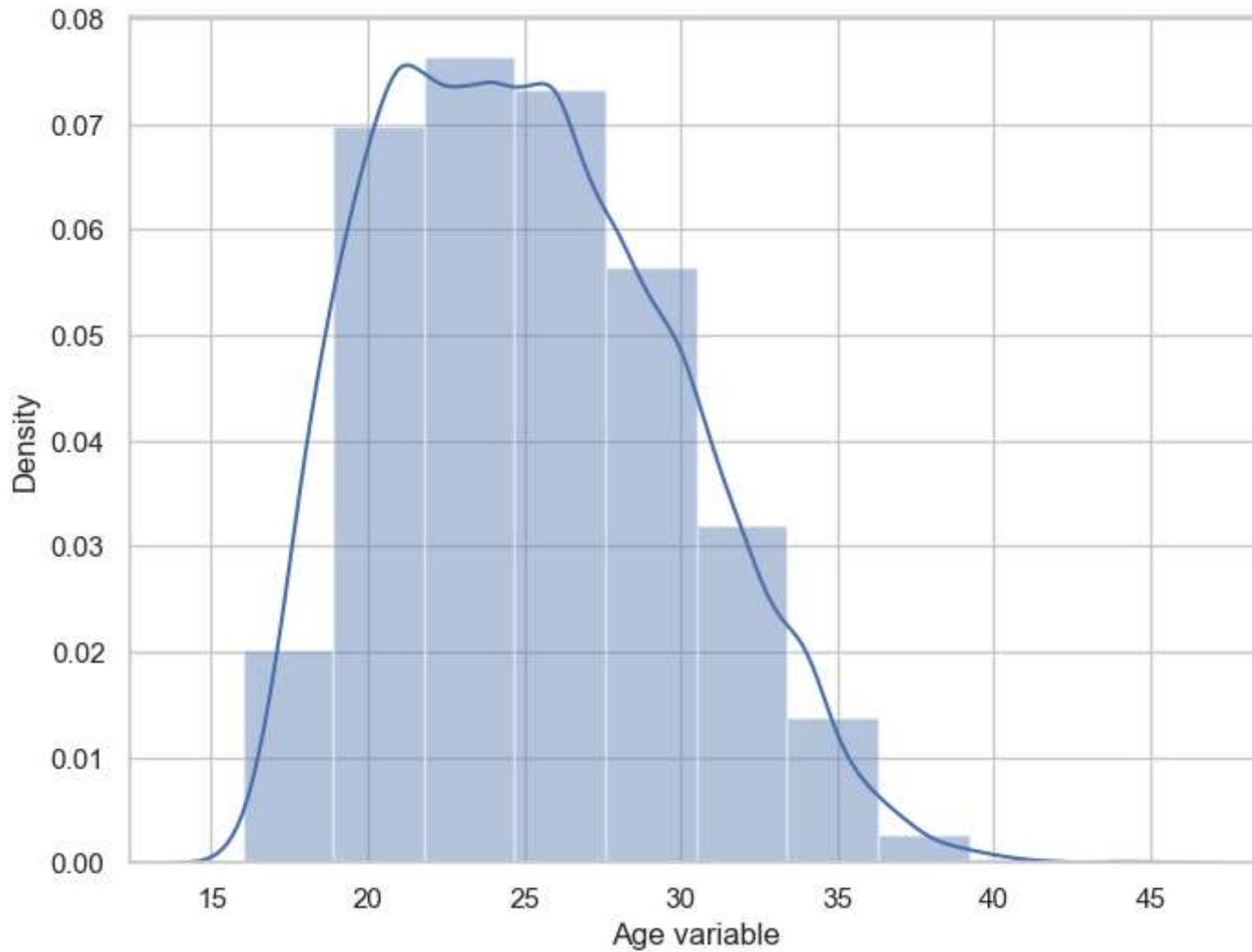


## Comment

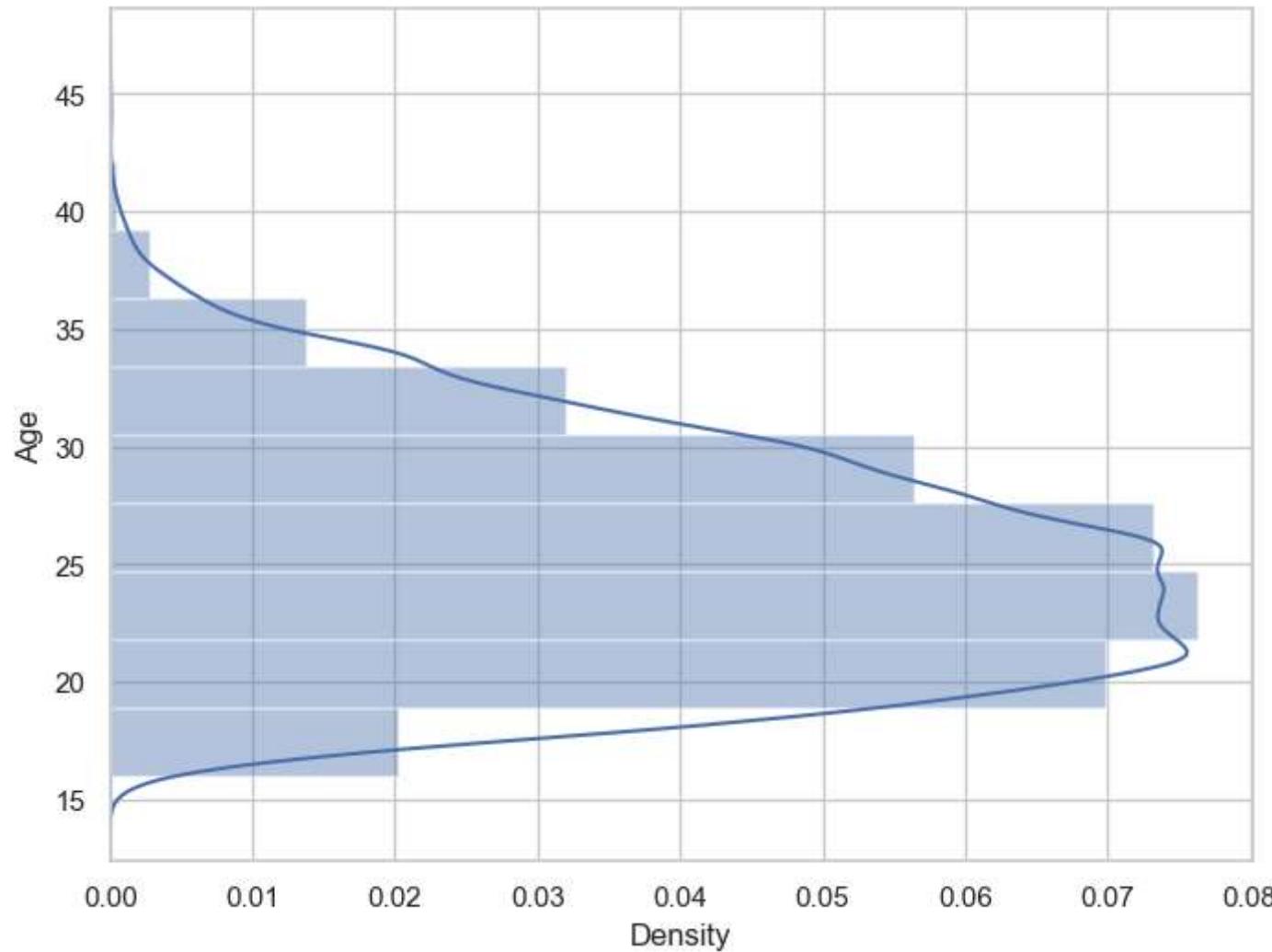
- It can be seen that the `Age` variable is slightly positively skewed.

```
In [9]: f, ax = plt.subplots(figsize=(8,6))
x = df['Age']
x = pd.Series(x, name="Age variable")
```

```
ax = sns.distplot(x, bins=10)
plt.show()
```



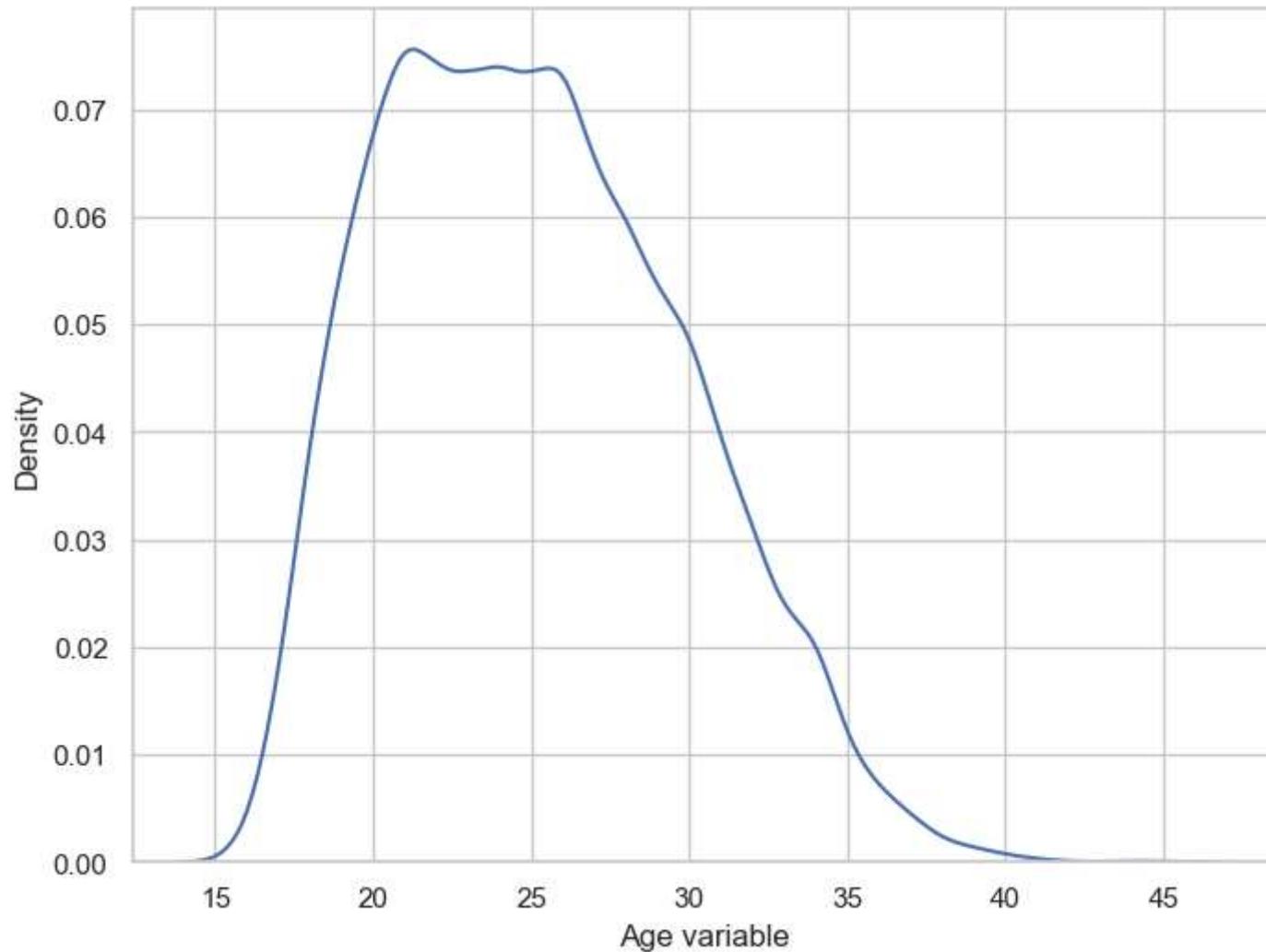
```
In [10]: f, ax = plt.subplots(figsize=(8,6))
x = df['Age']
ax = sns.distplot(x, bins=10, vertical = True)
plt.show()
```



## Seaborn Kernel Density Estimation (KDE) Plot

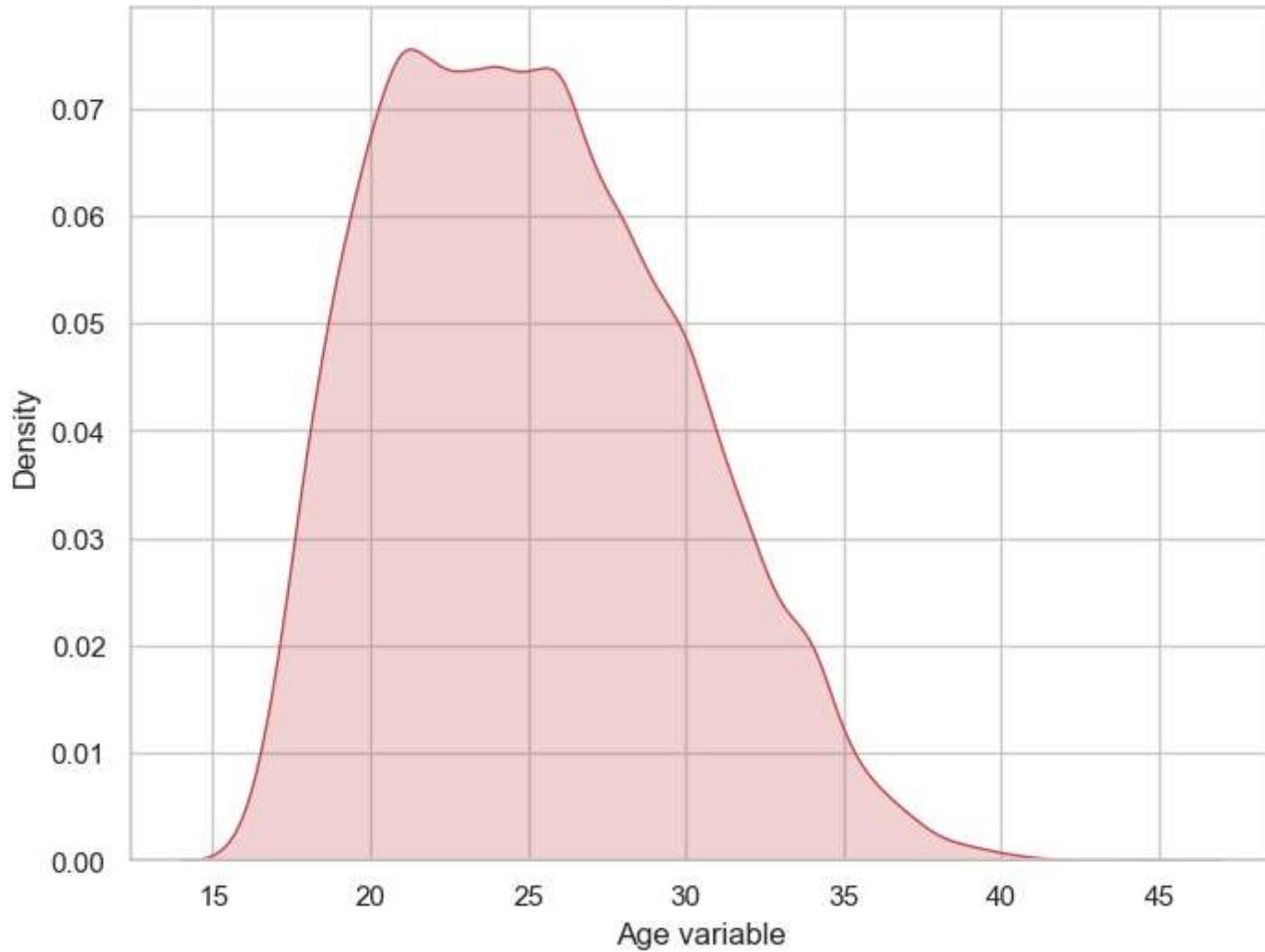
- The `kernel density estimate (KDE)` plot is a useful tool for plotting the shape of a distribution.
- Seaborn `kdeplot` is another seaborn plotting function that fits and plots a univariate or bivariate kernel density estimate.
- Like the histogram, the KDE plots encode the density of observations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

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



```
In [12]: f, ax = plt.subplots(figsize=(8,6))
x = df['Age']
x = pd.Series(x, name="Age variable")
```

```
ax = sns.kdeplot(x , shade =True , color='r')
plt.show()
```



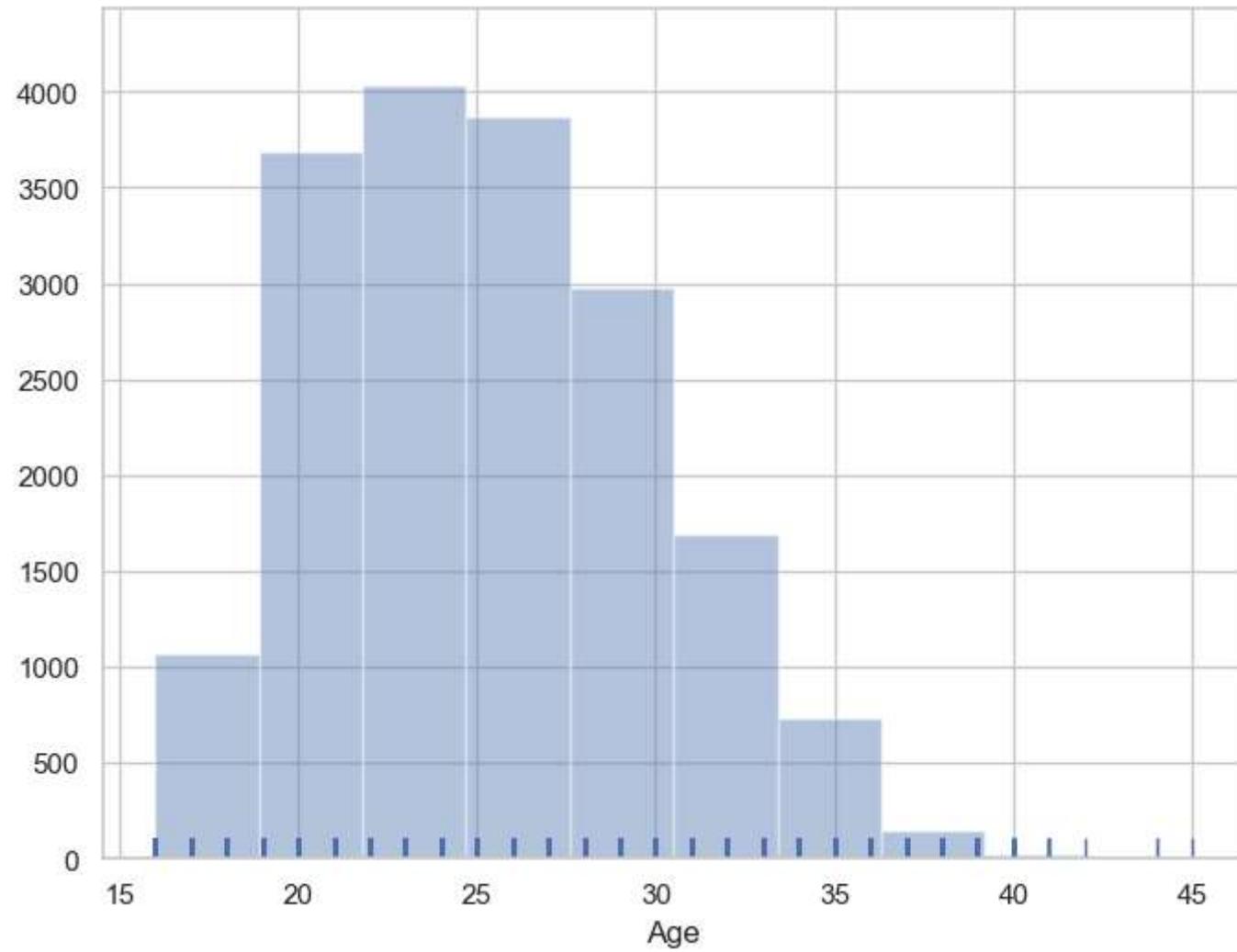
## Histograms

- 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.

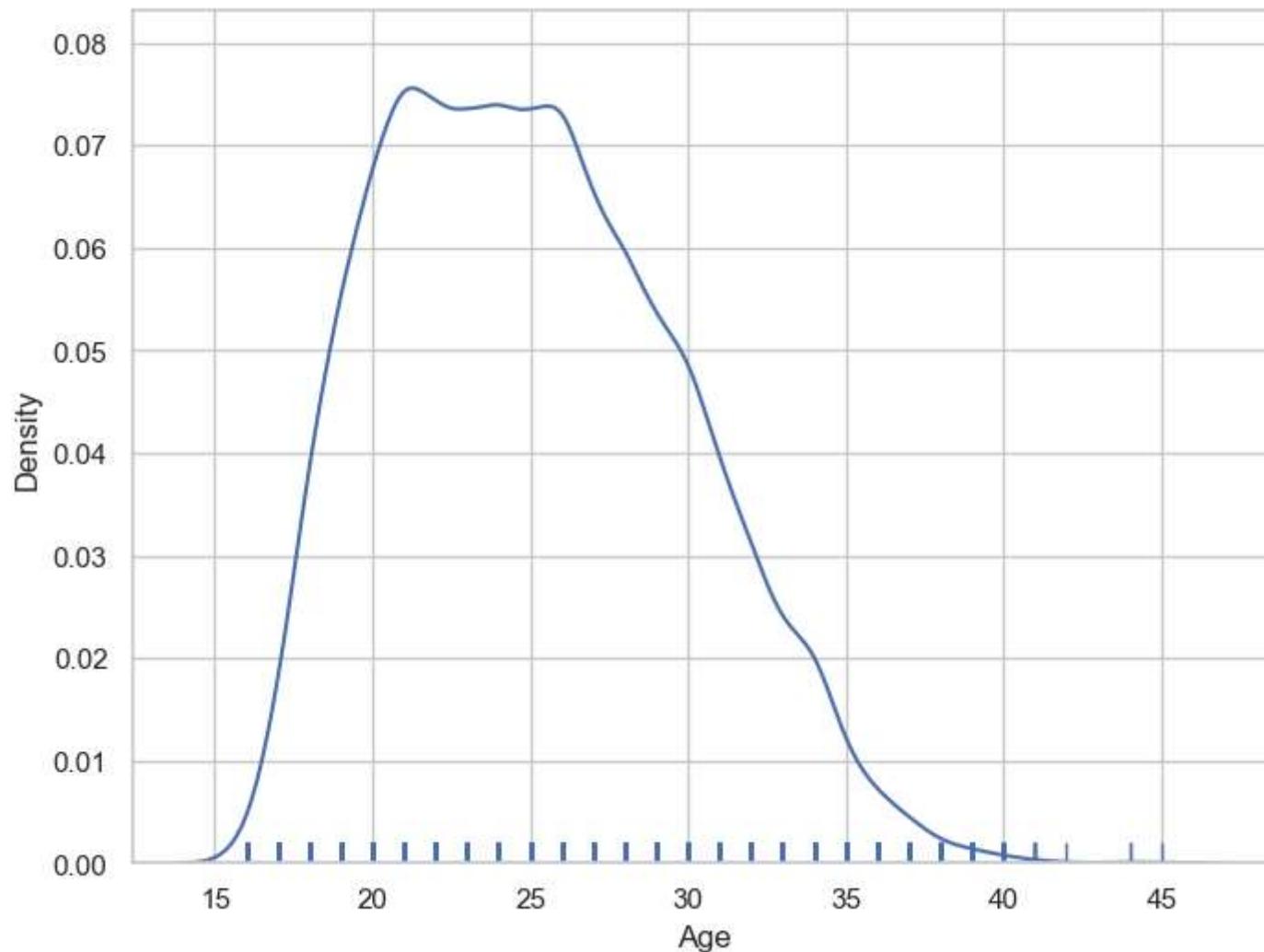
```
In [13]:
```

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



```
In [14]:
```

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



Explored Preferred Foot variable

## check number of nunique values in preferred foot variable

```
In [15]: df['Preferred Foot'].nunique()
```

```
Out[15]: 2
```

## check frequency distribution of values in preferred foot variable

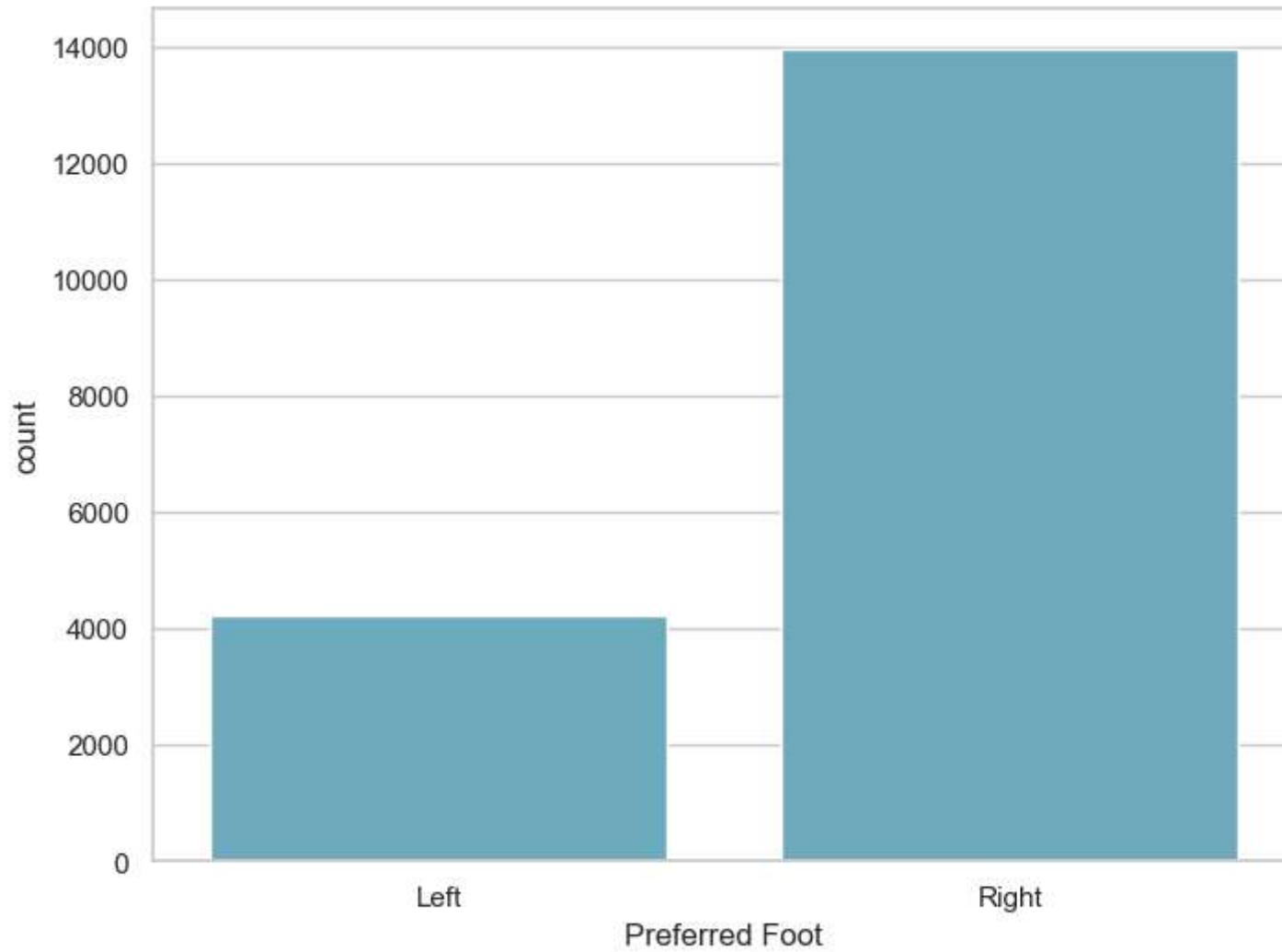
```
In [16]: df['Preferred Foot'].value_counts()
```

```
Out[16]: Preferred Foot
Right      13948
Left       4211
Name: count, dtype: int64
```

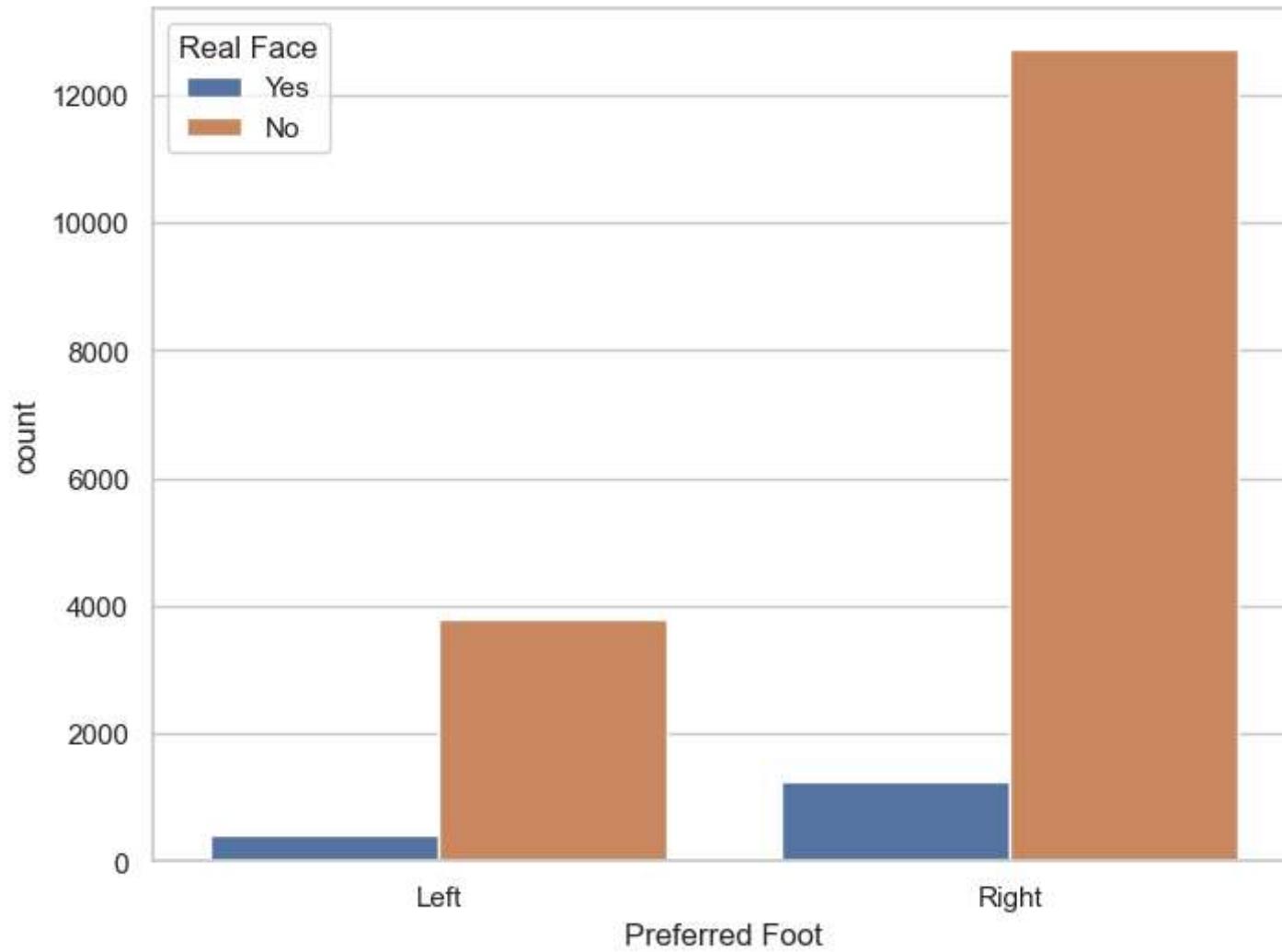
## Visualize distribution of values with Seaborn countplot() function.

- 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.

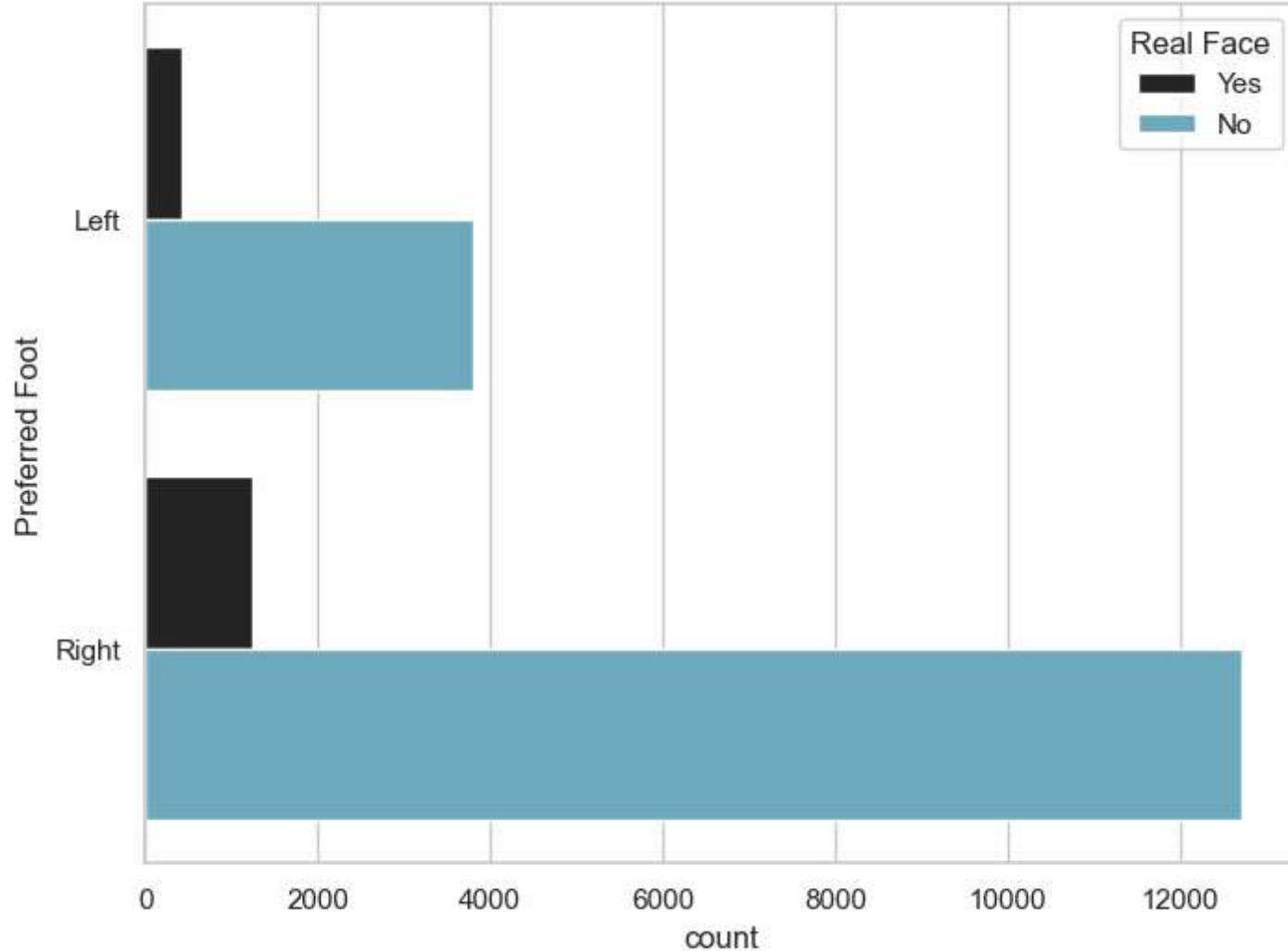
```
In [17]: f, ax = plt.subplots(figsize=(8, 6))
sns.countplot(x="Preferred Foot", data=df, color="c")
plt.show()
```



```
In [18]: f, ax = plt.subplots(figsize=(8, 6))
sns.countplot(x="Preferred Foot", hue="Real Face", data=df)
plt.show()
```



```
In [19]: f, ax = plt.subplots(figsize=(8, 6))
sns.countplot(y="Preferred Foot", hue="Real Face", data=df,color='c')
plt.show()
```

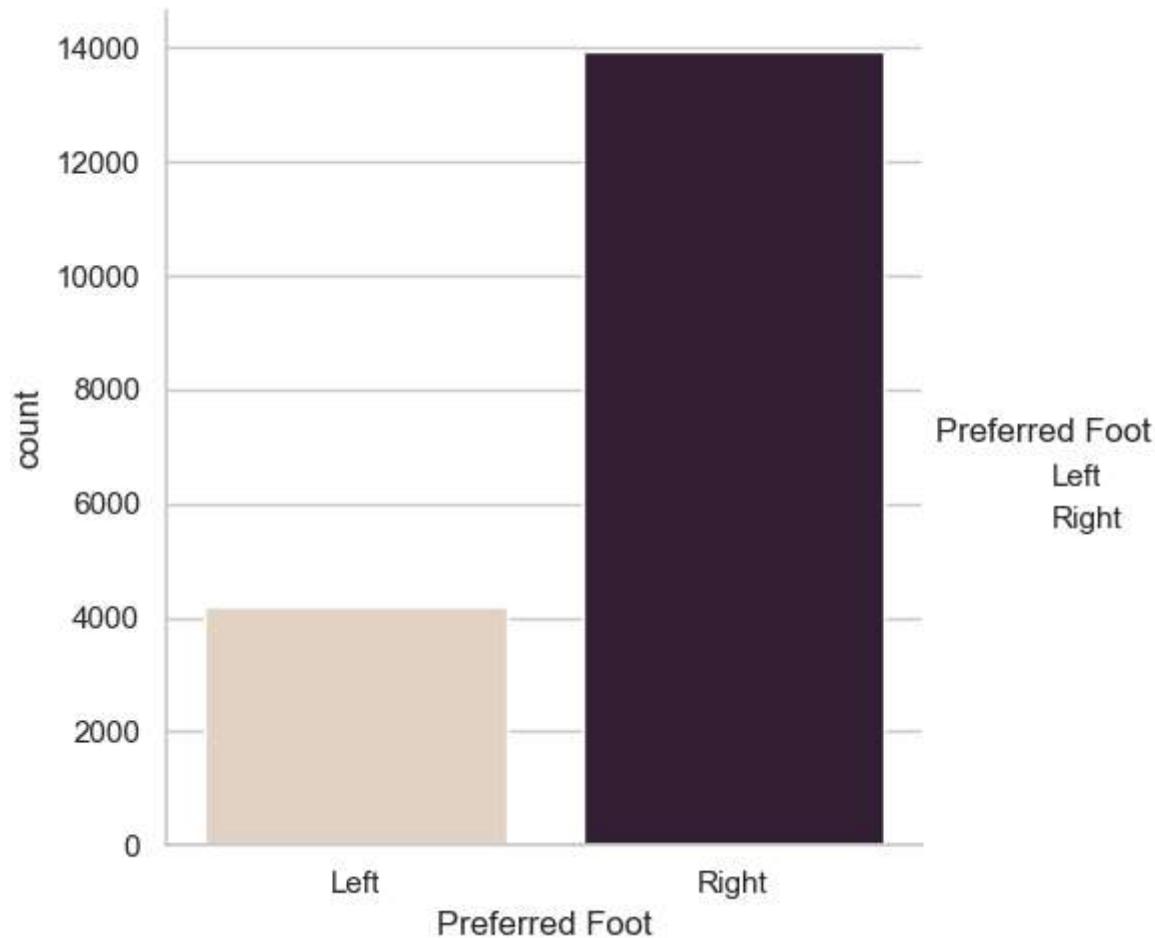


## Seaborn `Catplot()` function

- 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.

```
In [20]: g = sns.catplot(x="Preferred Foot", kind="count", palette="ch:.25", data=df)
```



Explore international reputation variable

check the number of unique values in 'International Reputation' variable

```
In [21]: df['International Reputation'].nunique()
```

```
Out[21]: 5
```

## check the distribution of values in international reputation variable

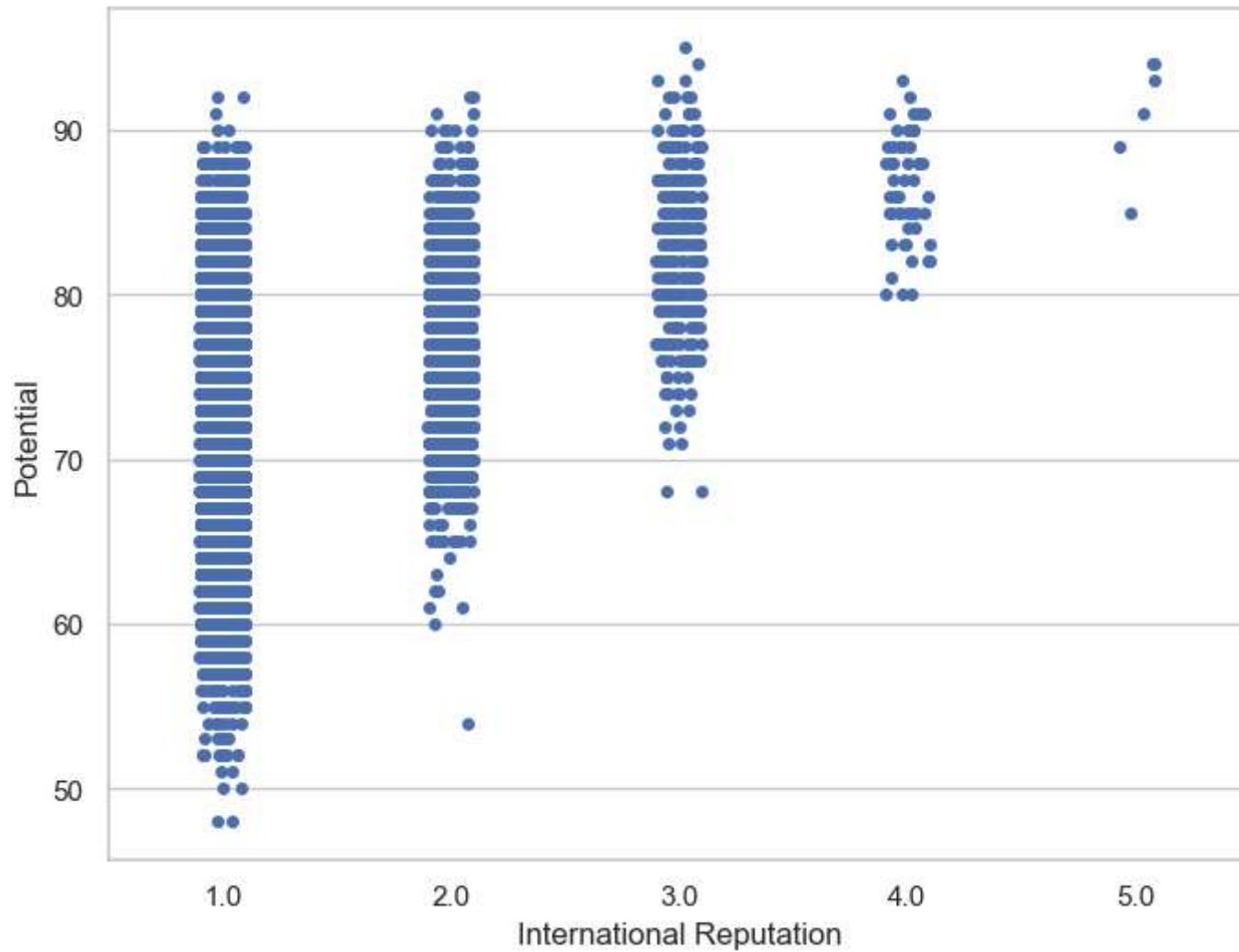
```
In [22]: df['International Reputation'].value_counts()
```

```
Out[22]: International Reputation
1.0    16532
2.0    1261
3.0     309
4.0      51
5.0       6
Name: count, dtype: int64
```

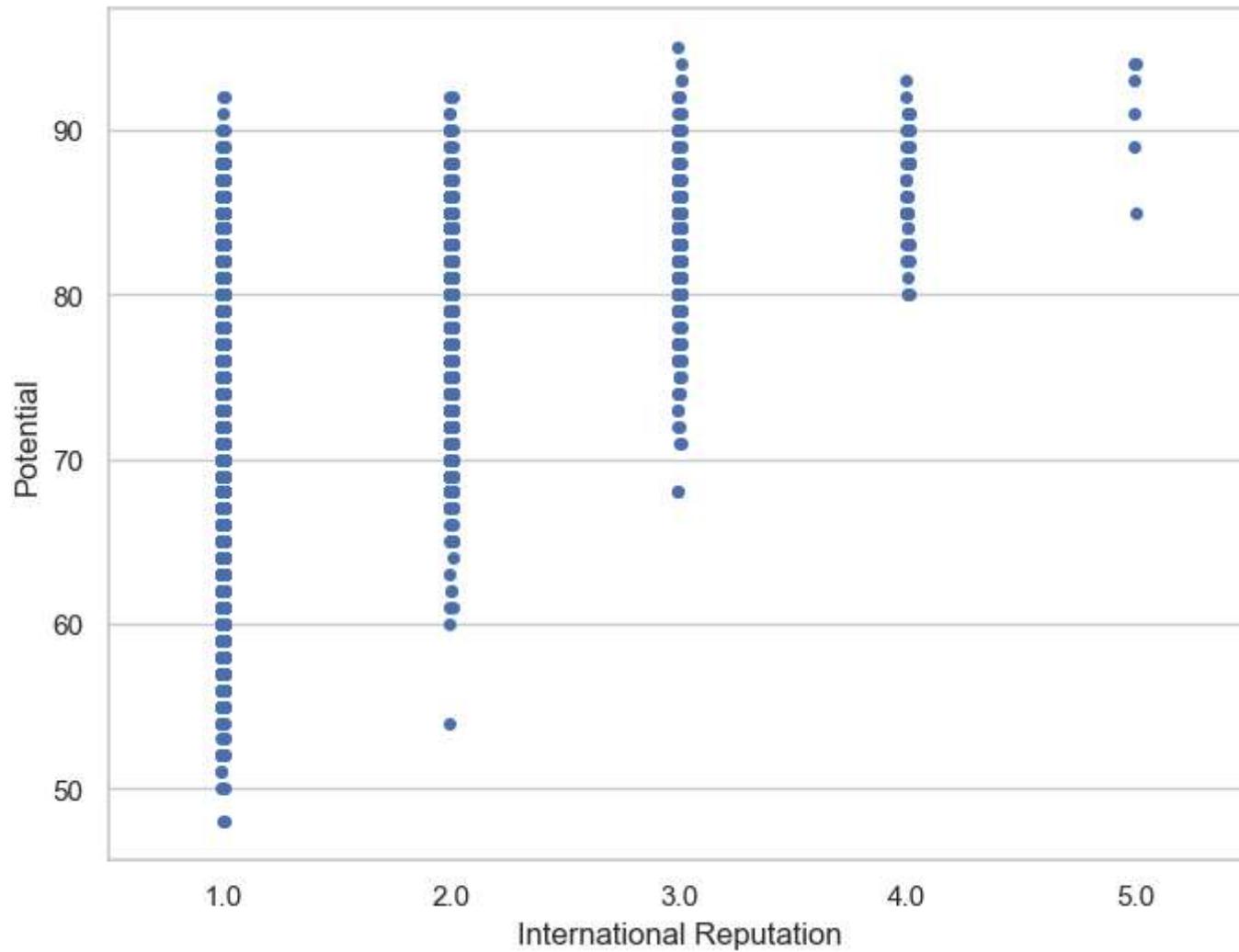
## Seaborn Stripplot() function

- 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.

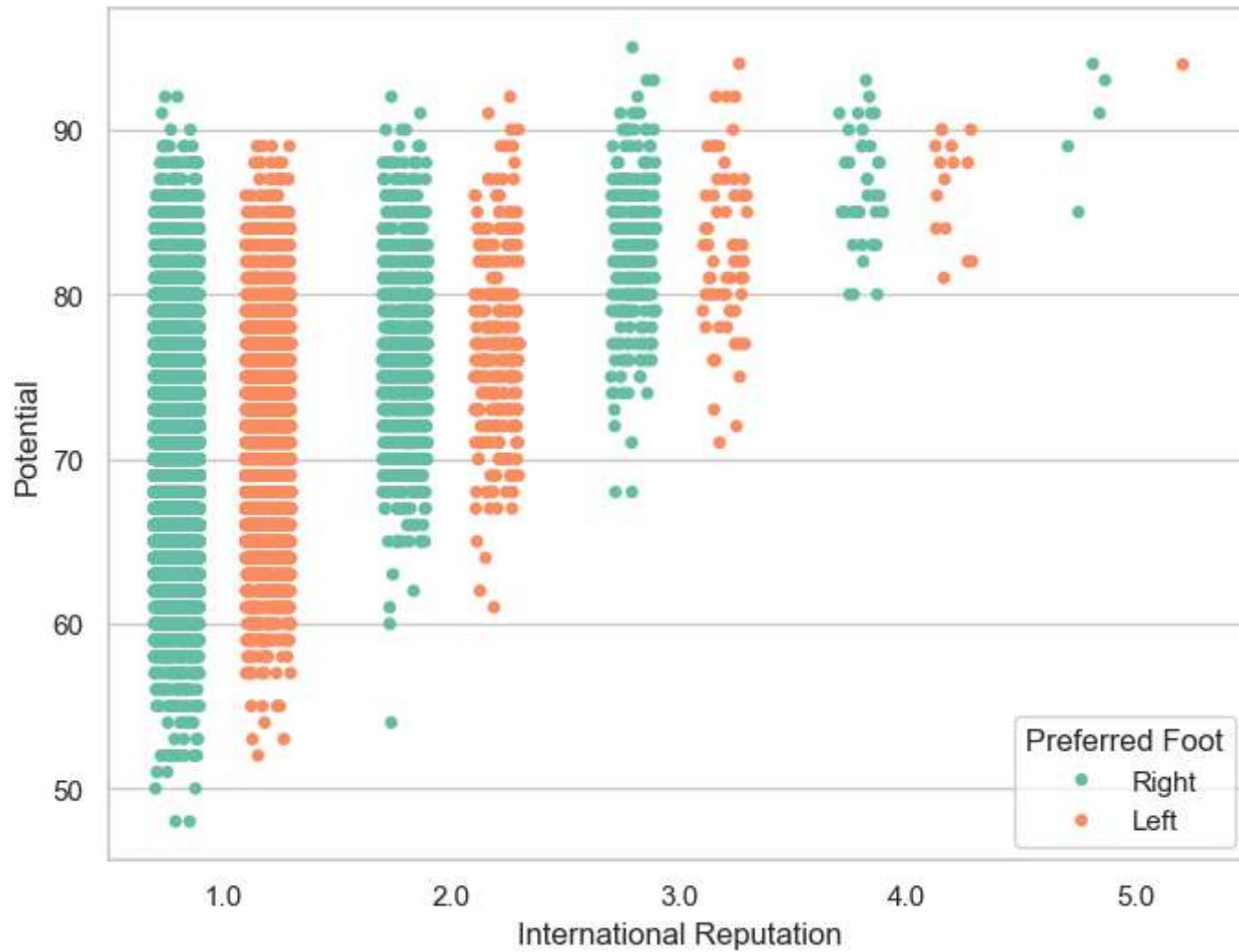
```
In [23]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", data=df)
plt.show()
```



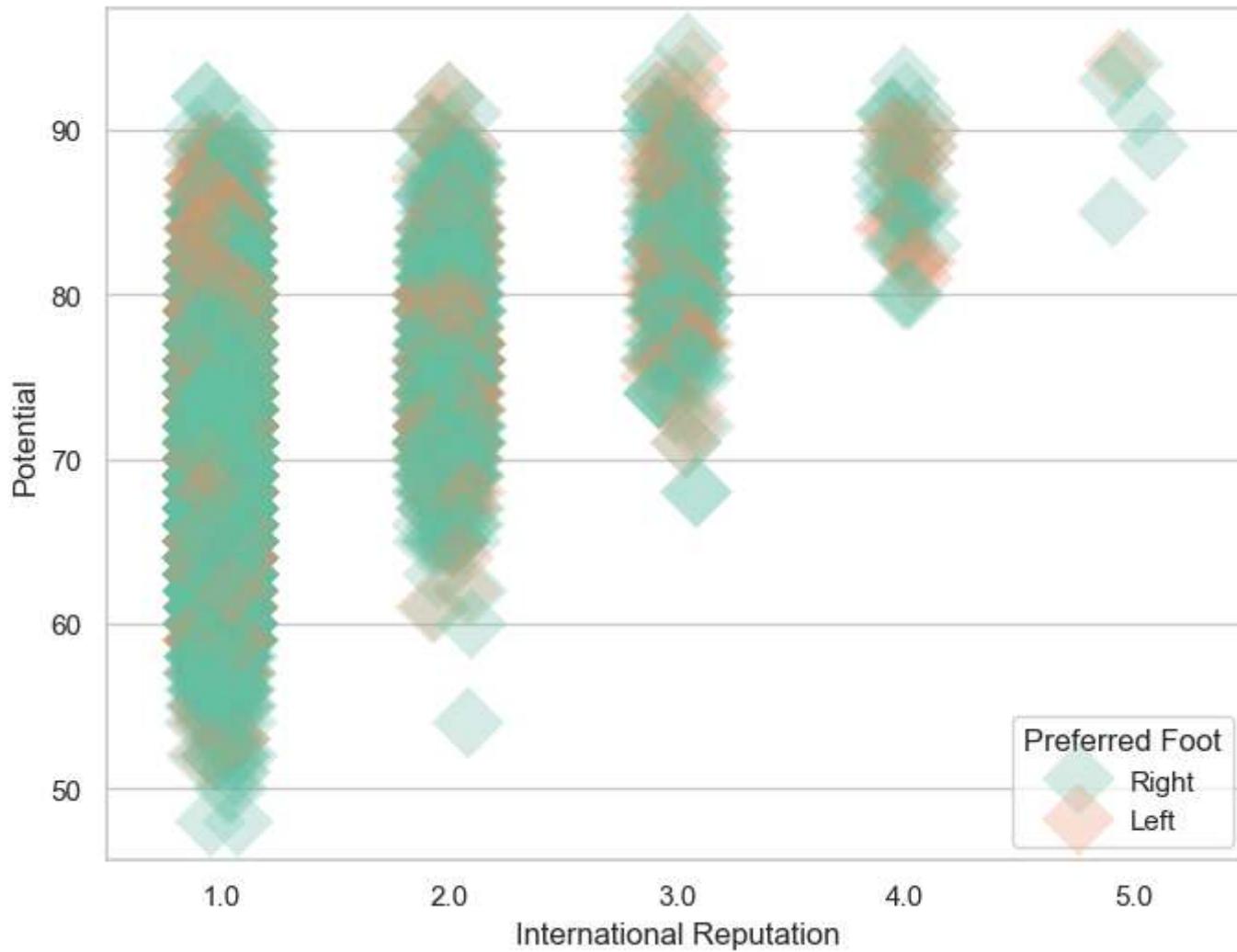
```
In [24]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", data=df, jitter=0.01)
plt.show()
```



```
In [25]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", data=df,hue = 'Preferred Foot',jitter=0.2,palette='Set2',order=[1,2,3,4,5])
plt.show()
```



```
In [26]: f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="International Reputation", y="Potential", hue="Preferred Foot",
              data=df, palette="Set2", size=20, marker="D",
              edgecolor="gray", alpha=.25)
plt.show()
```

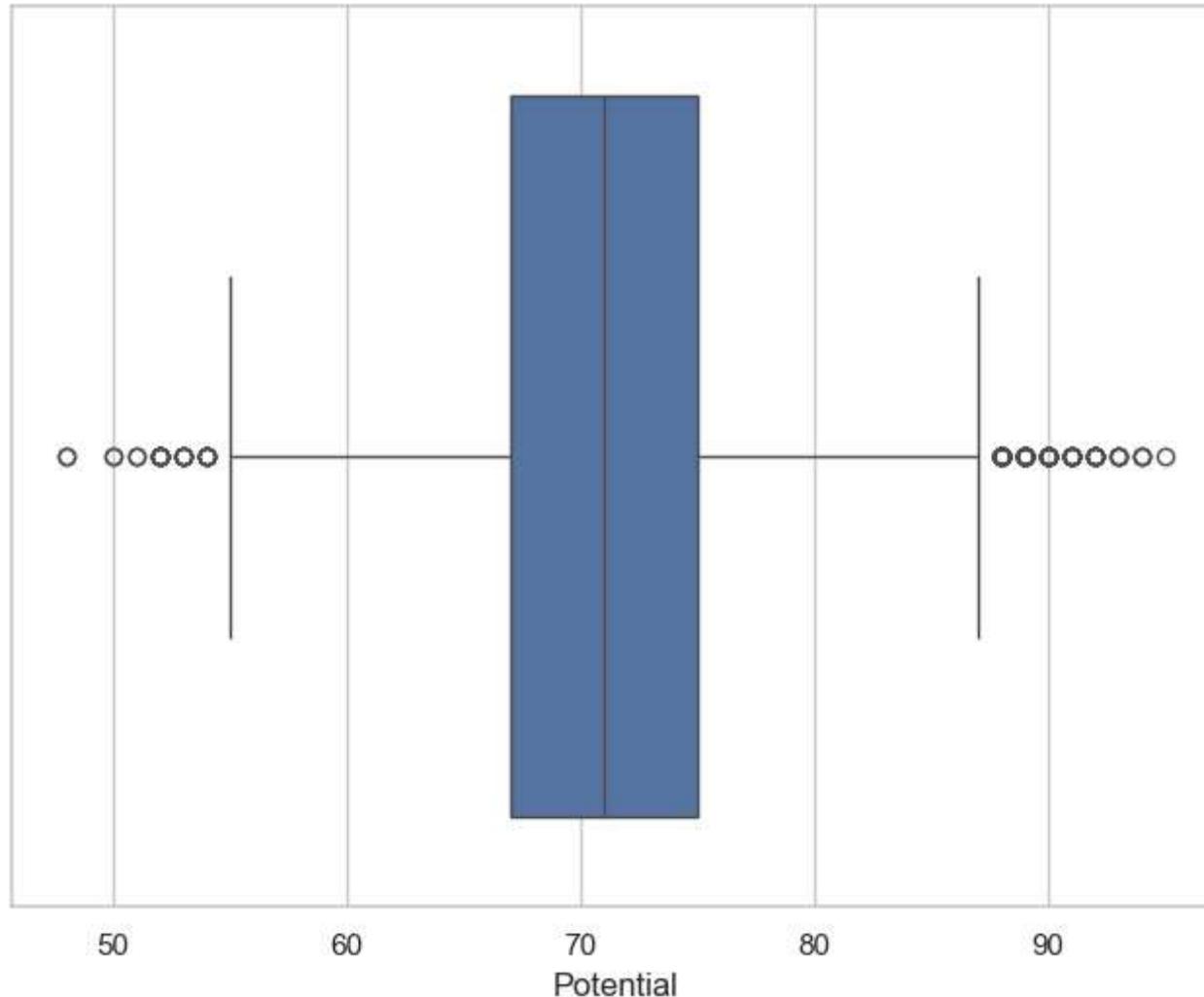


## Seaborn `boxplot()` function

- 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.

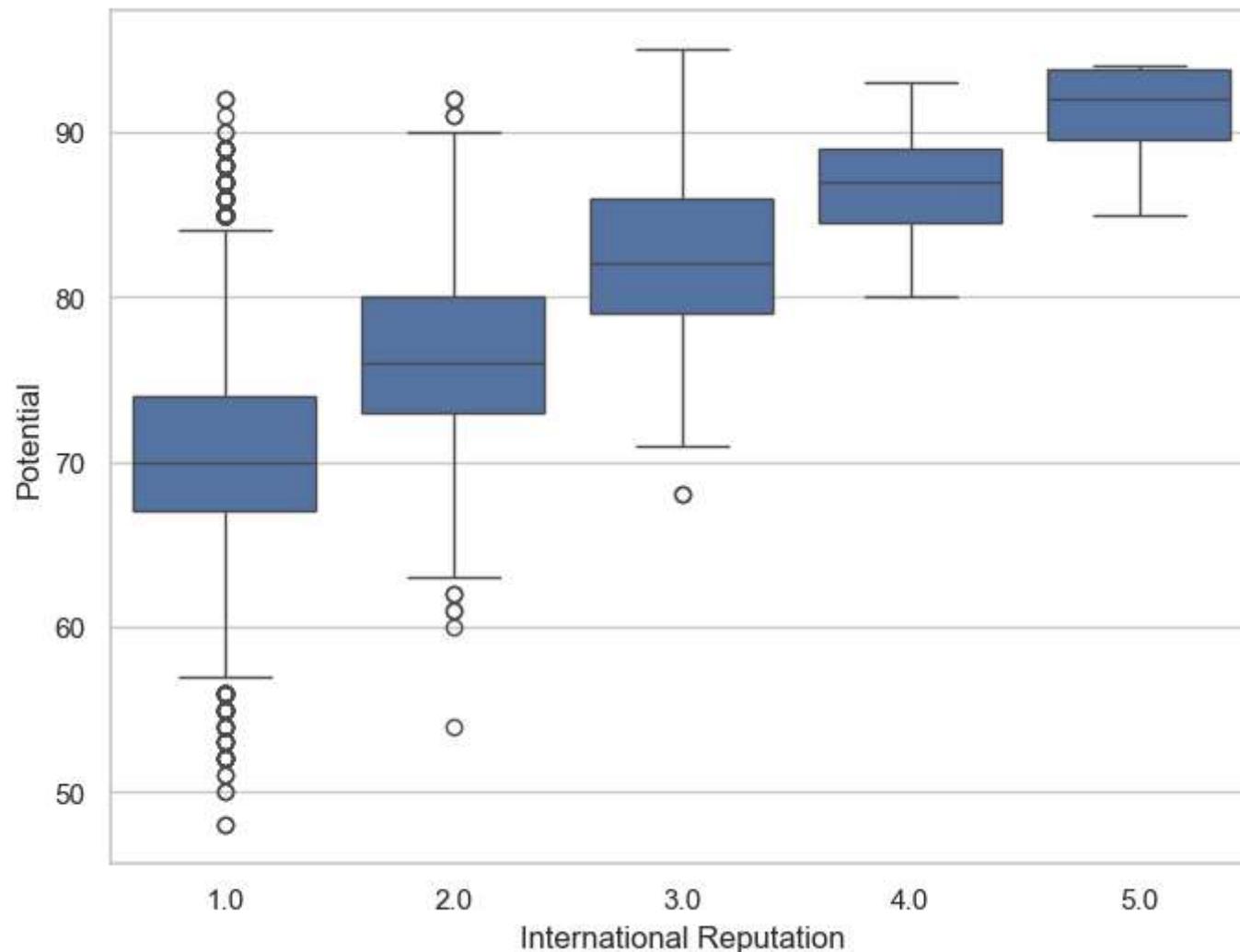
- The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

```
In [27]: f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["Potential"])
plt.show()
```

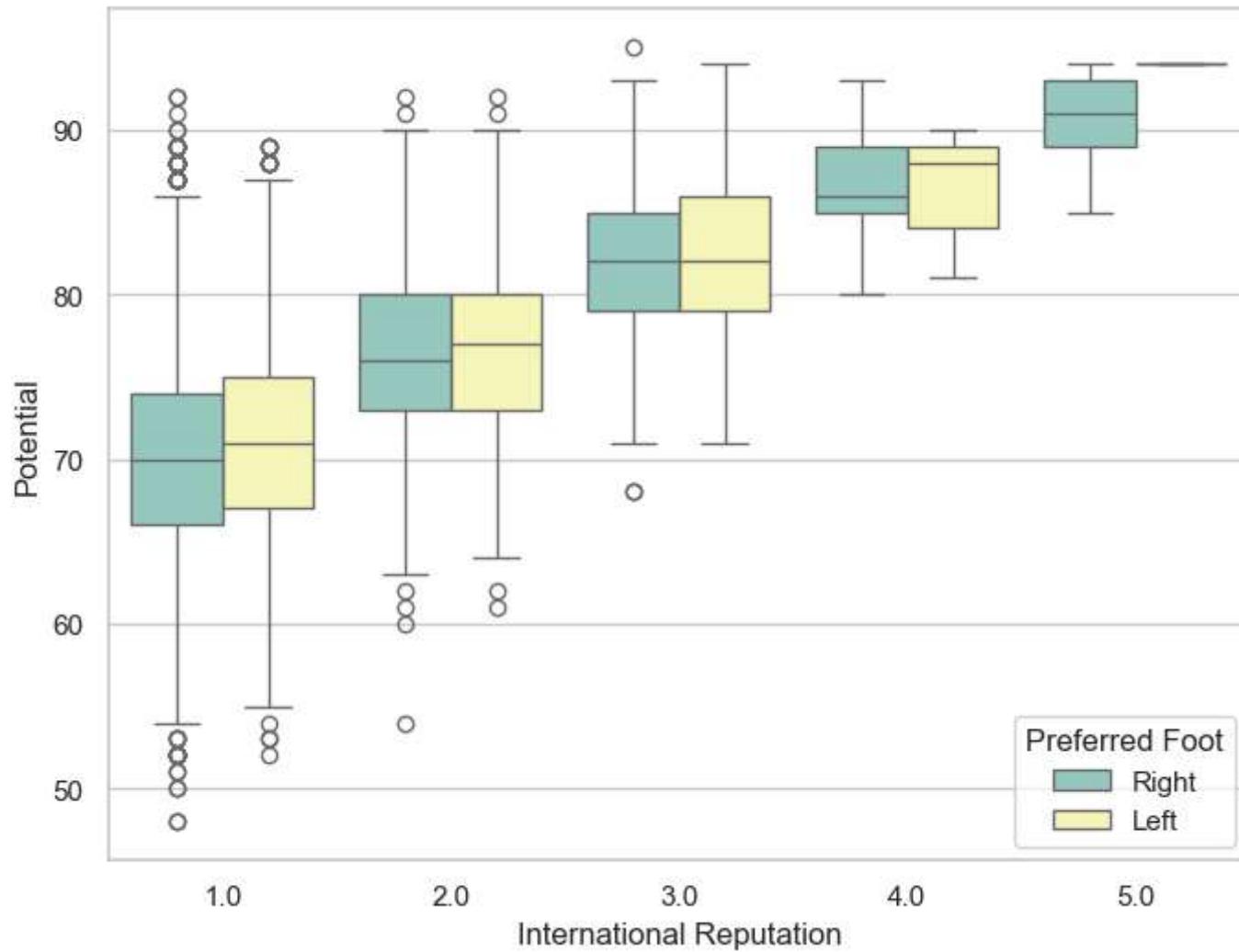


```
In [28]: f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="International Reputation", y="Potential", data=df)
```

```
plt.show()
```



```
In [29]: f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="International Reputation", y="Potential", hue="Preferred Foot", data=df, palette="Set3")
plt.show()
```

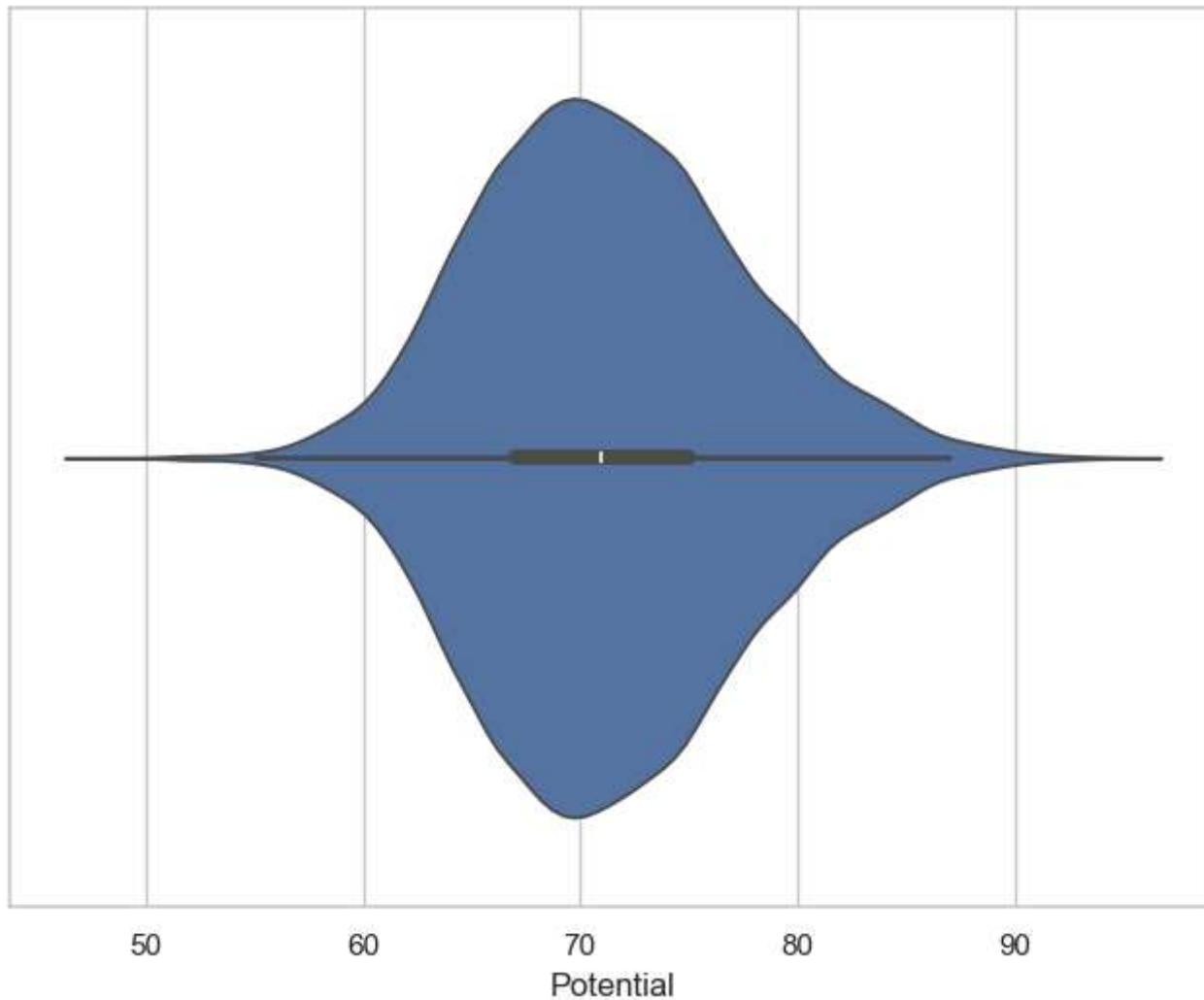


## Seaborn `violinplot()` function

- 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.

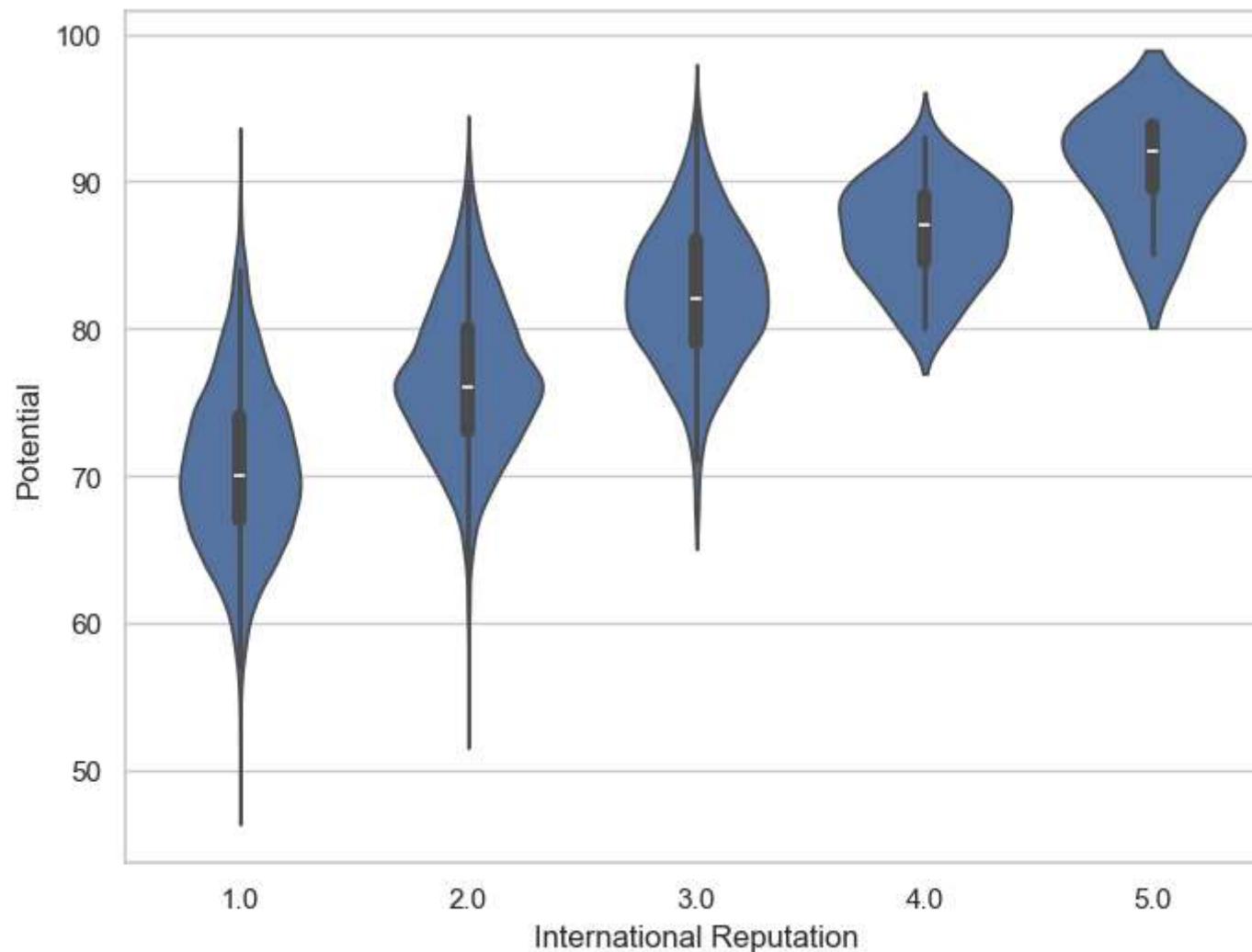
- Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

```
In [30]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x=df["Potential"])
plt.show()
```



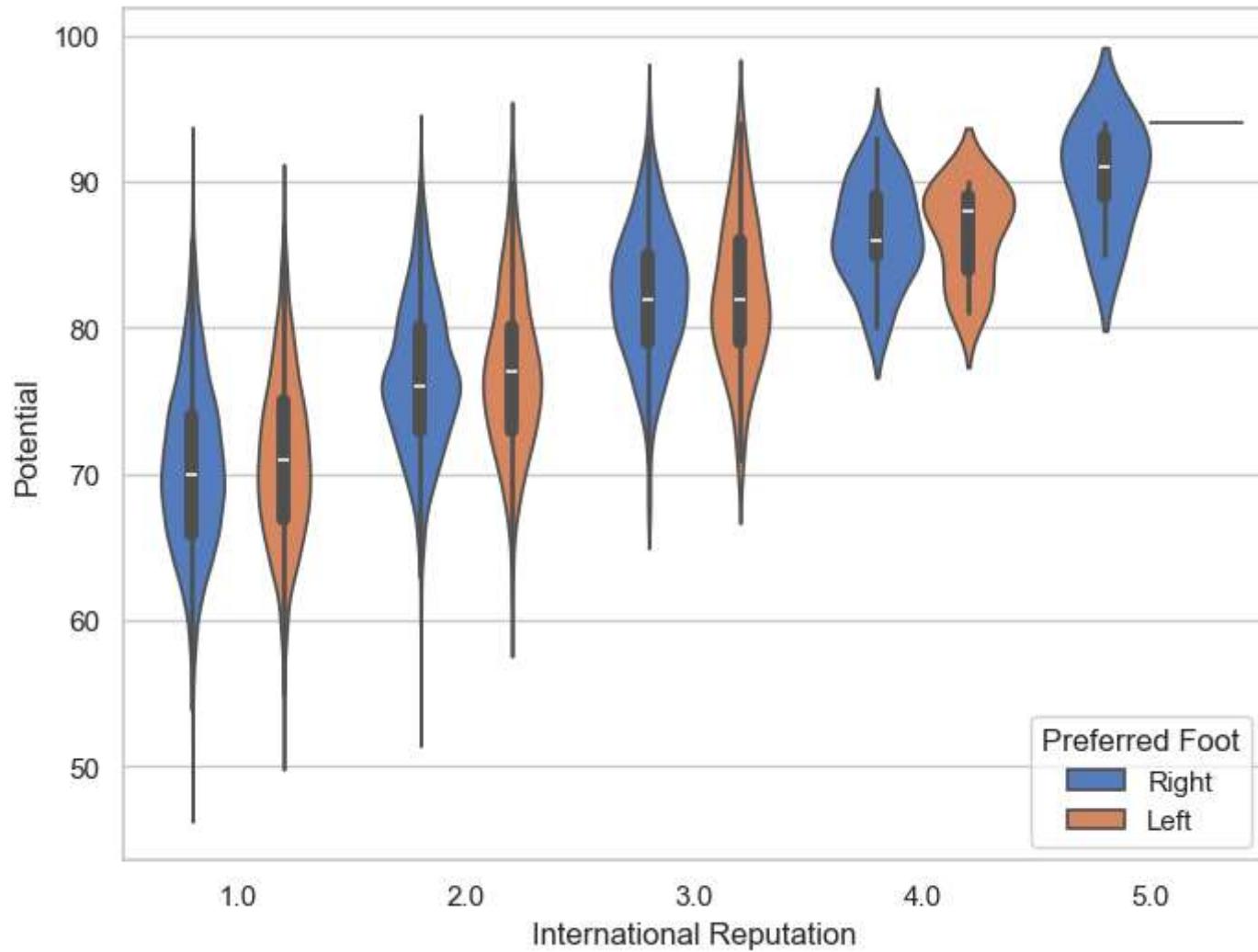
We can draw the vertical violinplot grouped by the categorical variable 'International Reputation' as follows-

```
In [31]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x="International Reputation", y="Potential", data=df)
plt.show()
```



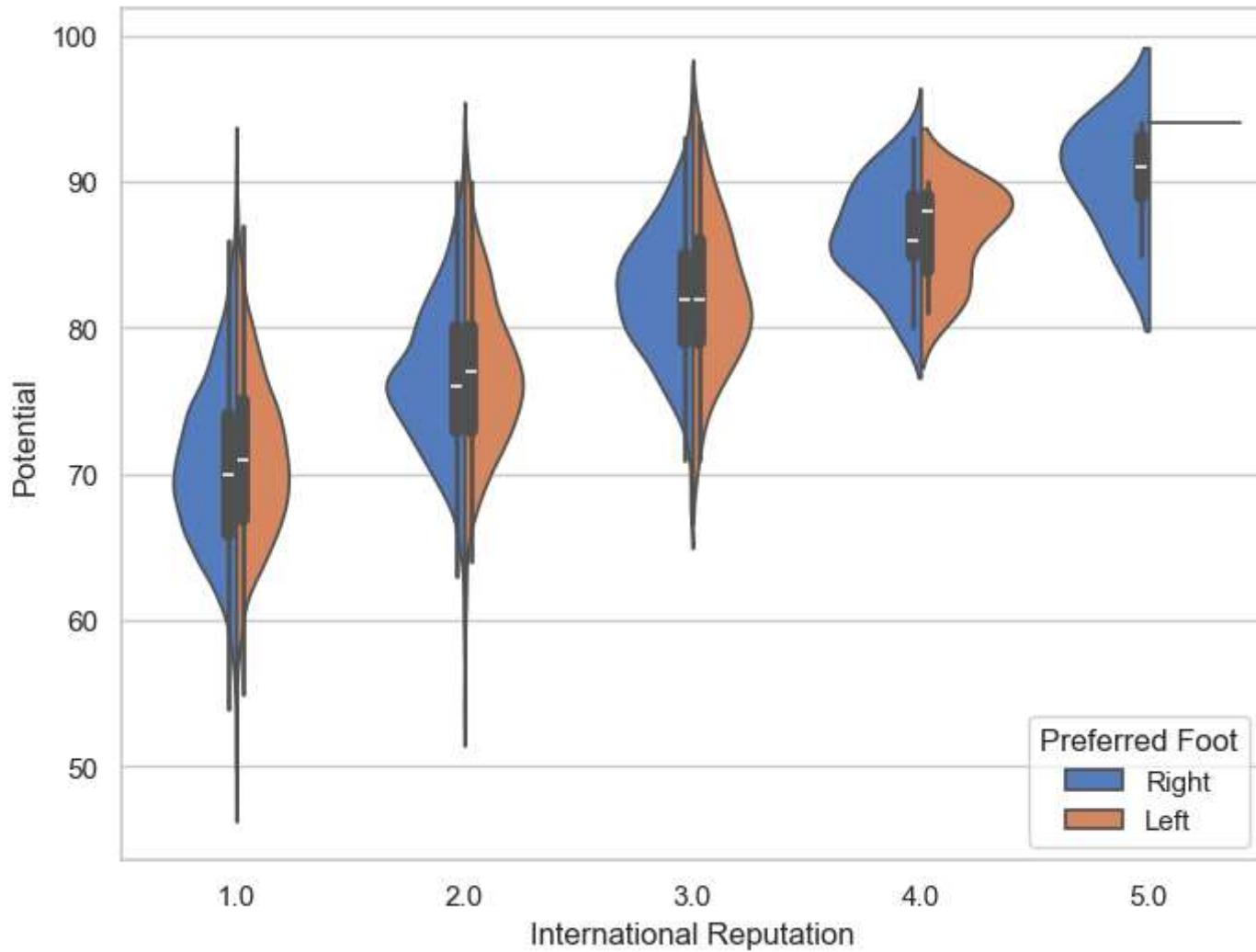
We can draw a violinplot with nested grouping by two categorical variables as follows-

```
In [32]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x="International Reputation", y="Potential", hue="Preferred Foot", data=df, palette="muted")
plt.show()
```



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

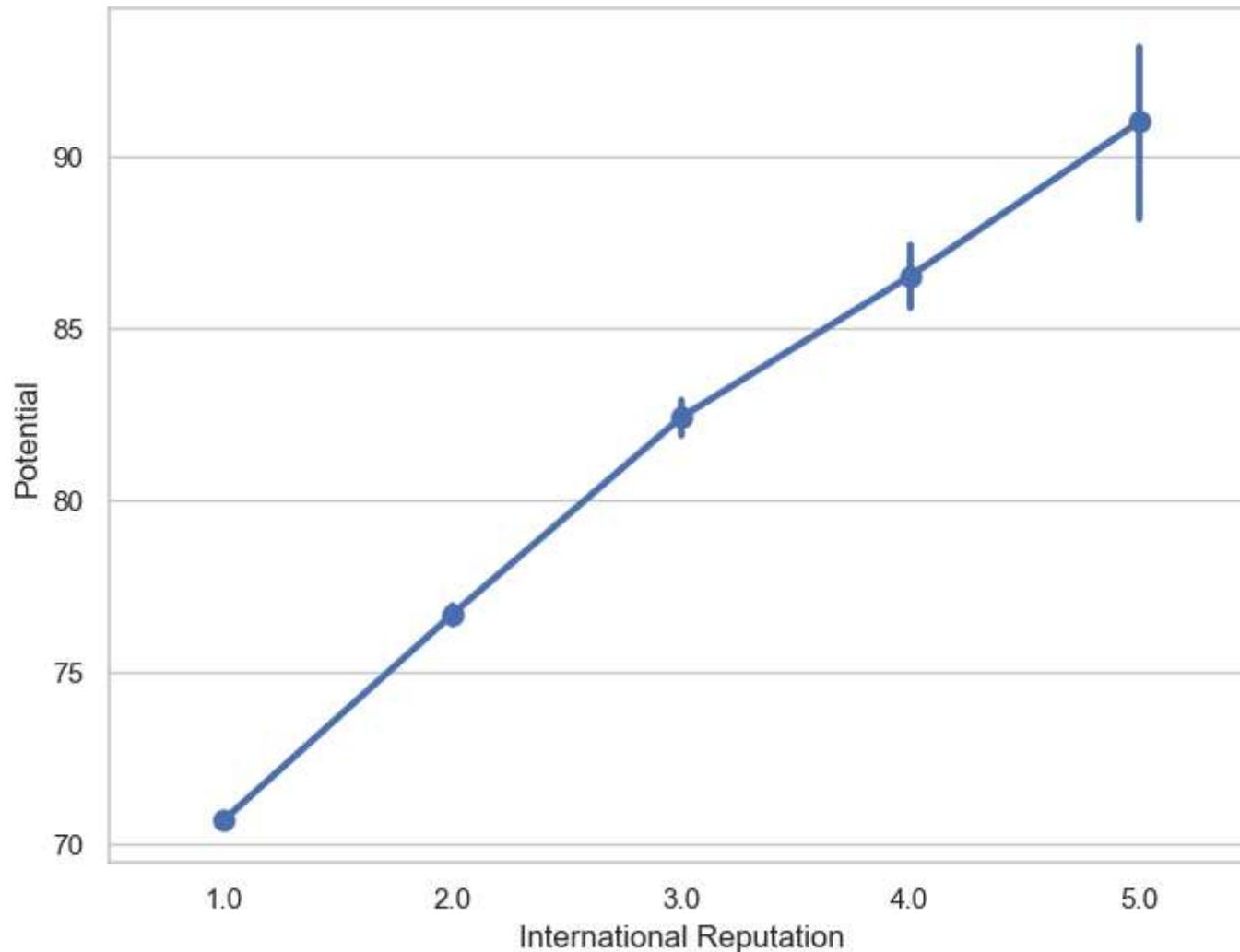
```
In [33]: f, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x="International Reputation", y="Potential", hue="Preferred Foot",
                 data=df, palette="muted", split=True)
plt.show()
```



## Seaborn `pointplot()` function

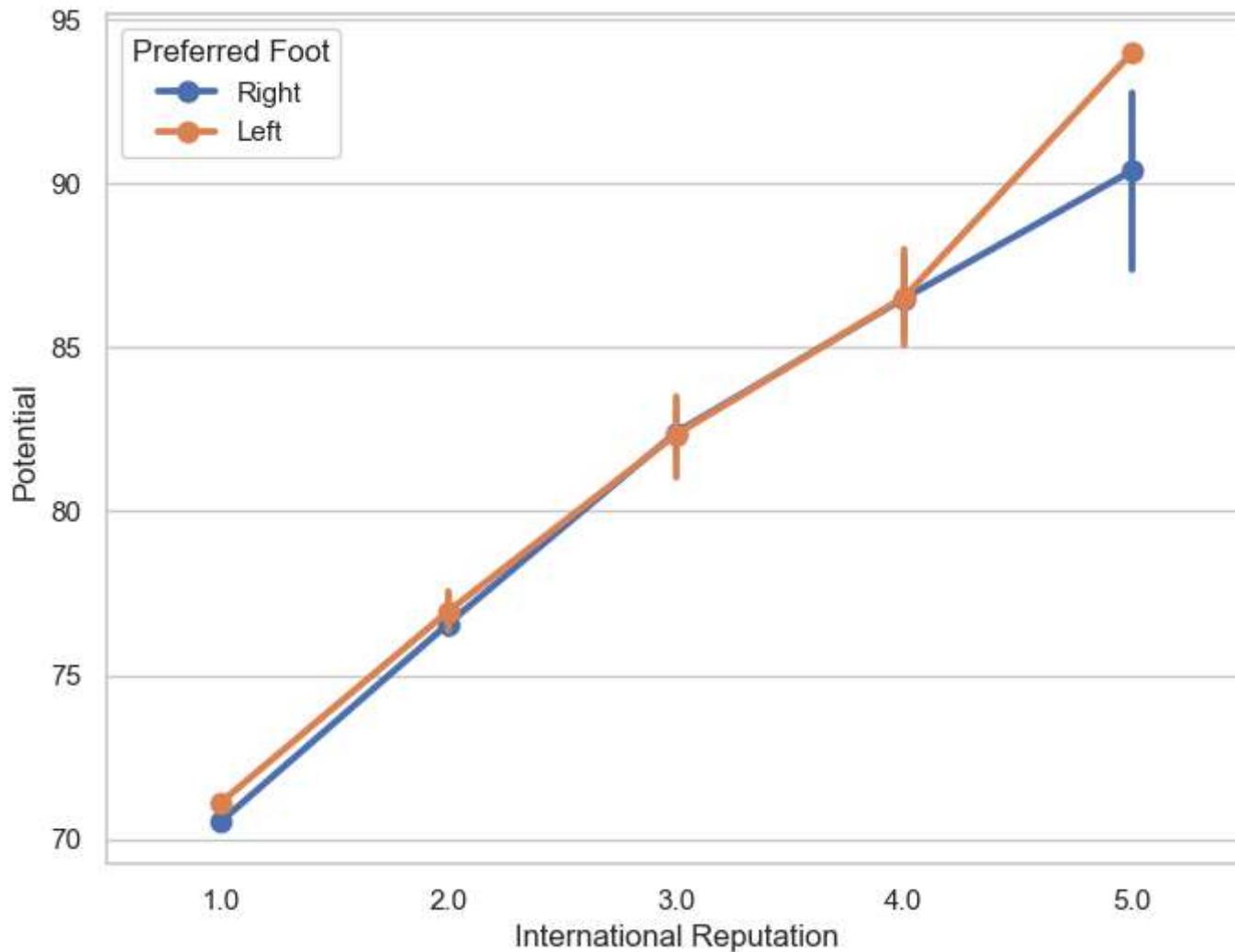
- This function show point estimates and confidence intervals using scatter plot glyphs.
- A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.

```
In [34]: f, ax = plt.subplots(figsize=(8, 6))
sns.pointplot(x="International Reputation", y="Potential", data=df)
plt.show()
```



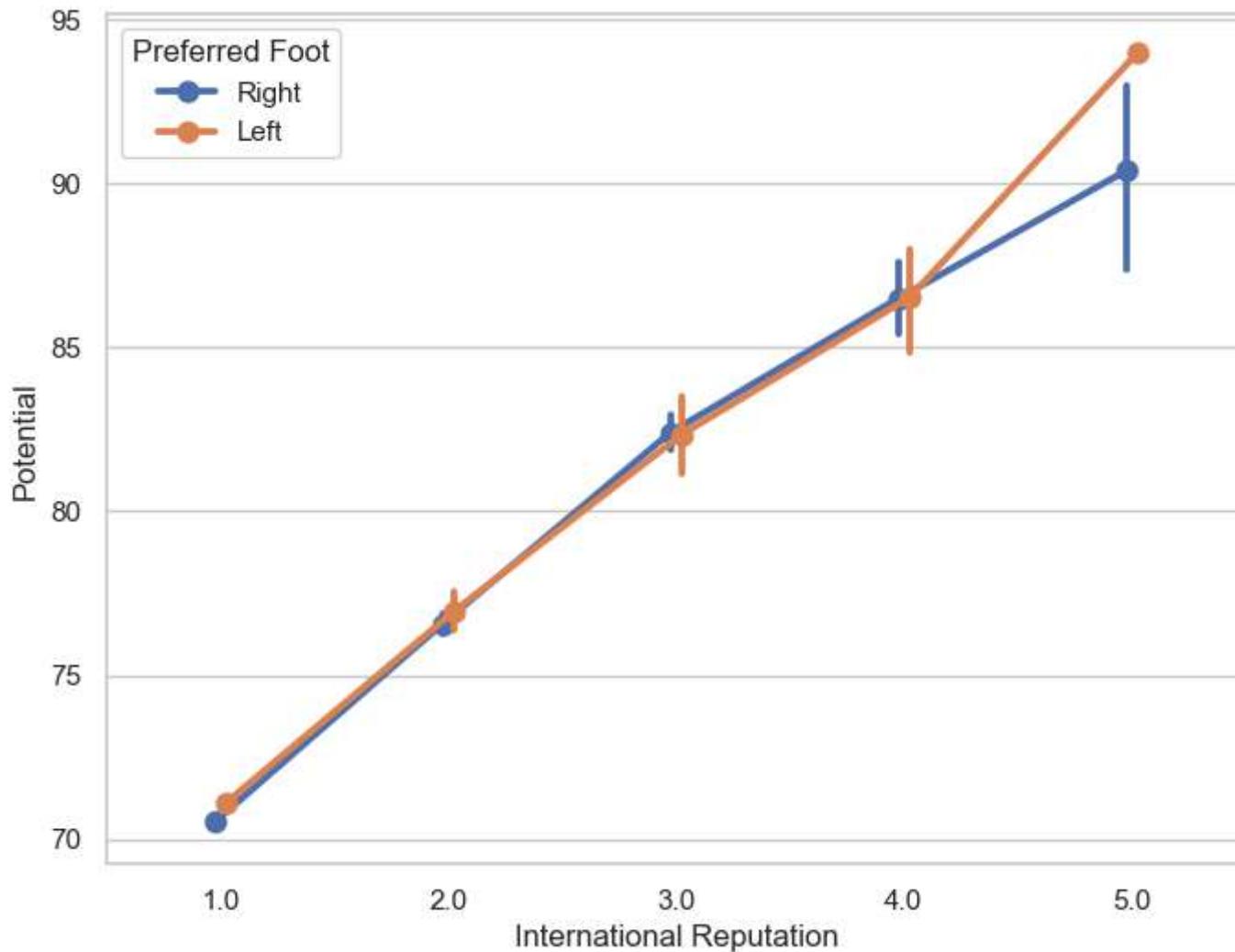
We can draw a set of vertical points with nested grouping by a two variables as follows-

```
In [35]: f, ax = plt.subplots(figsize=(8, 6))
sns.pointplot(x="International Reputation", y="Potential", data=df, hue="Preferred Foot")
plt.show()
```



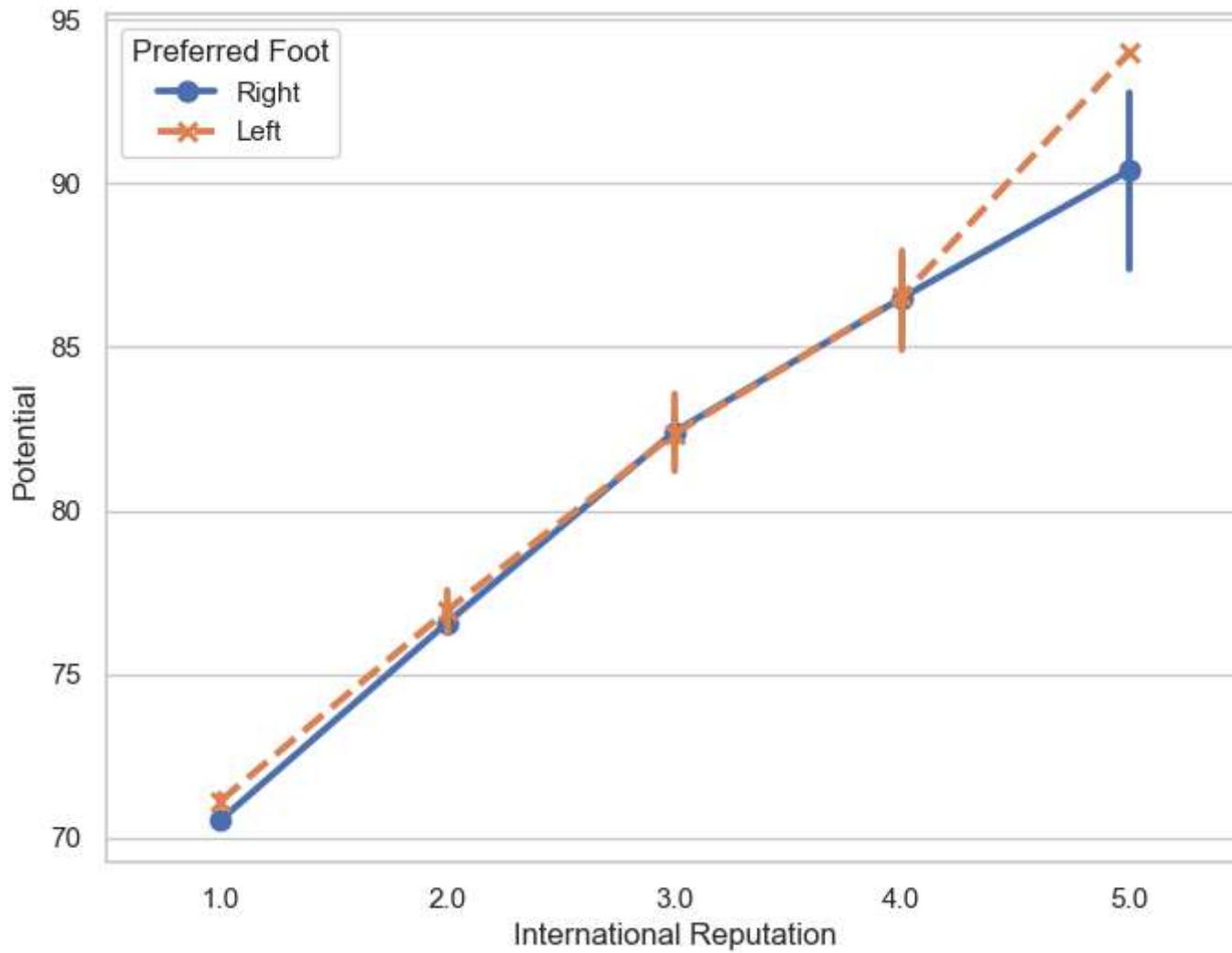
We can separate the points for different hue levels along the categorical axis as follows-

```
In [36]: f, ax = plt.subplots(figsize=(8, 6))
sns.pointplot(x="International Reputation", y="Potential", data=df, hue="Preferred Foot", dodge=True)
plt.show()
```



We can use a different marker and line style for the hue levels as follows-

```
In [37]: f, ax = plt.subplots(figsize=(8, 6))
sns.pointplot(x="International Reputation", y="Potential", hue="Preferred Foot",
               data=df, markers=["o", "x"], linestyles=[ "-", "--"])
plt.show()
```

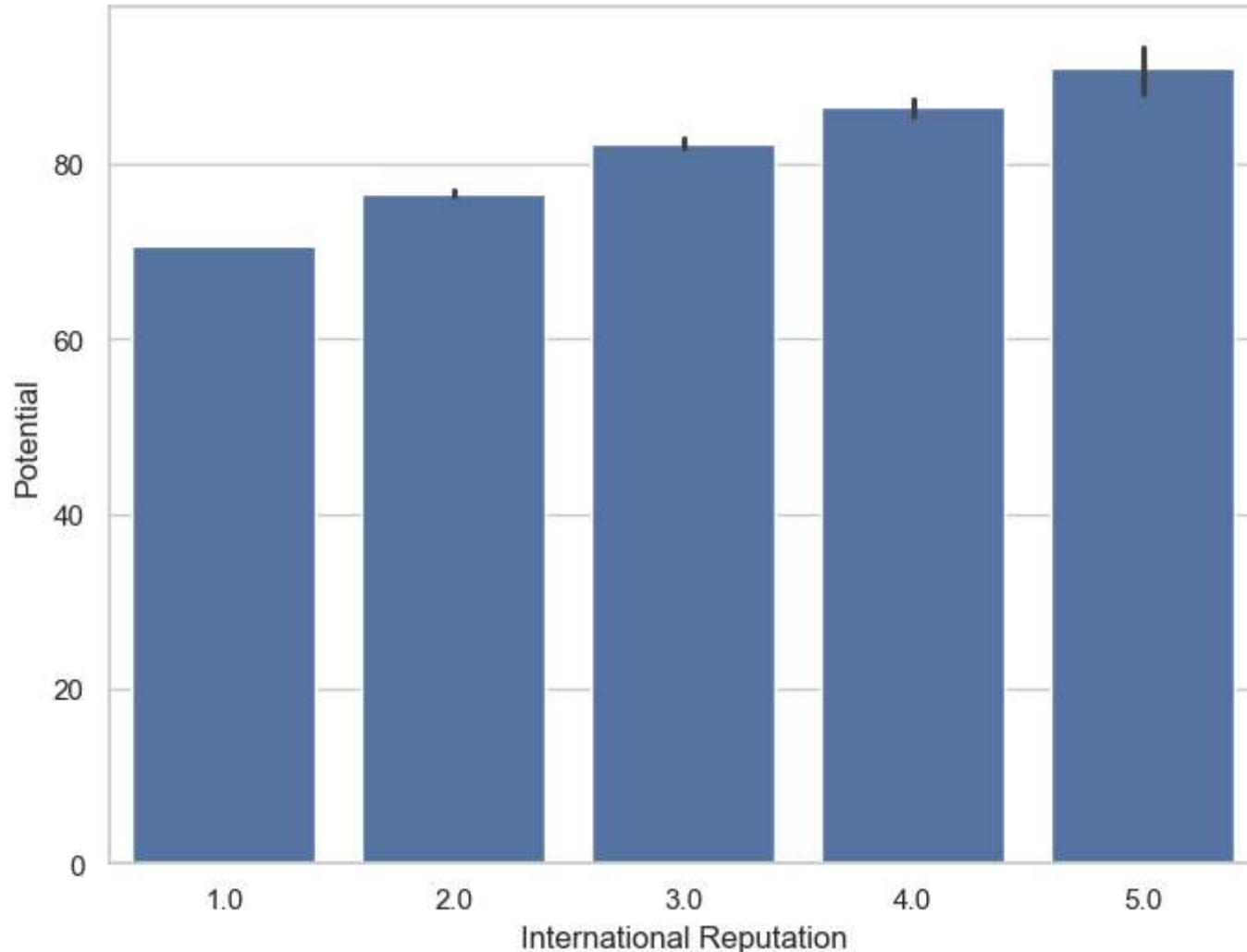


## Seaborn `barplot()` function

- 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.

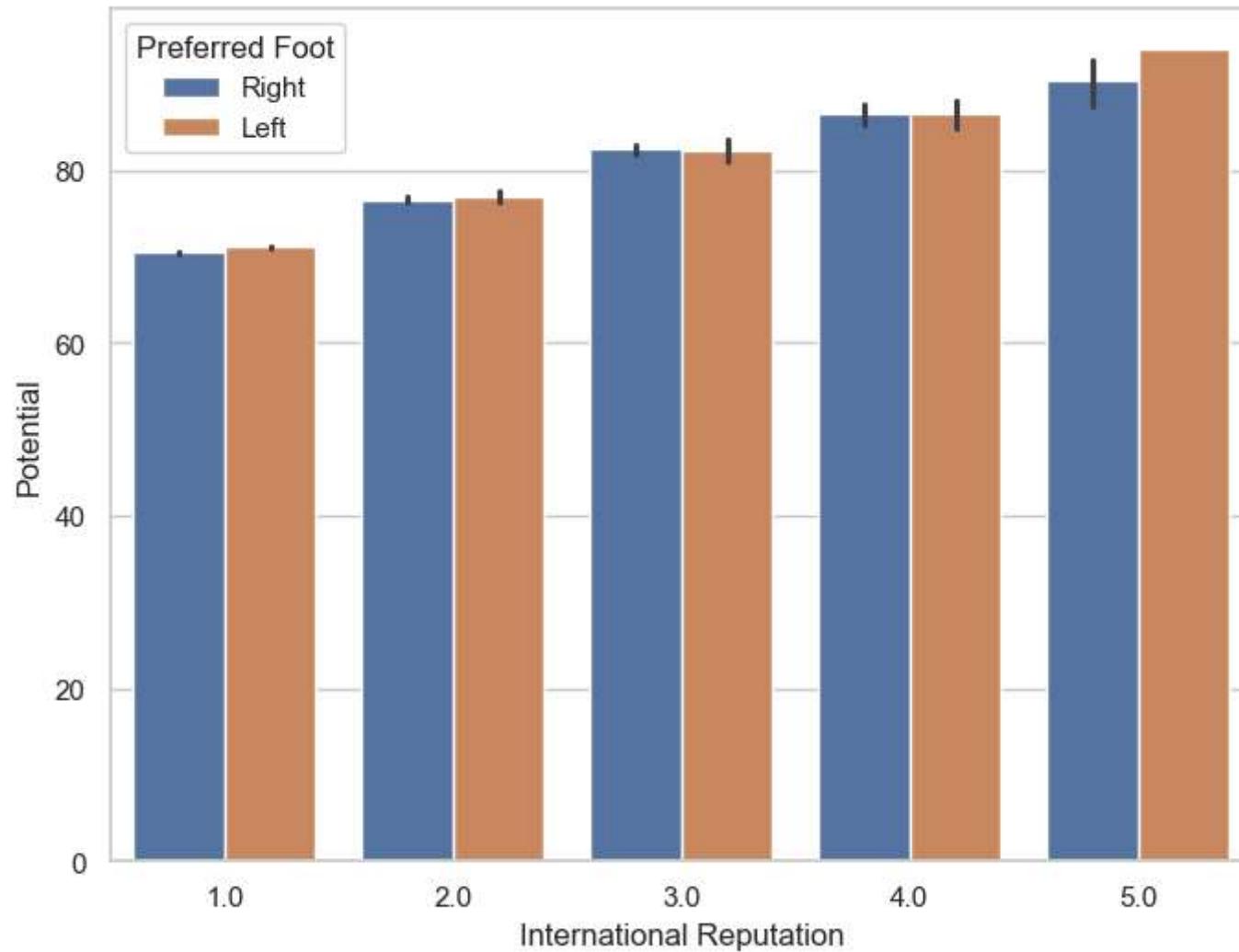
- Bar plots include 0 in the quantitative axis range, and they are a good choice when 0 is a meaningful value for the quantitative variable, and you want to make comparisons against it.

```
In [38]: f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", data=df)
plt.show()
```



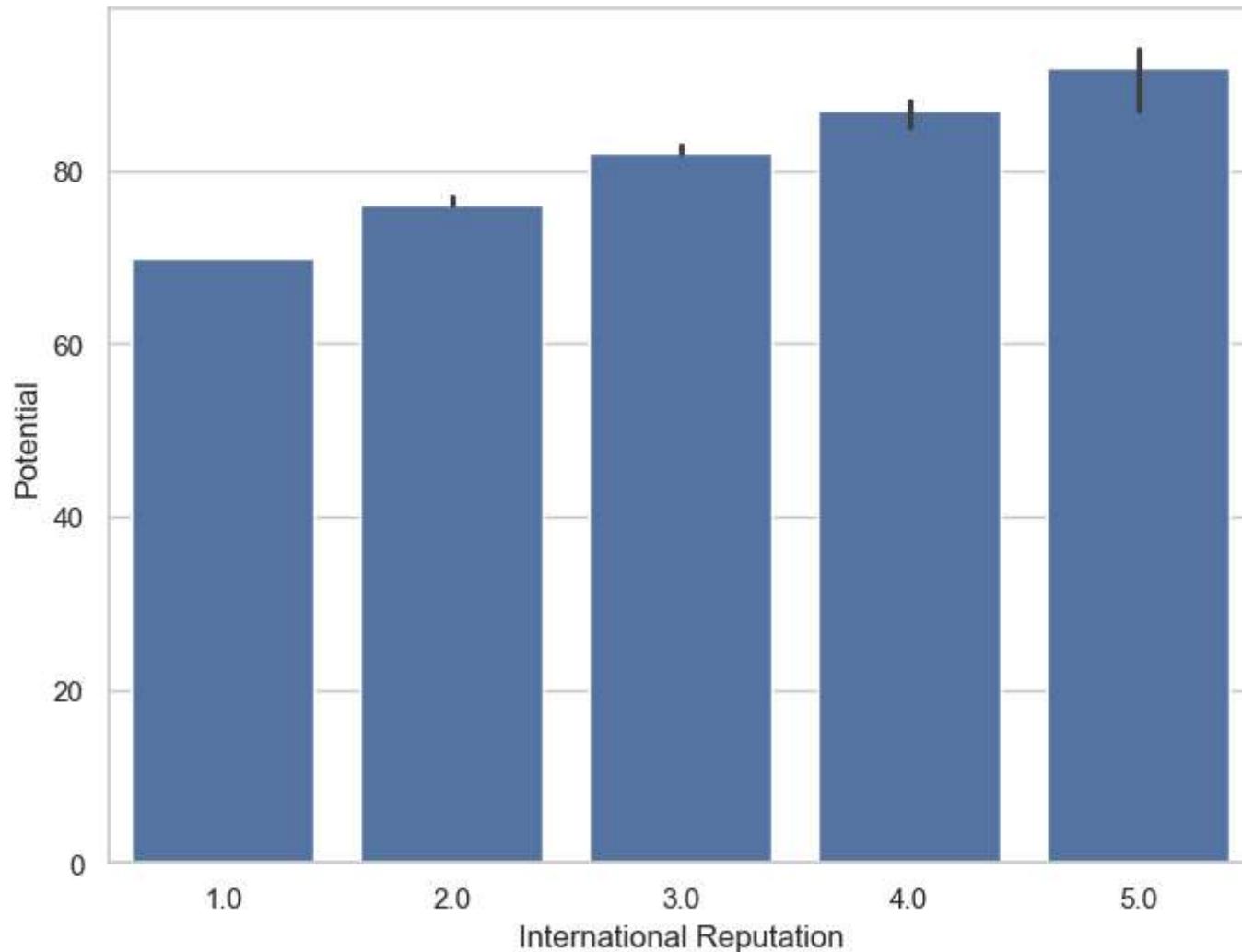
We can draw a set of vertical bars with nested grouping by a two variables as follows-

```
In [39]: f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", hue="Preferred Foot", data=df)
plt.show()
```



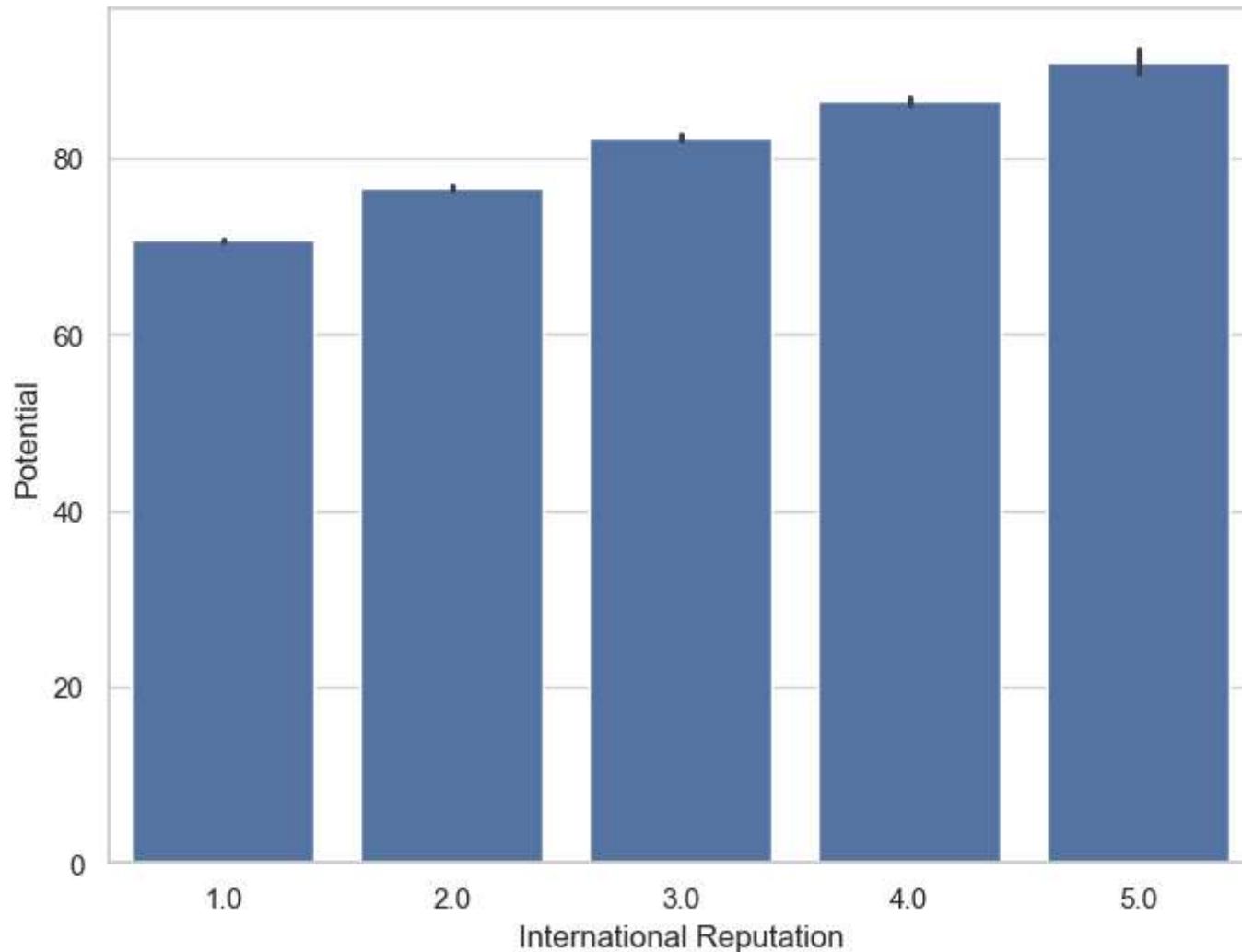
We can use median as the estimate of central tendency as follows-

```
In [40]: from numpy import median
f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", data=df, estimator=median)
plt.show()
```



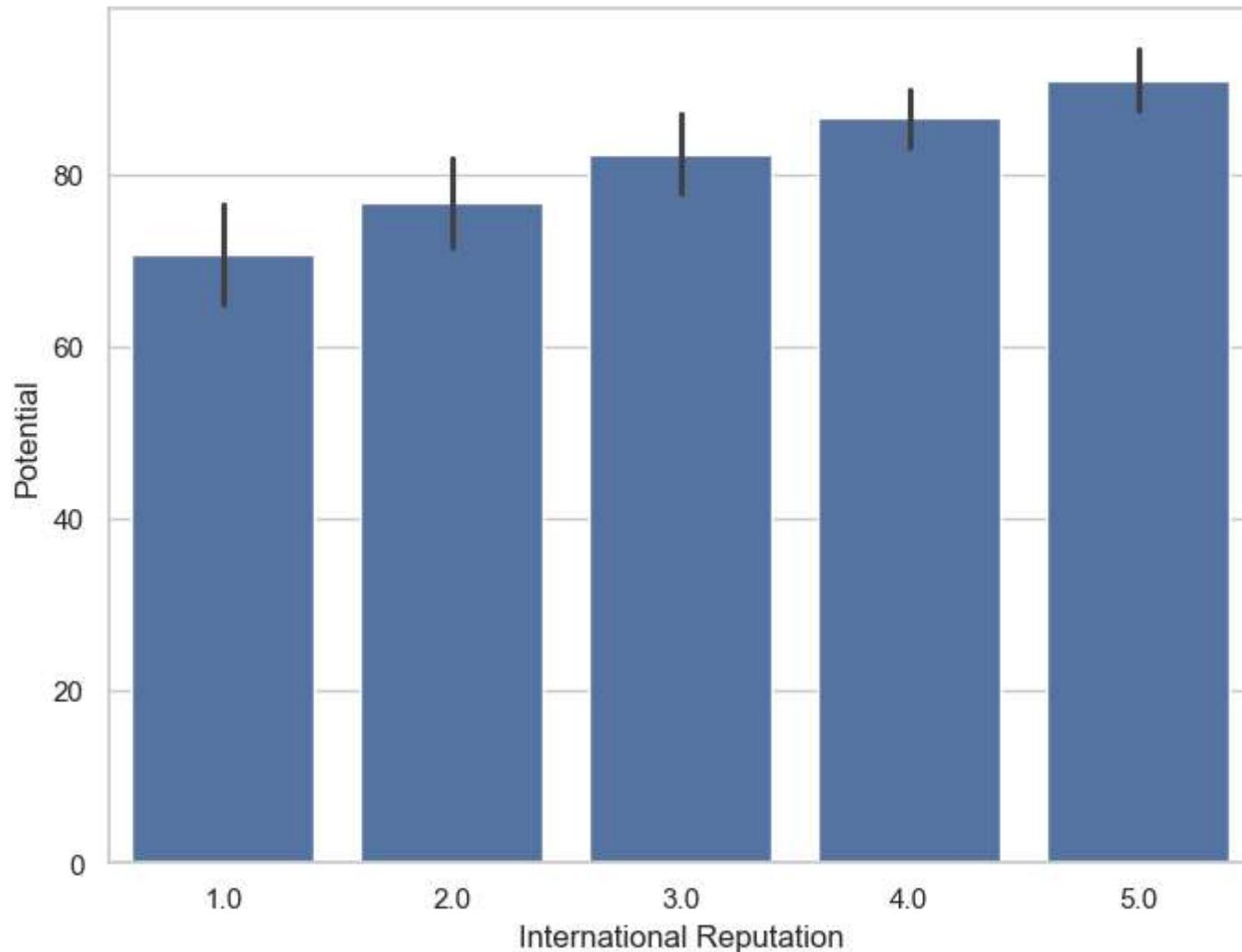
We can show the standard error of the mean with the error bars as follows

```
In [41]: f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", data=df, ci=68)
plt.show()
```



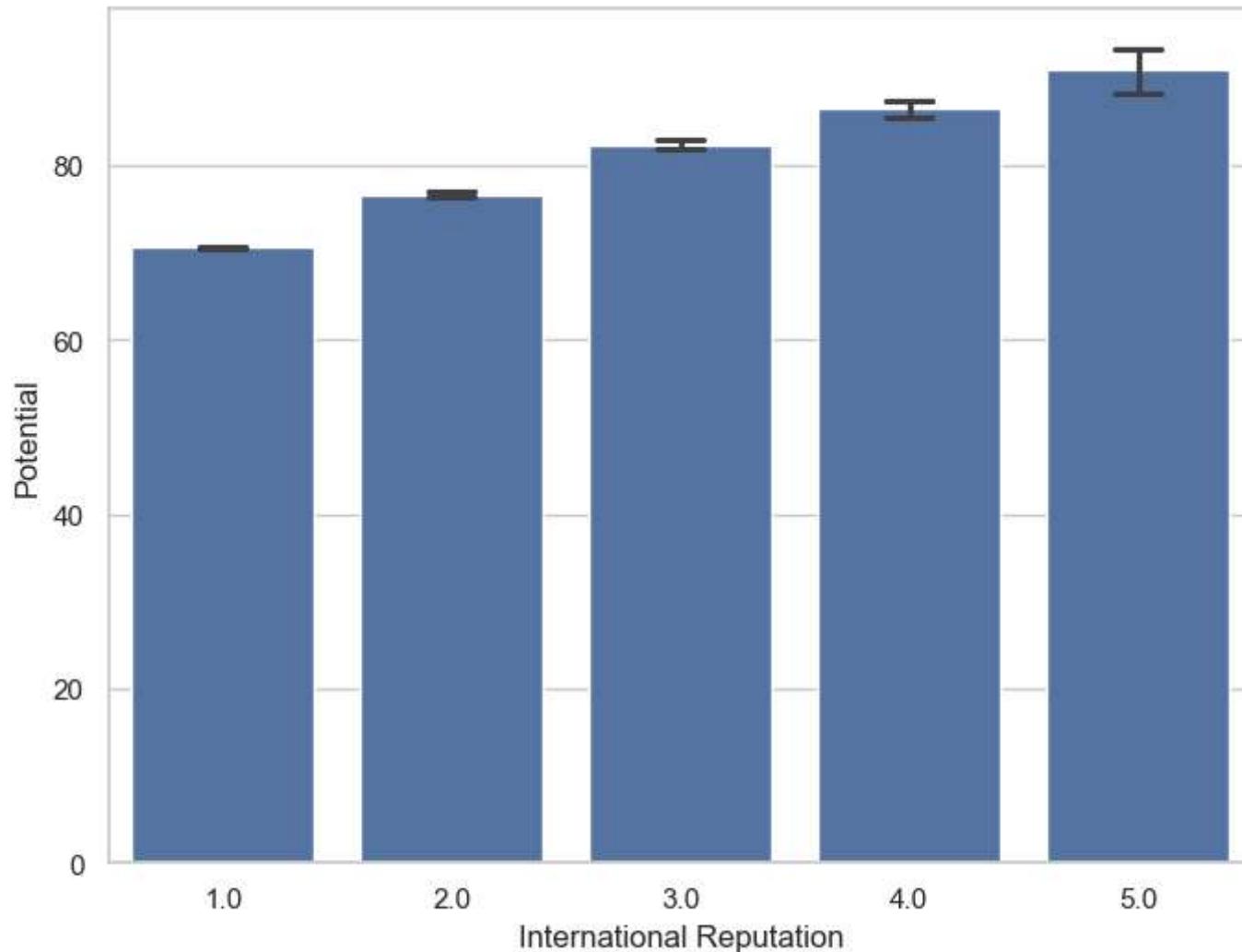
We can show standard deviation of observations instead of a confidence interval as follows-

```
In [42]: f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", data=df, ci="sd")
plt.show()
```



We can add “caps” to the error bars as follows-

```
In [43]: f, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x="International Reputation", y="Potential", data=df, capsize=0.2)
plt.show()
```



## Visualizing statistical relationship with Seaborn `regplot()` function

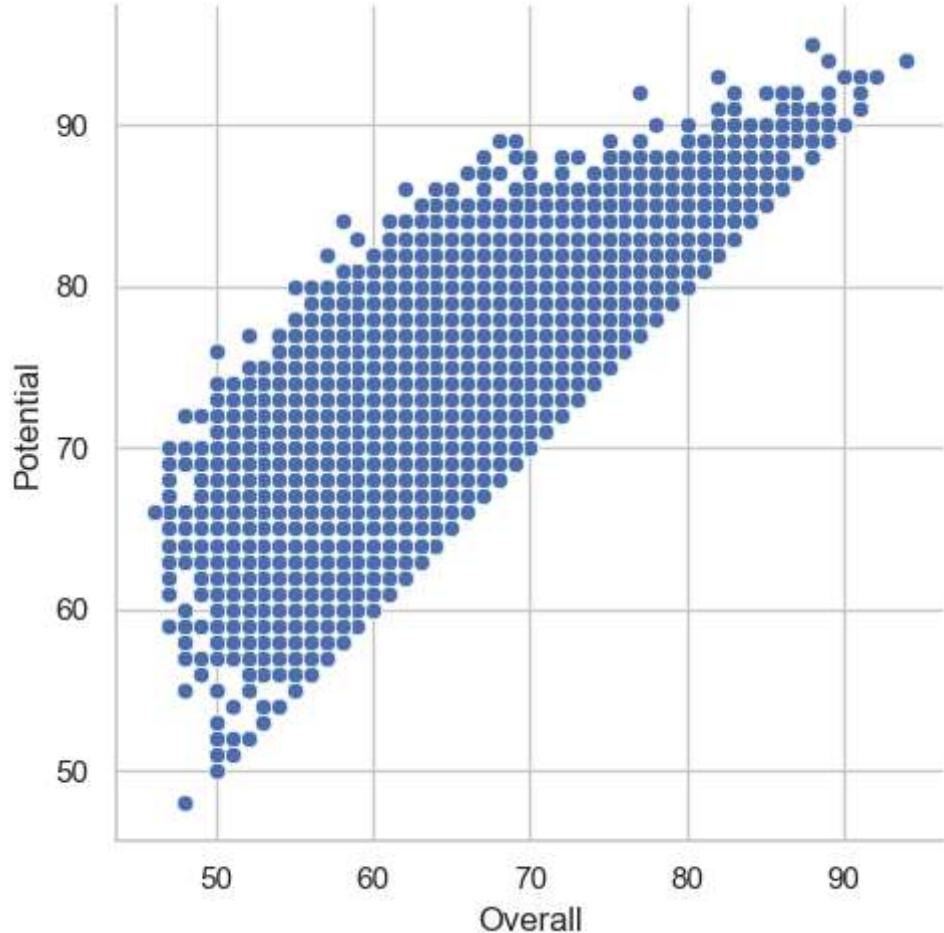
### Seaborn `relplot()` function

- 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 scatterplot with variables `height` and `Weight` with seaborn `regplot()` function

```
In [44]: g = sns.relplot(x="Overall", y="Potential", data=df)
```

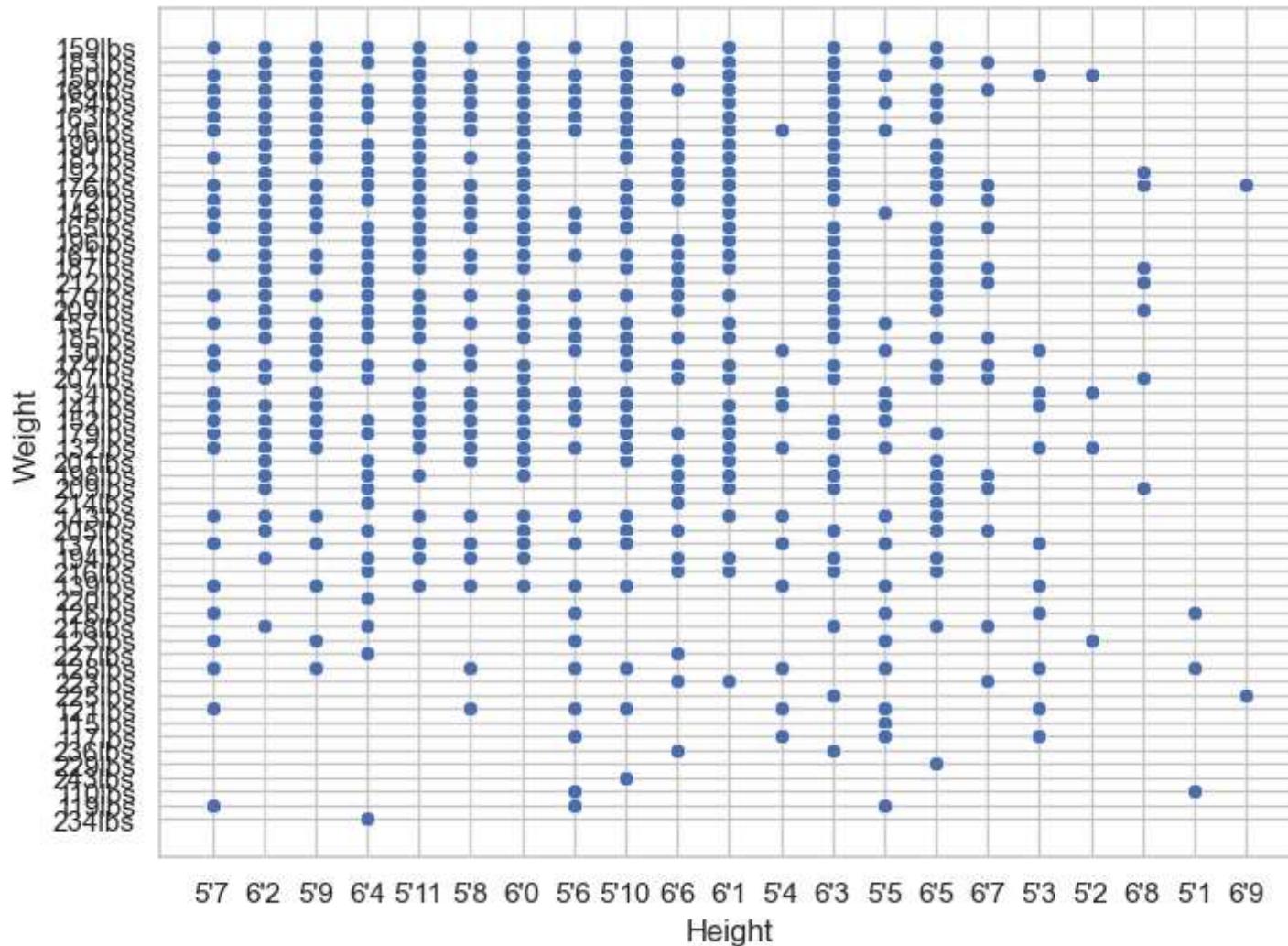


## Seaborn `scatterplot()` function

- 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 [45]:

```
f, ax = plt.subplots(figsize=(8, 6))
sns.scatterplot(x="Height", y="Weight", data=df)
plt.show()
```

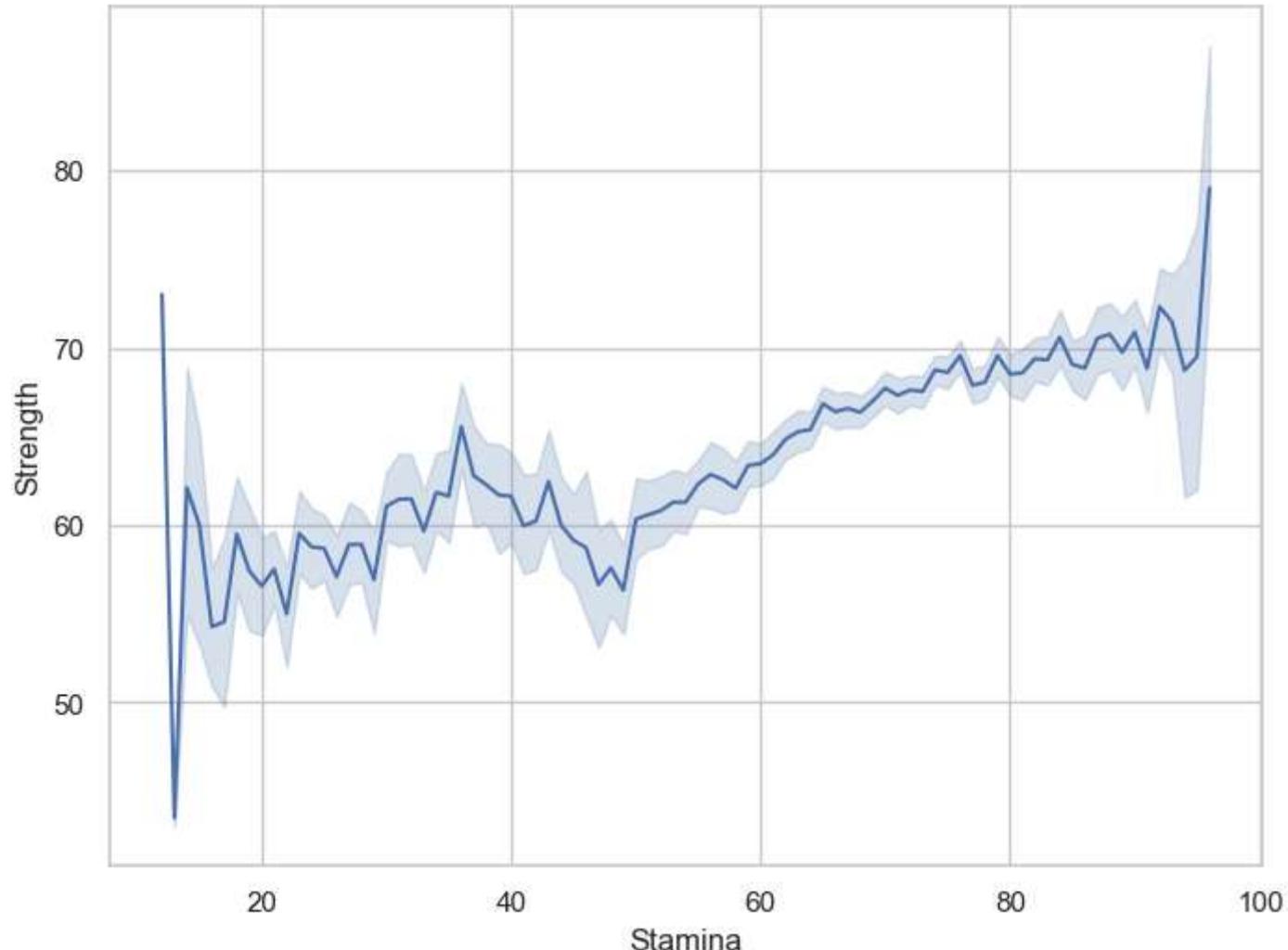


## Seaborn lineplot() function

- 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 [46]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.lineplot(x="Stamina", y="Strength", data=df)
plt.show()
```

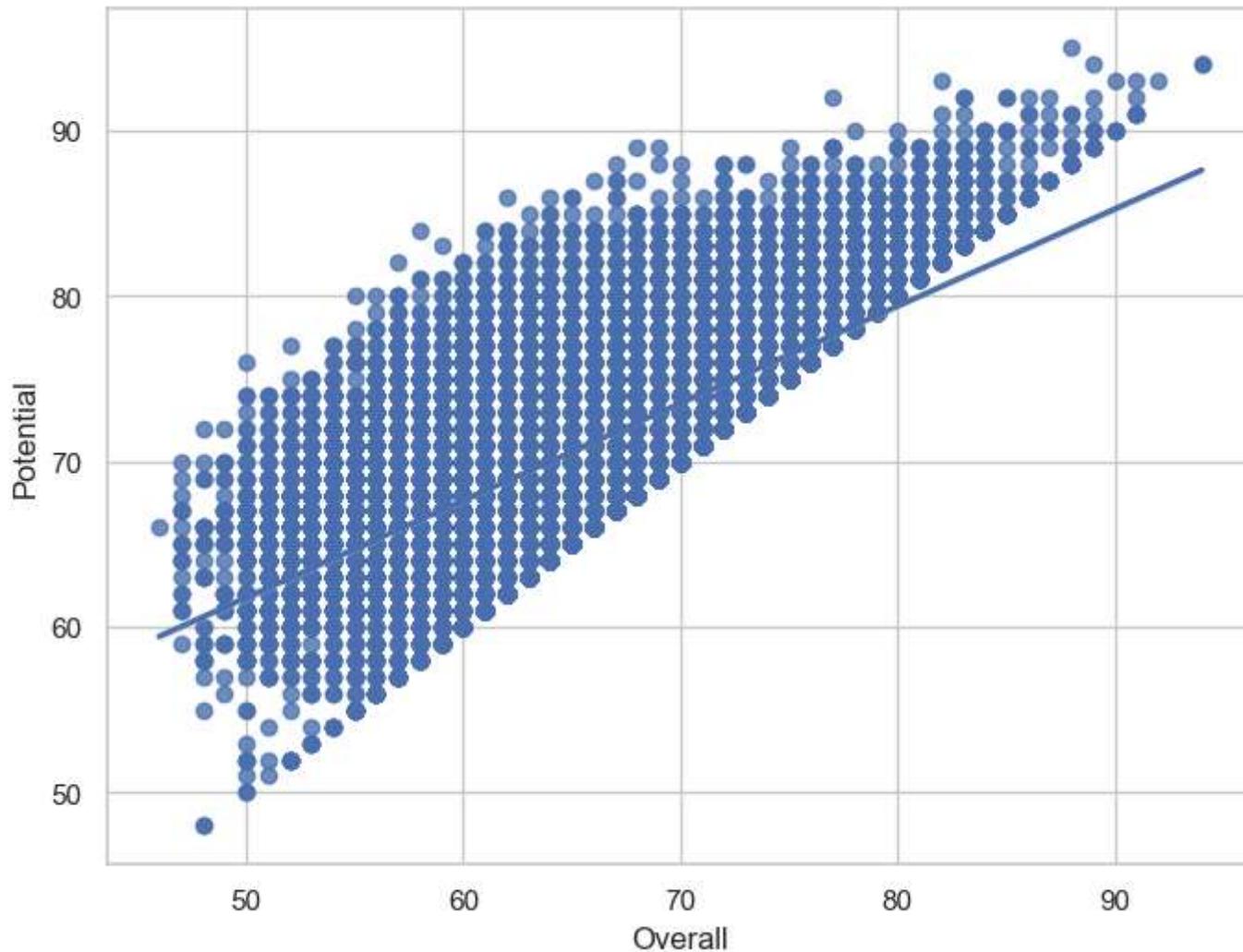


# Visualize linear relationship with Seaborn `regplot()` function

## Seaborn `regplot()` function

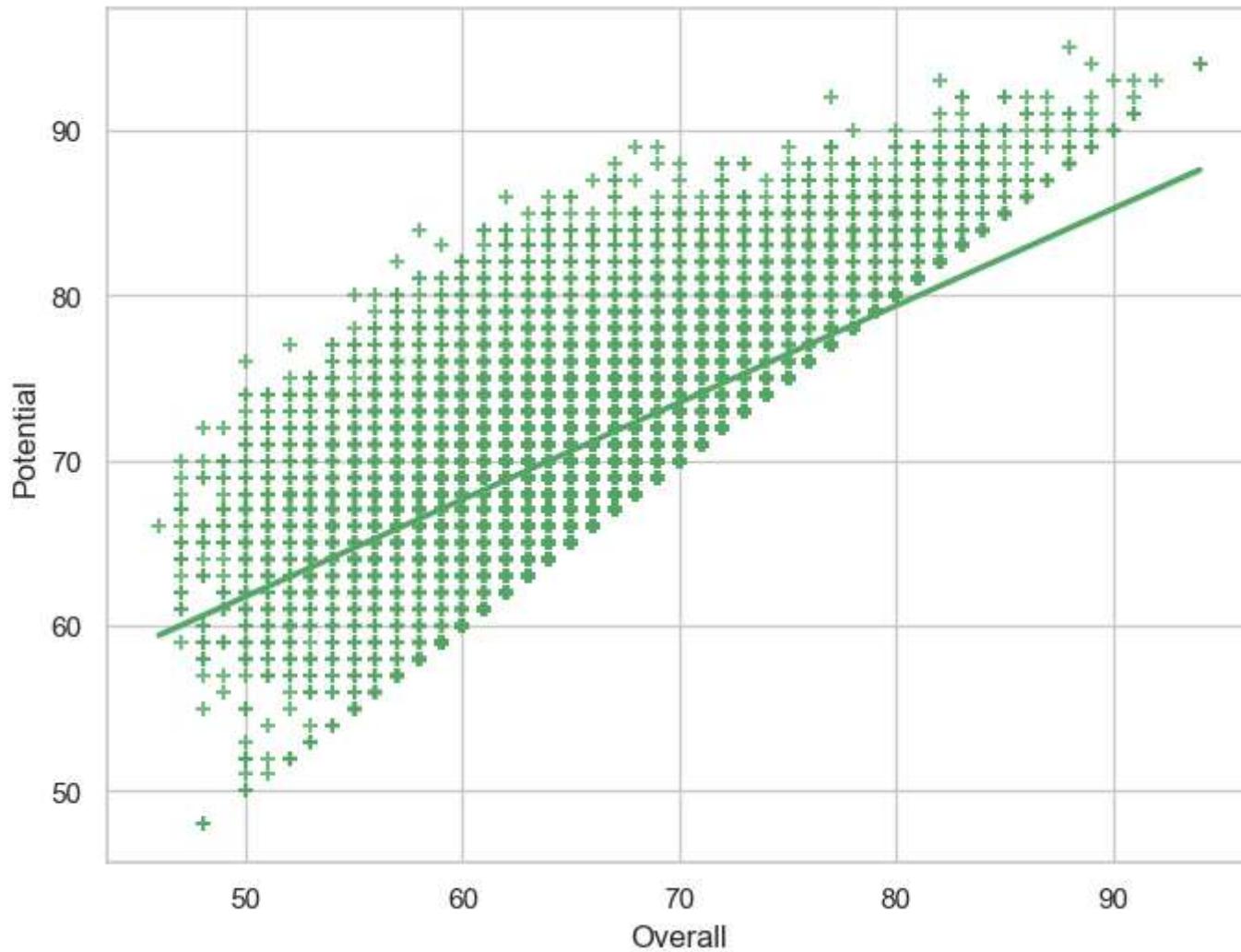
- This function plots data and a linear regression model fit.
- We can plot a linear regression model between `Overall` and `Potential` variable with `regplot()` function as follows-

```
In [47]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=df)
plt.show()
```



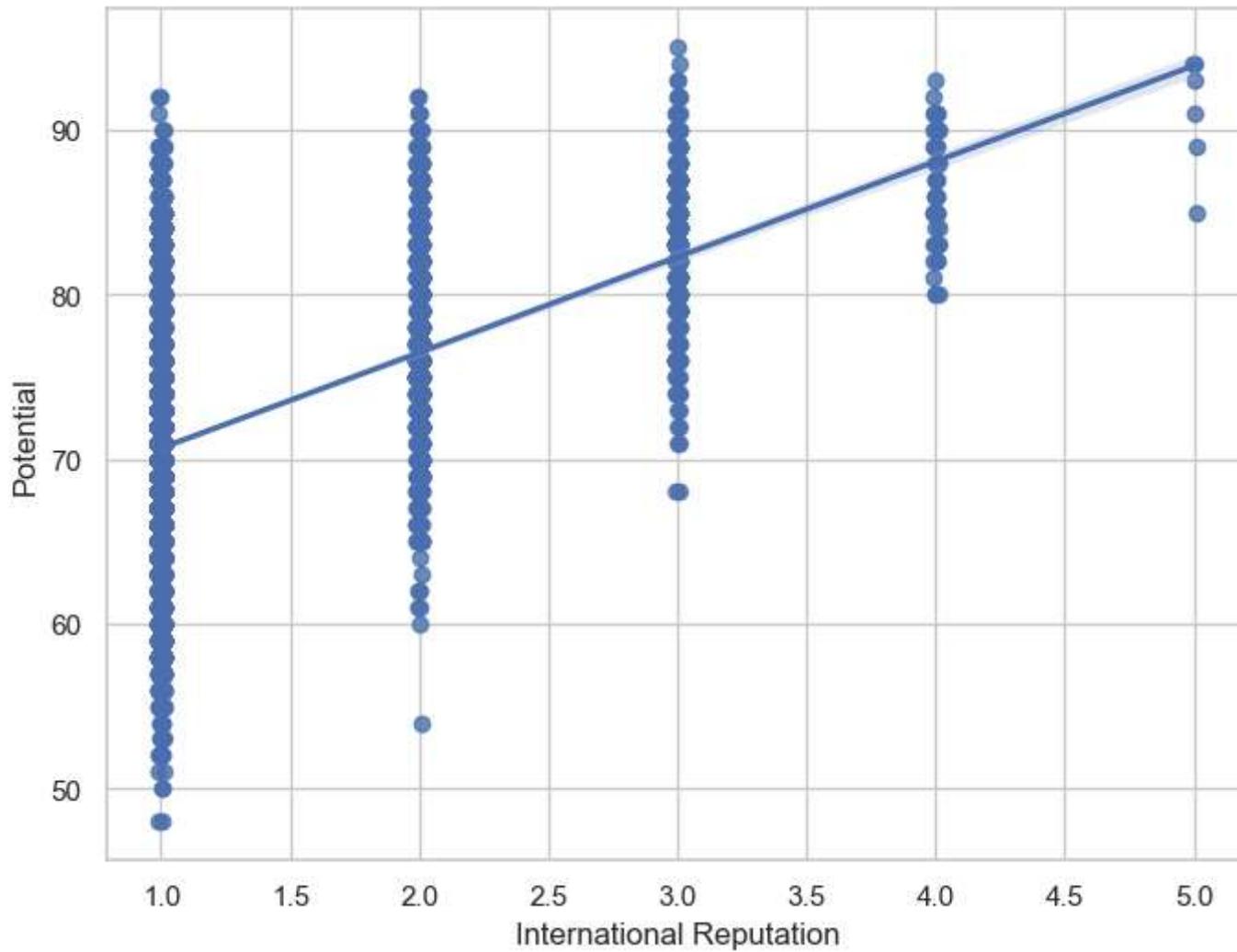
We can use a different color and marker as follows-

```
In [48]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=df, color= "g", marker="+")
plt.show()
```



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

```
In [49]: f, ax = plt.subplots(figsize=(8, 6))
sns.regplot(x="International Reputation", y="Potential", data=df, x_jitter=.01)
plt.show()
```

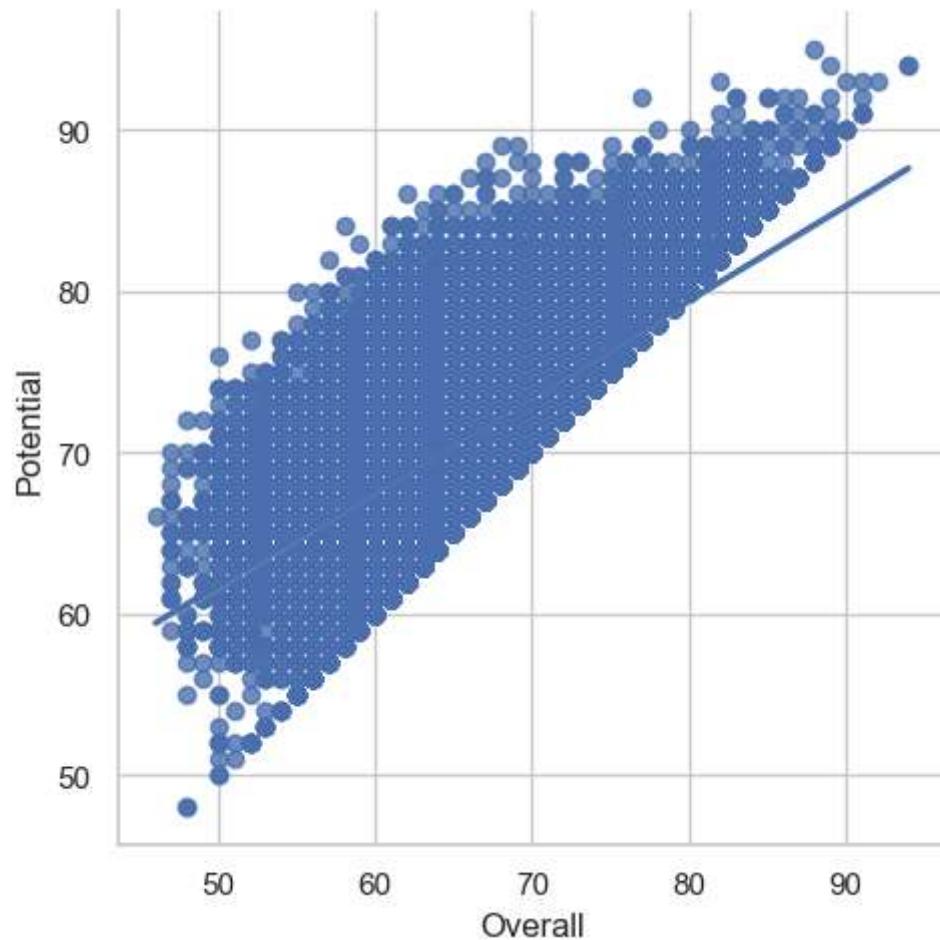


## Seaborn `lmplot()` function

- 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.

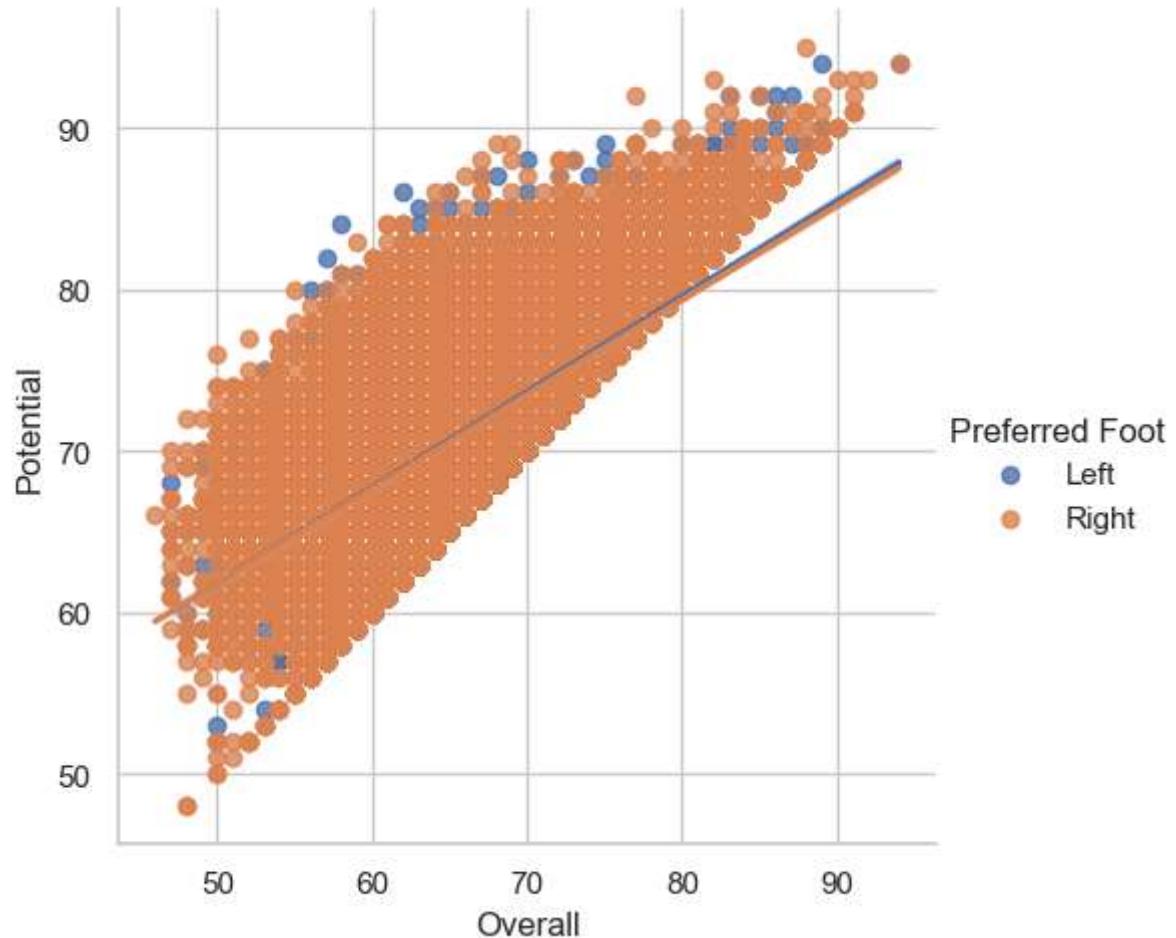
- We can plot a linear regression model between Overall and Potential variable with lmplot() function as follows-

```
In [50]: g = sns.lmplot(x="Overall", y="Potential", data=df)
```



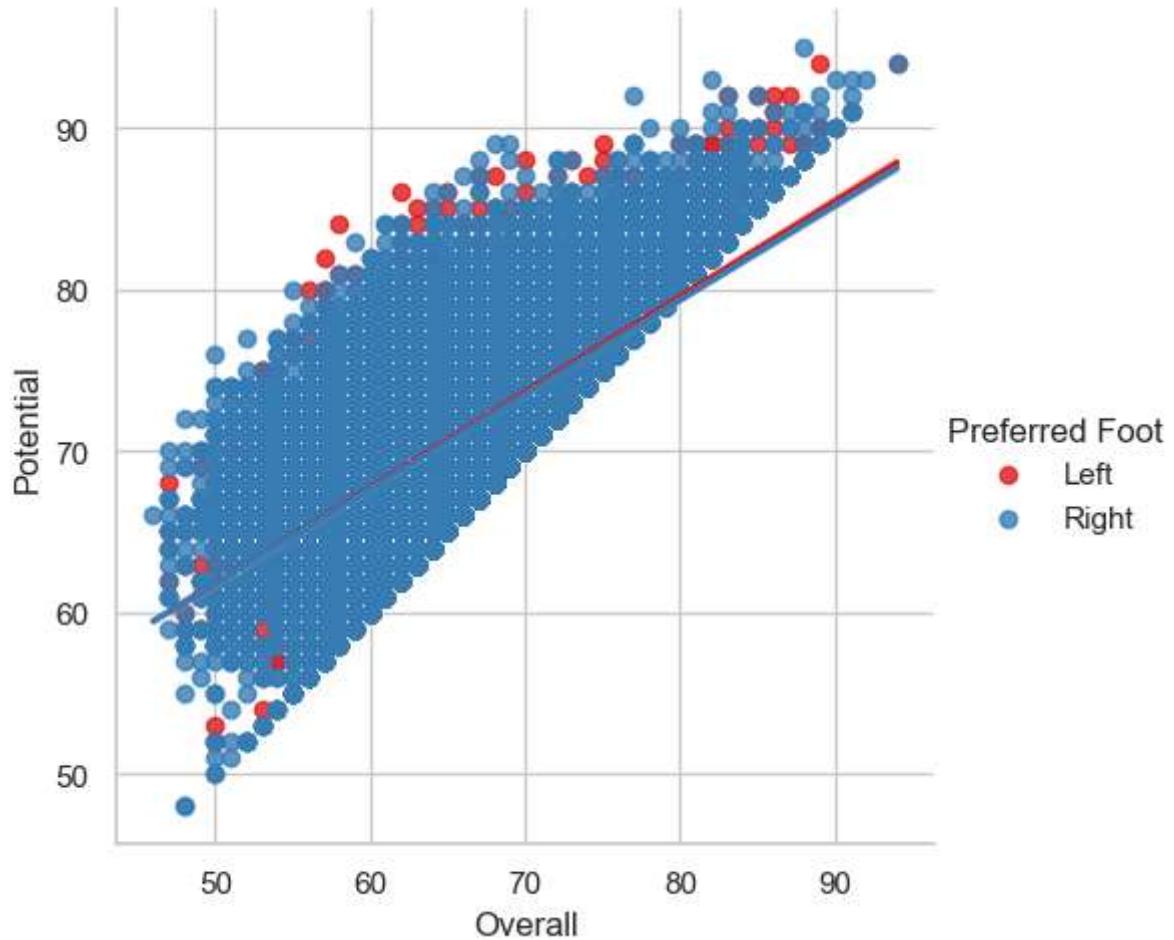
We can condition on a third variable and plot the levels in different colors as follows-

```
In [51]: g = sns.lmplot(x="Overall", y="Potential", hue="Preferred Foot", data=df)
```



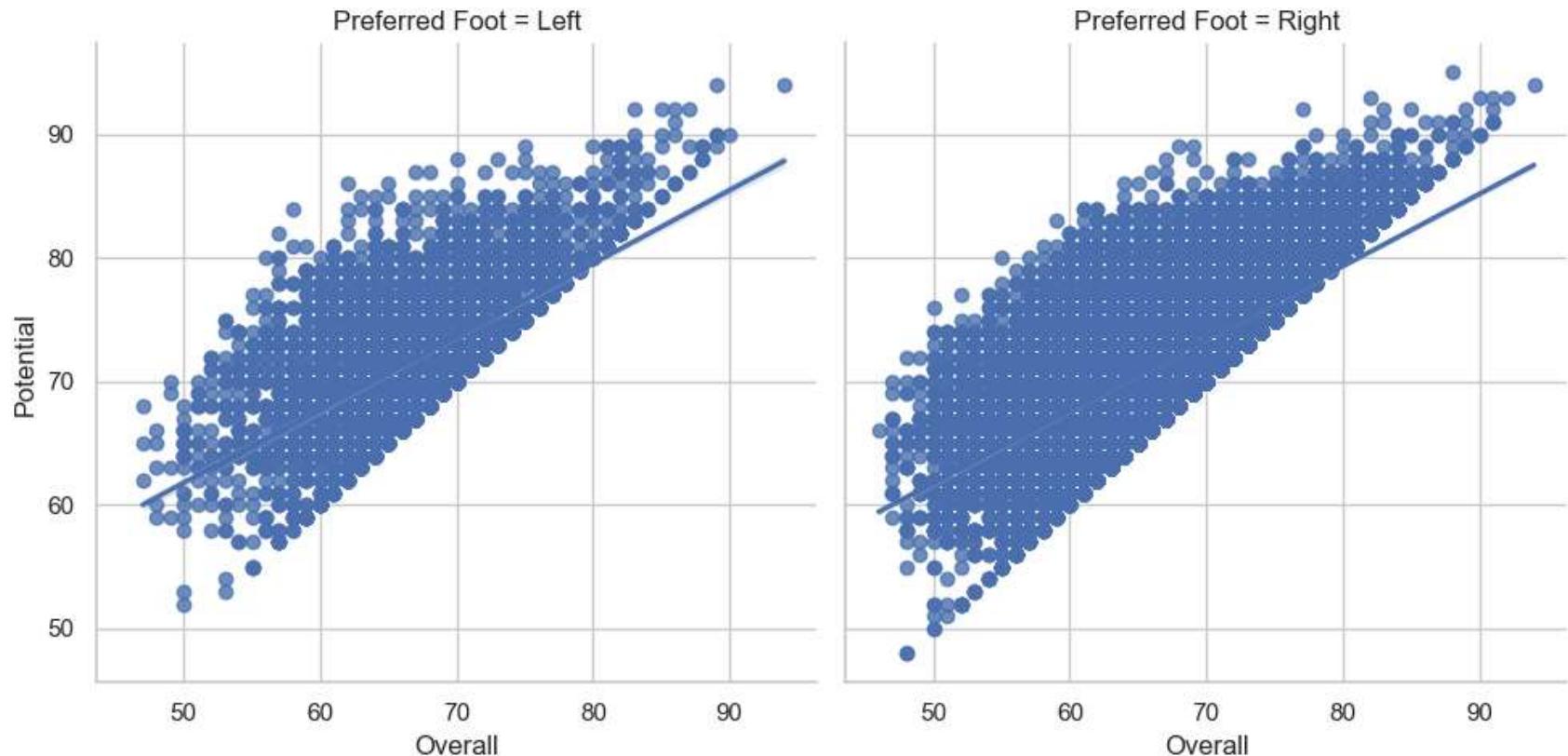
We can use a different color palette as follows-

```
In [52]: g = sns.lmplot(x="Overall", y="Potential", hue="Preferred Foot", data=df, palette="Set1")
```



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

```
In [53]: g = sns.lmplot(x="Overall", y="Potential", col="Preferred Foot", data=df)
```



## Multi-plot grids

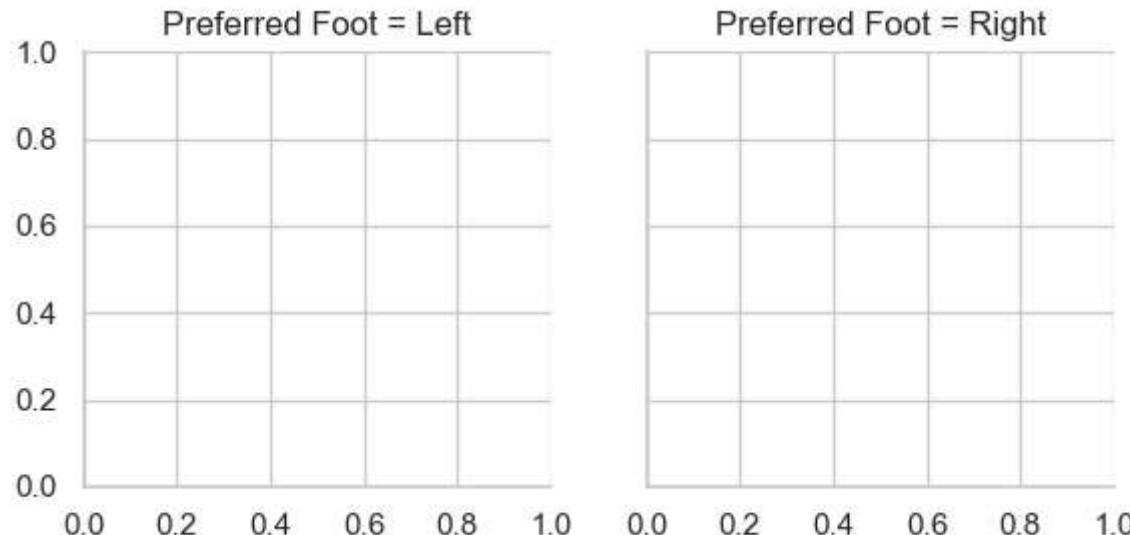
### Seaborn `FacetGrid()` function

- 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.

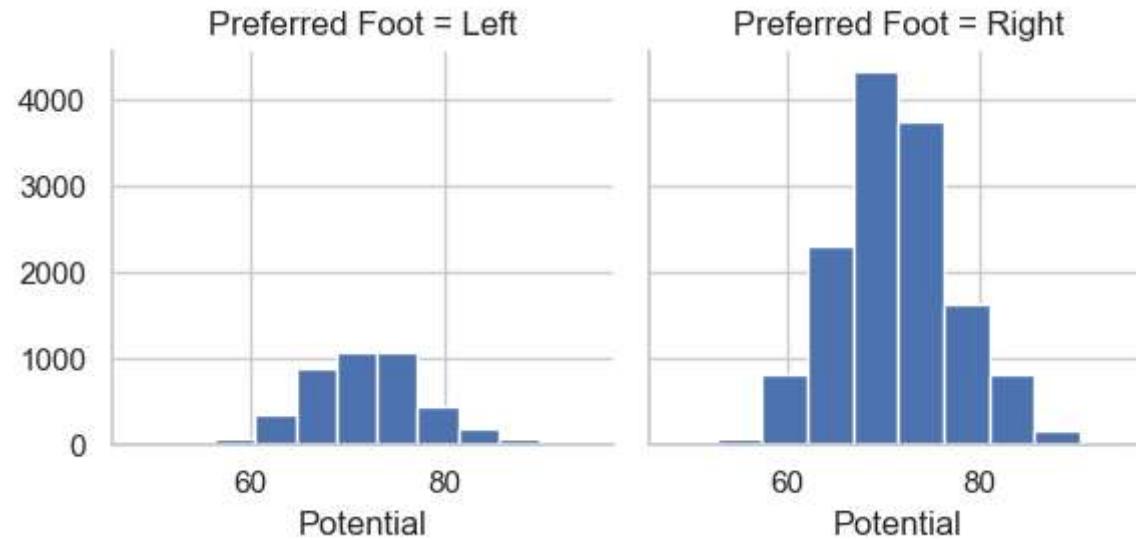
We can initialize a 1x2 grid of facets using the fifa19 dataset.

```
In [54]: g = sns.FacetGrid(df, col="Preferred Foot")
```

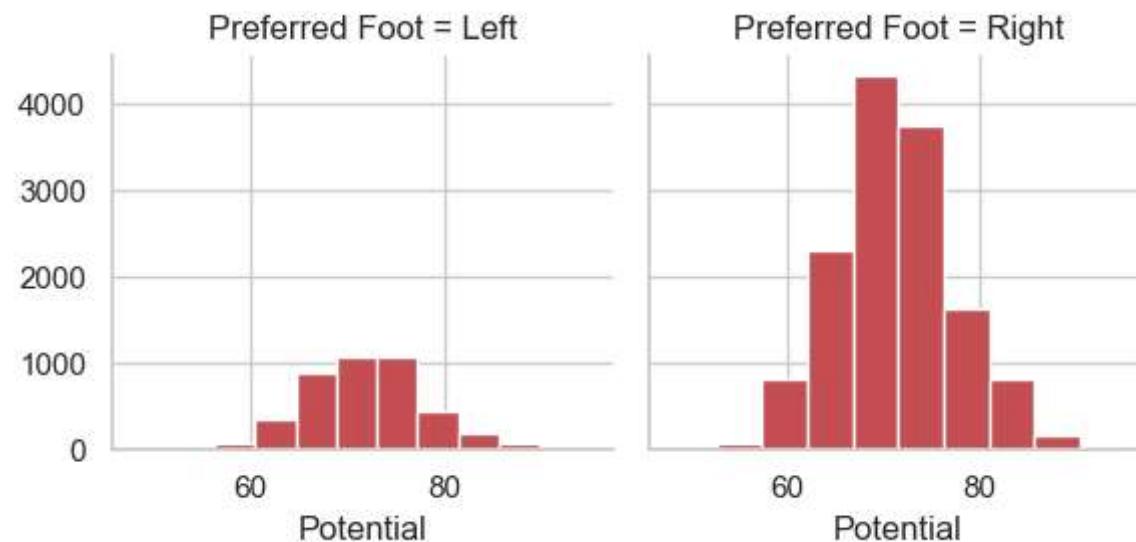


We can draw a univariate plot of `Potential` variable on each facet as follows-

```
In [55]: g = sns.FacetGrid(df, col="Preferred Foot")
g = g.map(plt.hist, "Potential")
```

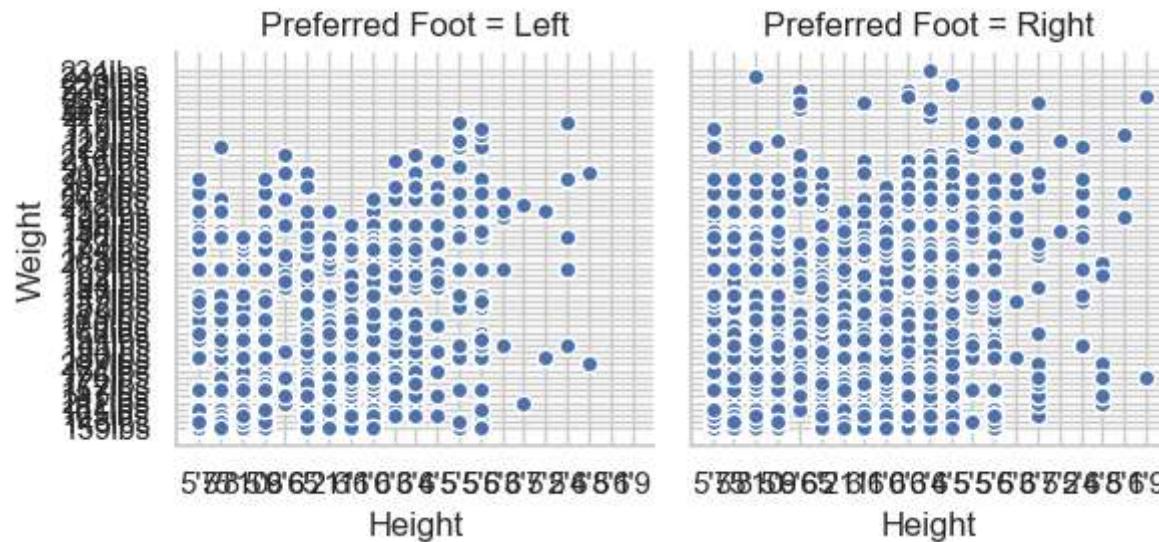


```
In [56]: g = sns.FacetGrid(df, col="Preferred Foot")
g = g.map(plt.hist, "Potential", bins=10, color="r")
```



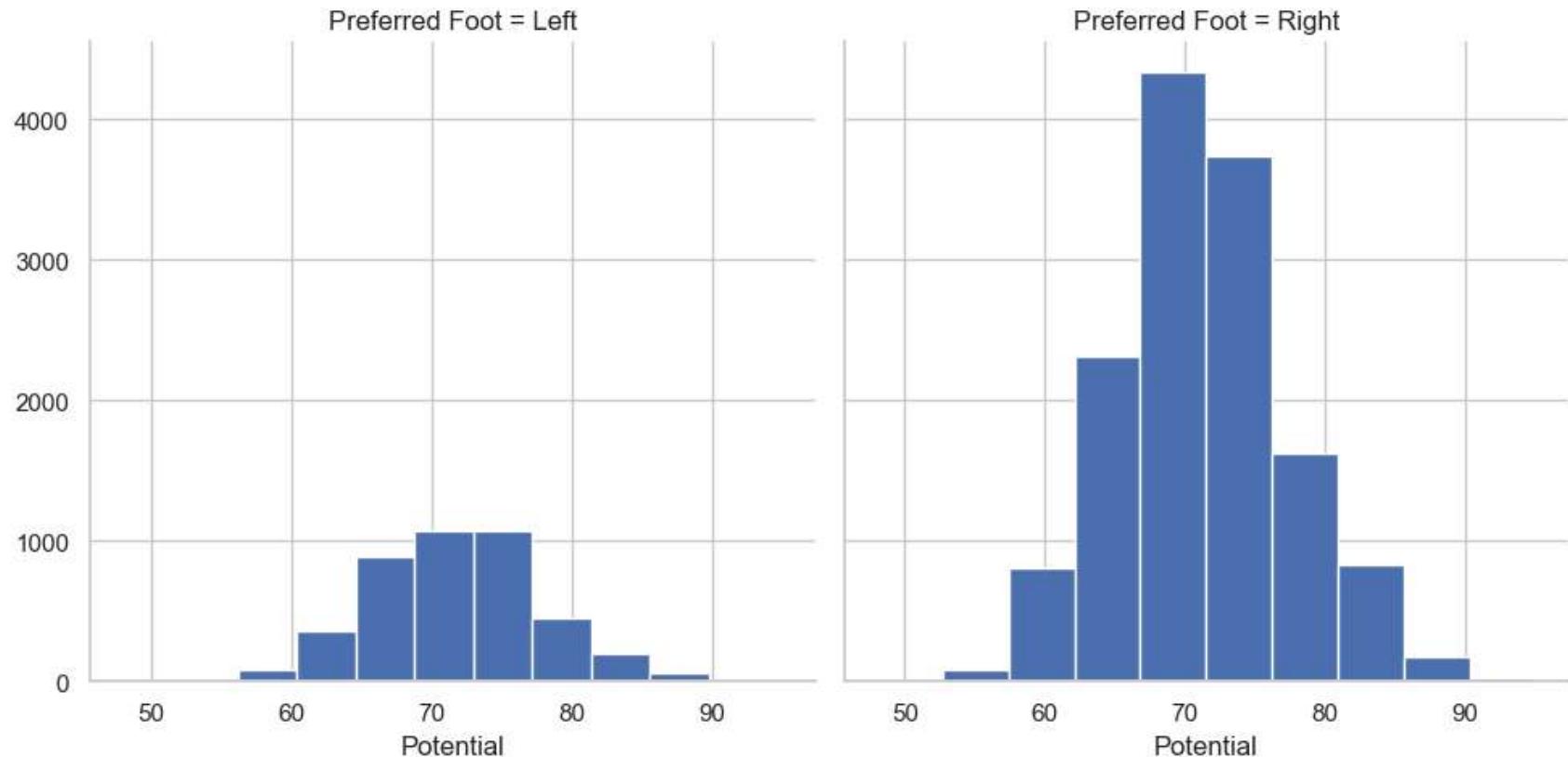
We can plot a bivariate function on each facet as follows-

```
In [57]: g = sns.FacetGrid(df, 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 [58]: g = sns.FacetGrid(df, col="Preferred Foot", height=5, aspect=1)
g = g.map(plt.hist, "Potential")
```



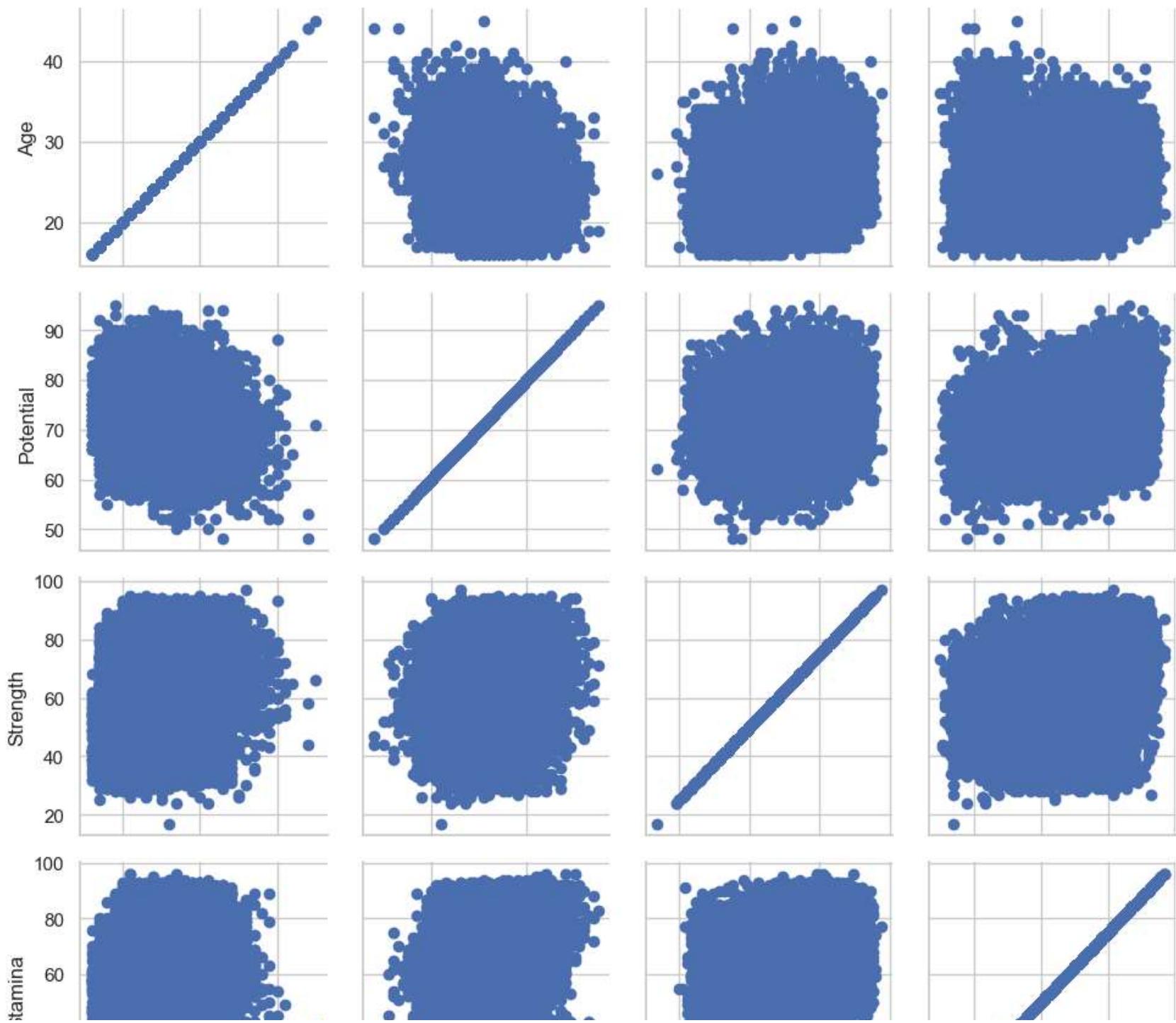
## Seaborn `Pairgrid()` function

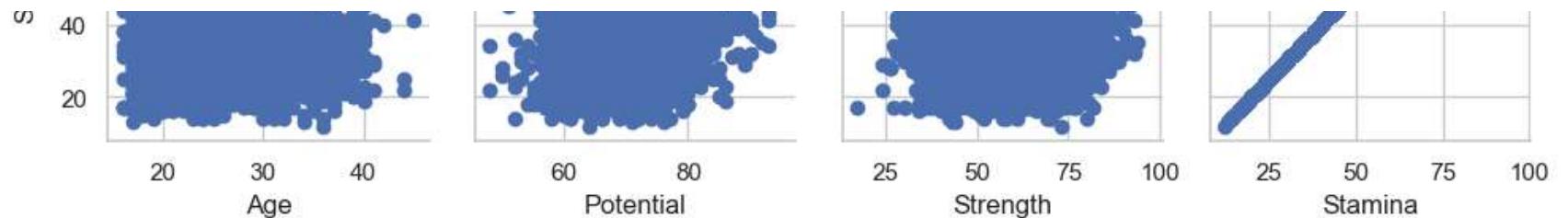
- This function plots subplot grid for plotting pairwise relationships in a dataset.
- This class maps each variable in a dataset onto a column and row in a grid of multiple axes.
- Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the marginal 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 [59]: fifa19_new = df[['Age', 'Potential', 'Strength', 'Stamina', 'Preferred Foot']]
```

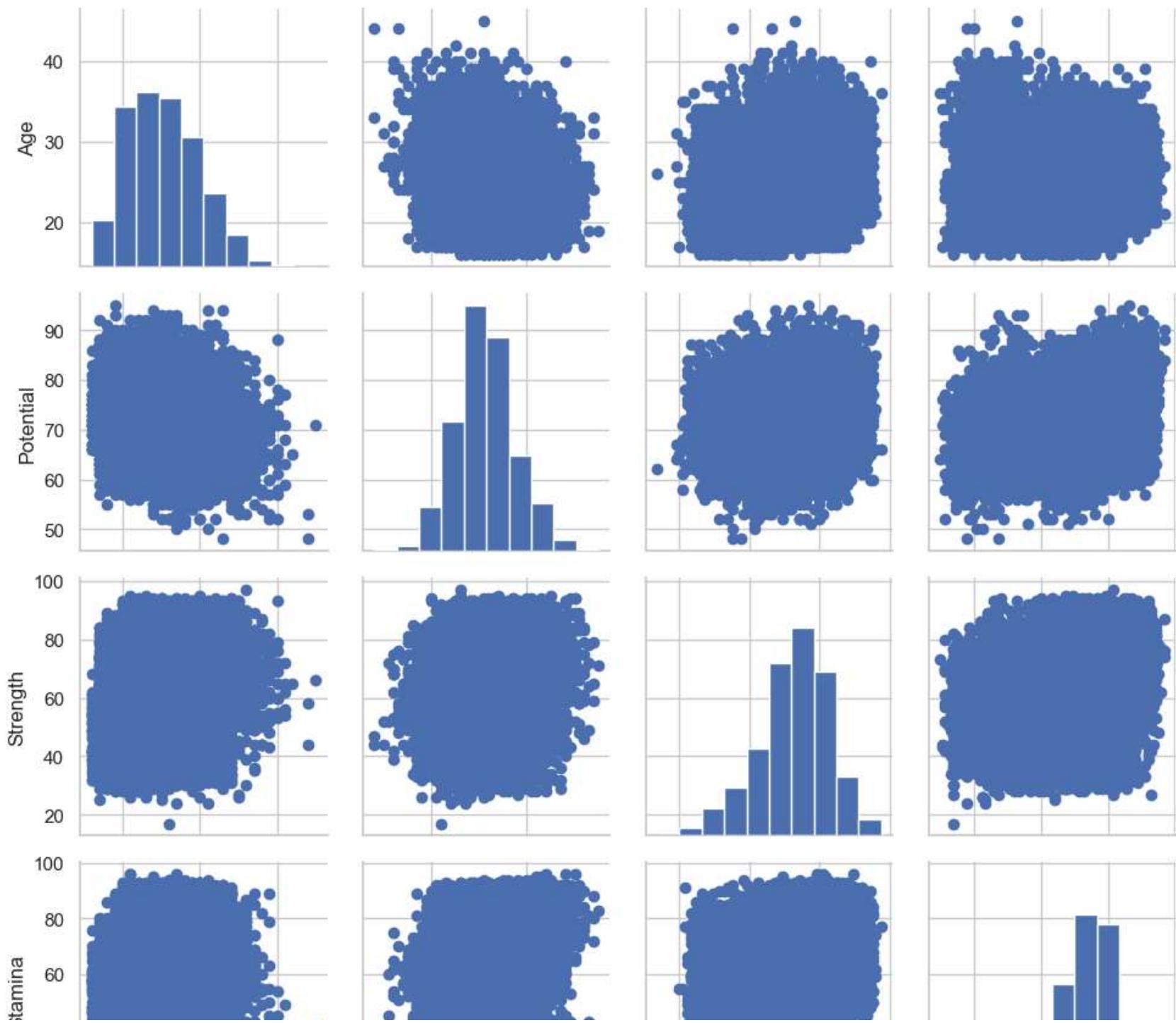
```
In [60]: g = sns.PairGrid(fifa19_new)
g = g.map(plt.scatter)
```

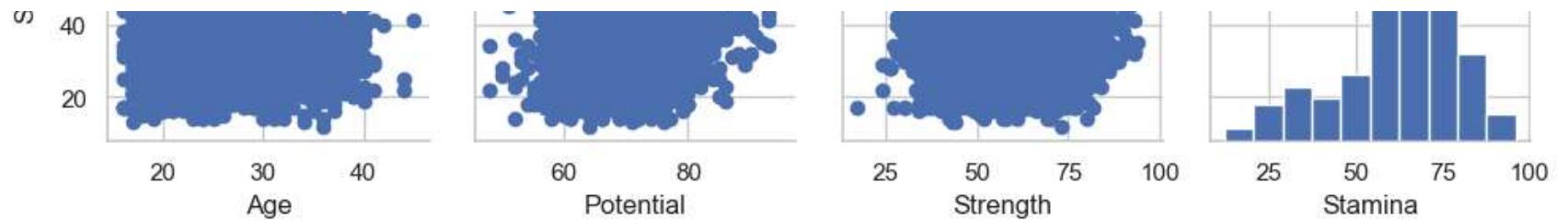




We can show a univariate distribution on the diagonal as follows-

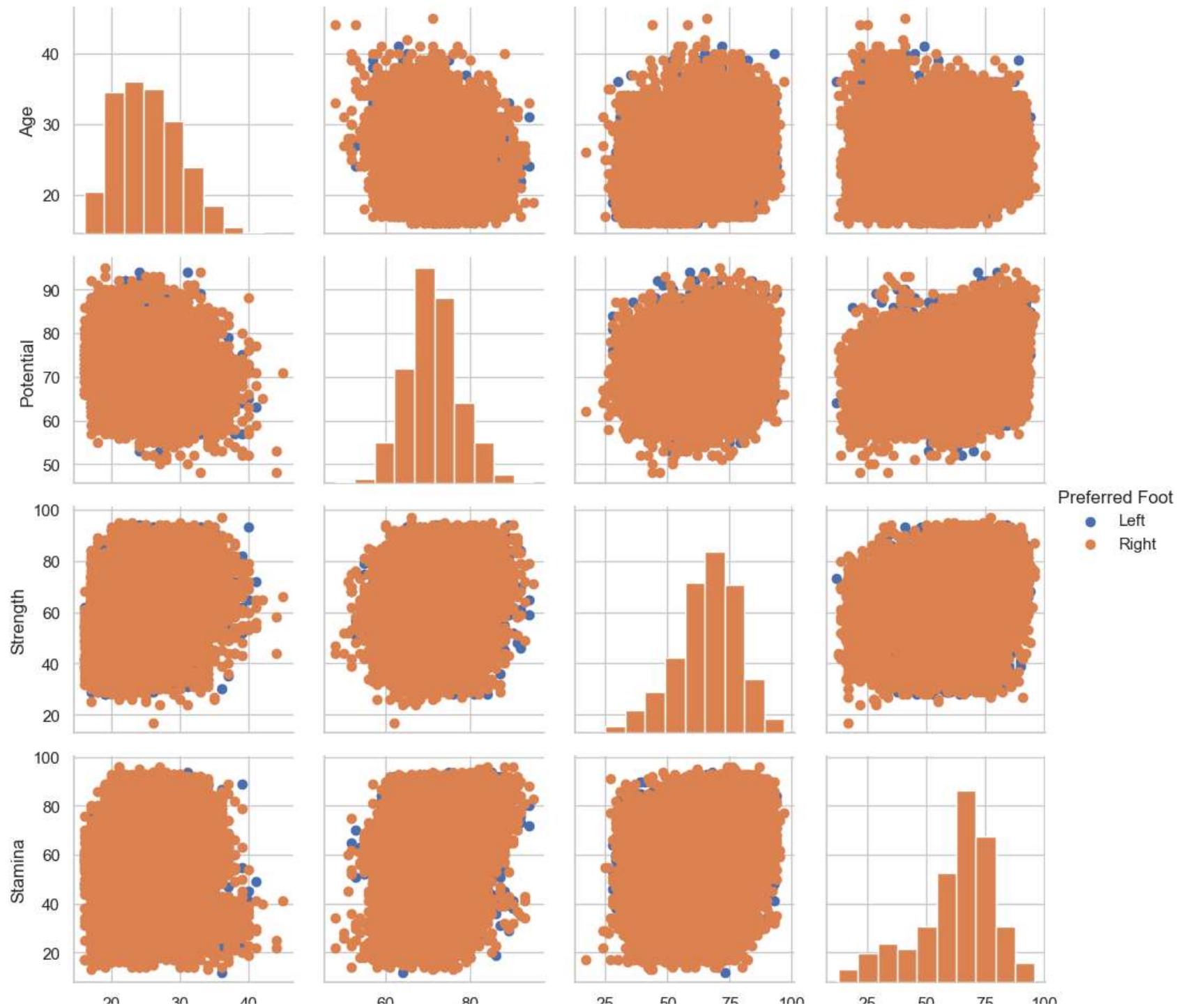
```
In [61]: g = sns.PairGrid(fifa19_new)
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
```





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

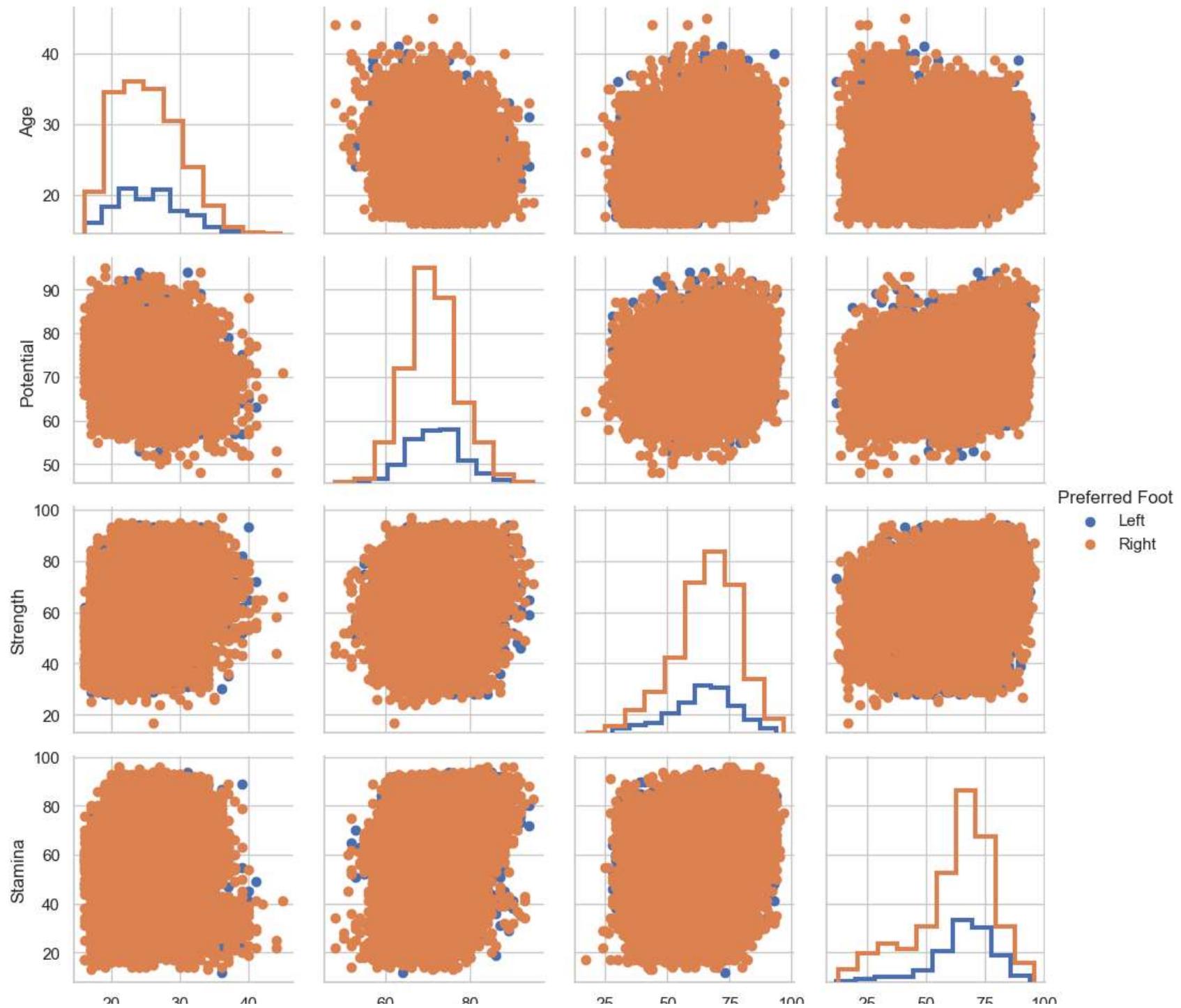
```
In [62]: g = sns.PairGrid(fifa19_new, hue="Preferred Foot")
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
g = g.add_legend()
```





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

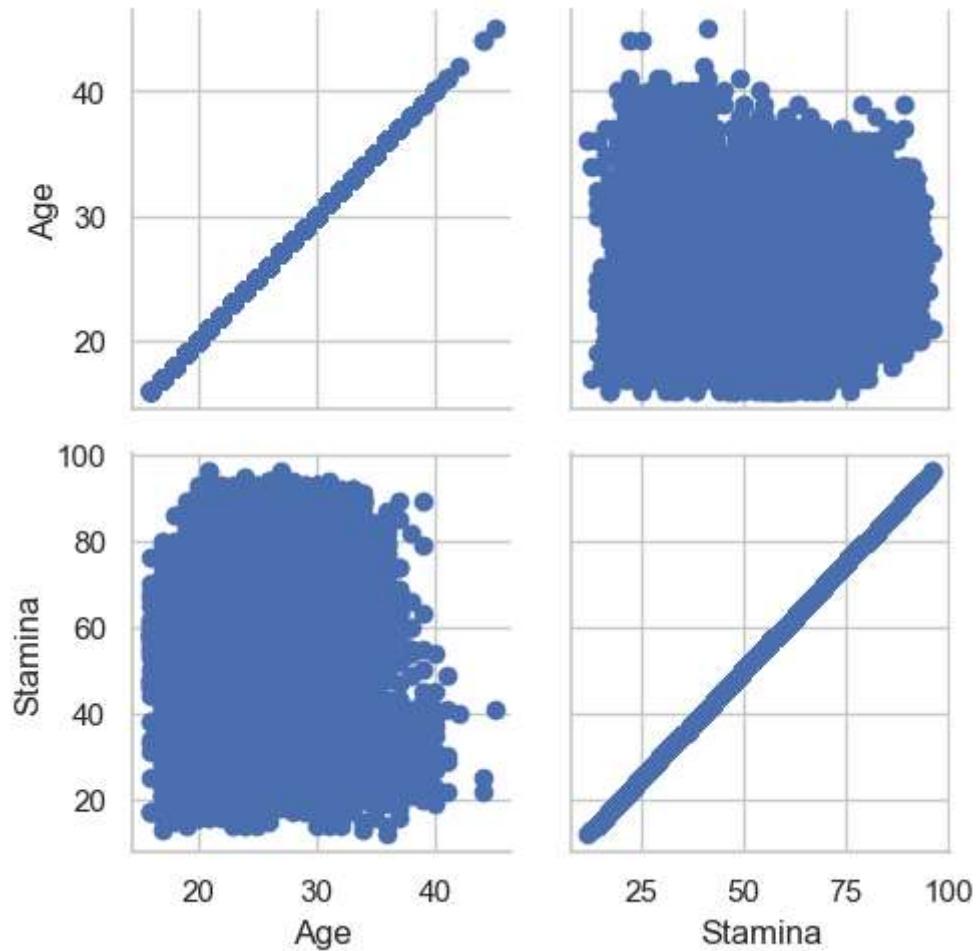
```
In [63]: g = sns.PairGrid(fifa19_new, hue="Preferred Foot")
g = g.map_diag(plt.hist, histtype="step", linewidth=3)
g = g.map_offdiag(plt.scatter)
g = g.add_legend()
```



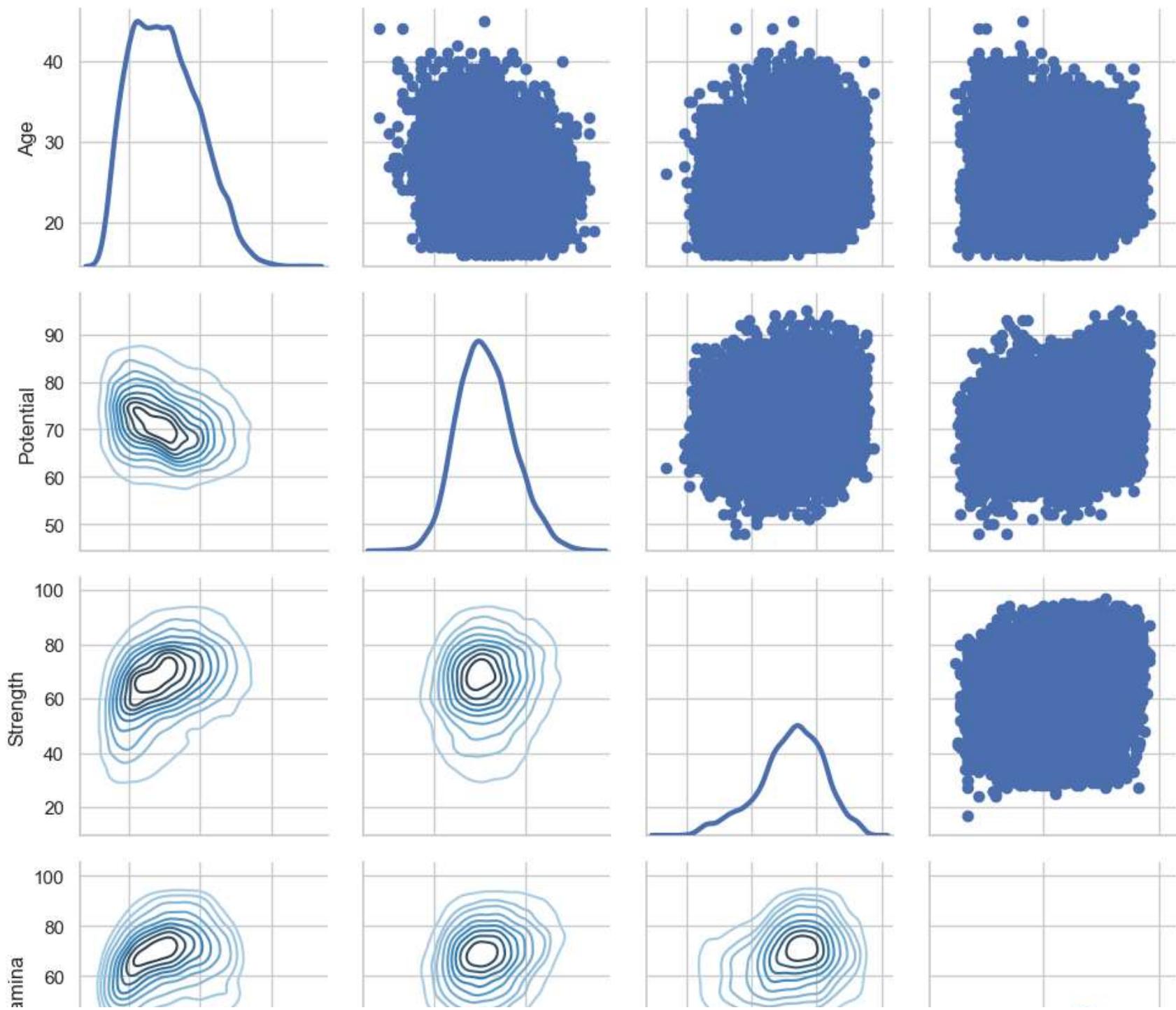


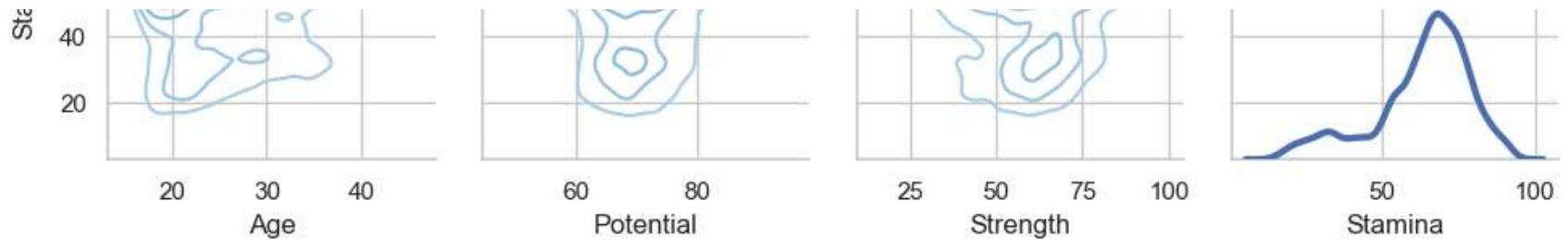
We can plot a subset of variables as follows-

```
In [64]: g = sns.PairGrid(fifa19_new, vars=['Age', 'Stamina'])
g = g.map(plt.scatter)
```



```
In [65]: g = sns.PairGrid(fifa19_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)
```



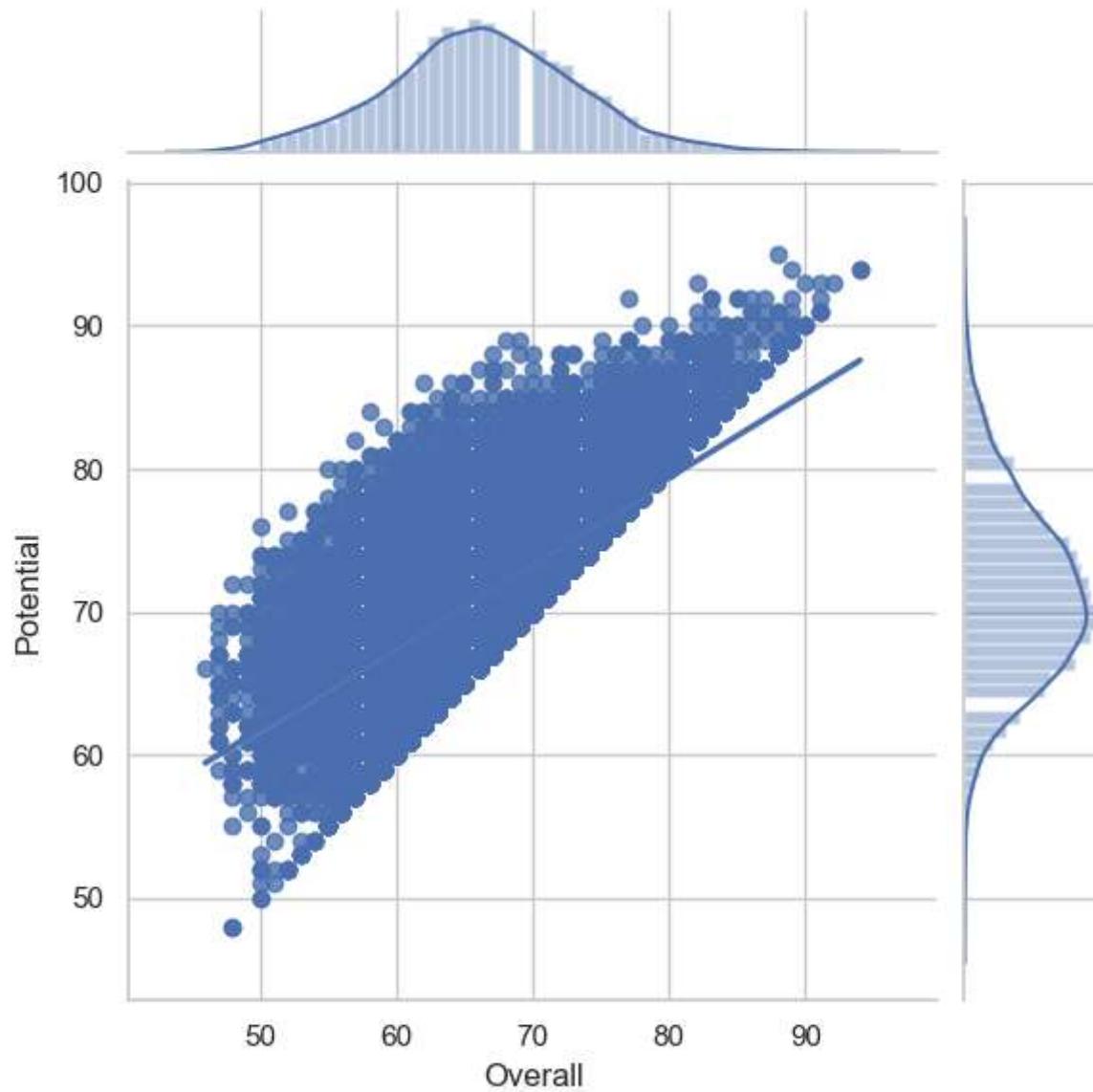


## Seaborn Jointgrid() function

- 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 [66]: g = sns.JointGrid(x="Overall", y="Potential", data=df)
g = g.plot(sns.regplot, sns.distplot)
```

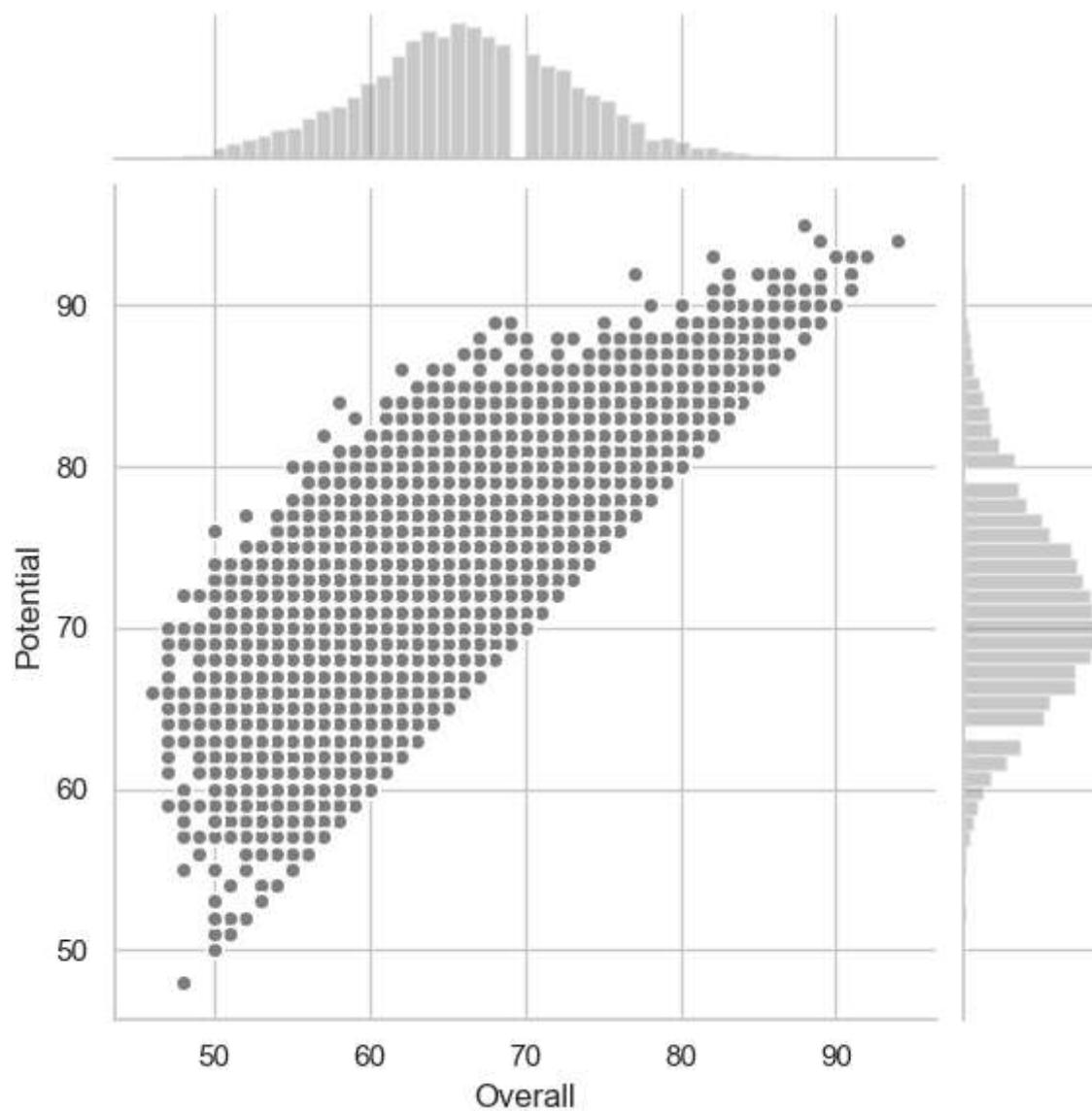


We can draw the joint and marginal plots separately, which allows finer-level control other parameters as follows -

```
In [67]: import matplotlib.pyplot as plt
```

```
In [68]: g = sns.JointGrid(x="Overall", y="Potential", data=df)
g = g.plot_joint(plt.scatter, color=".5", edgecolor="white")
```

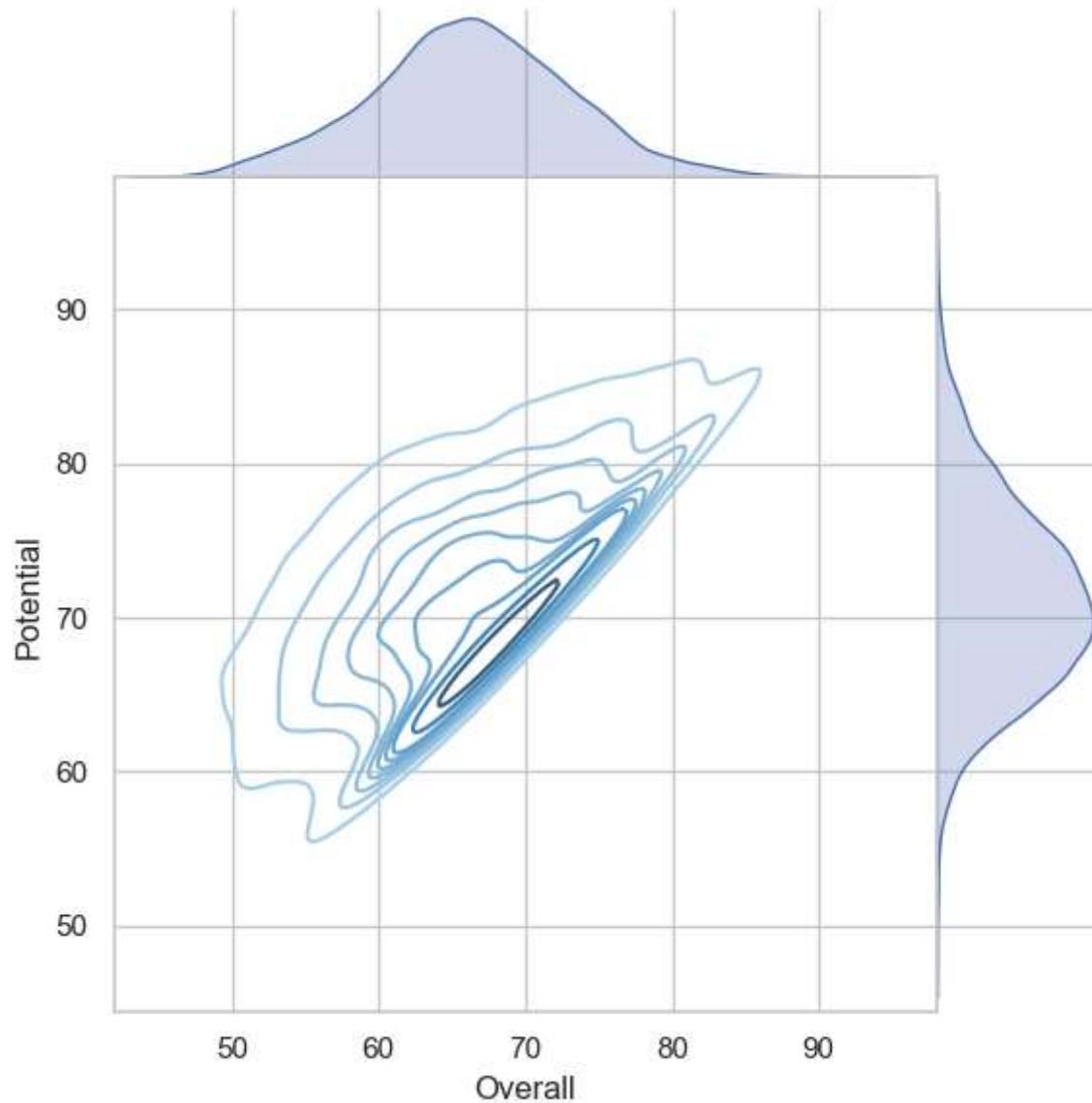
```
g = g.plot_marginals(sns.distplot, kde=False, color=".5")
```



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

```
In [69]: g = sns.JointGrid(x="Overall", y="Potential", data=df, space=0)
g = g.plot_joint(sns.kdeplot, cmap="Blues_d")
```

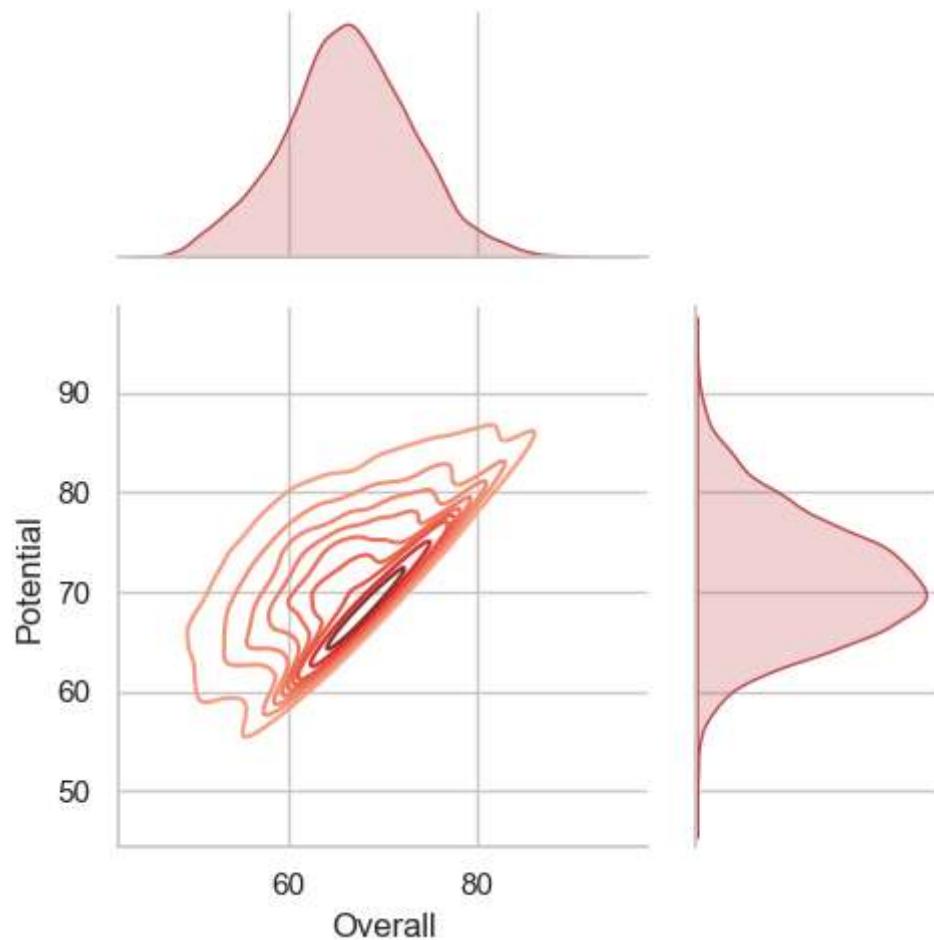
```
g = g.plot_marginals(sns.kdeplot, shade=True)
```



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

```
In [70]: g = sns.JointGrid(x="Overall", y="Potential", data=df, height=5, ratio=2)
g = g.plot_joint(sns.kdeplot, cmap="Reds_d")
```

```
g = g.plot_marginals(sns.kdeplot, color="r", shade=True)
```

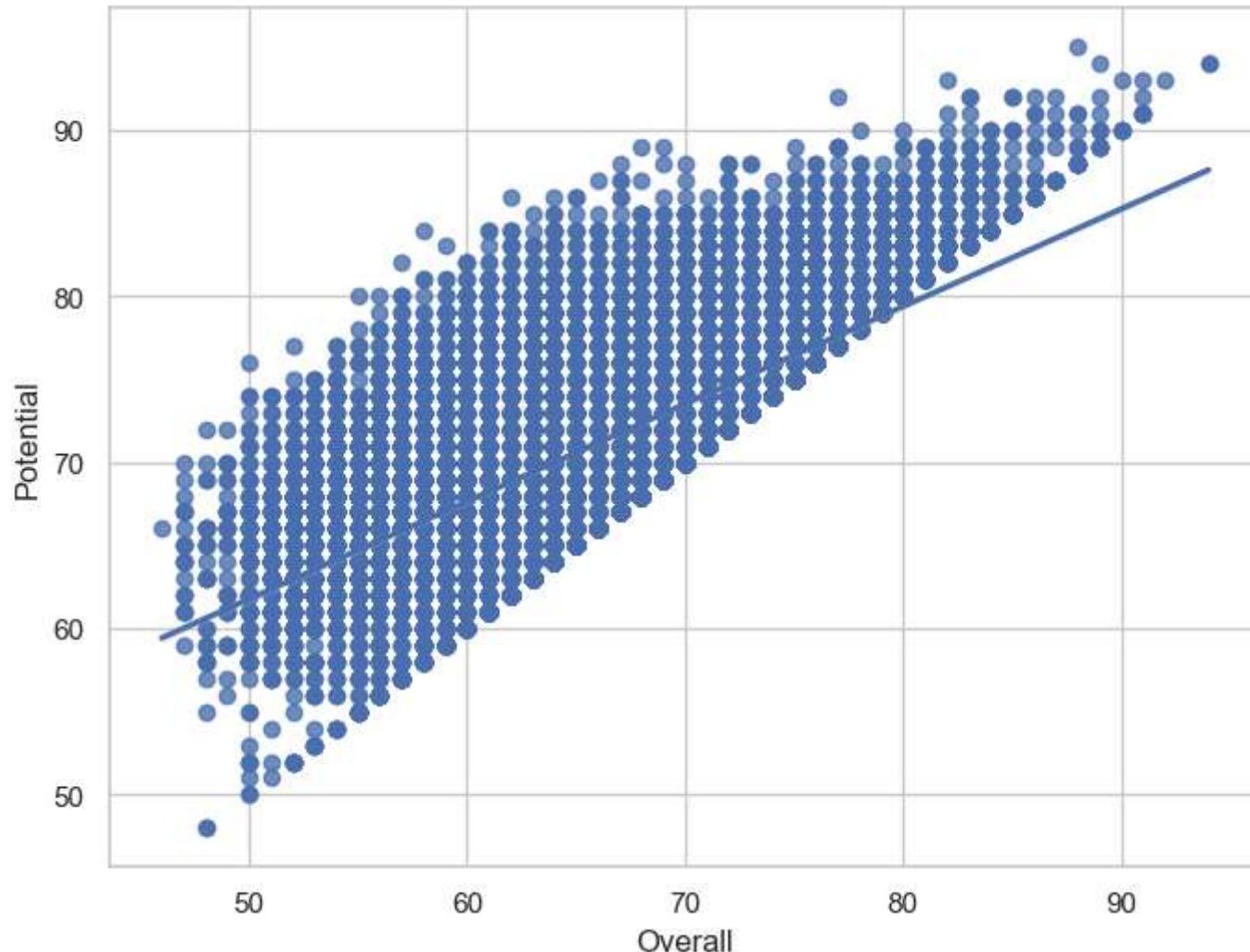


## Controlling the size and shape of the plot

- 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-

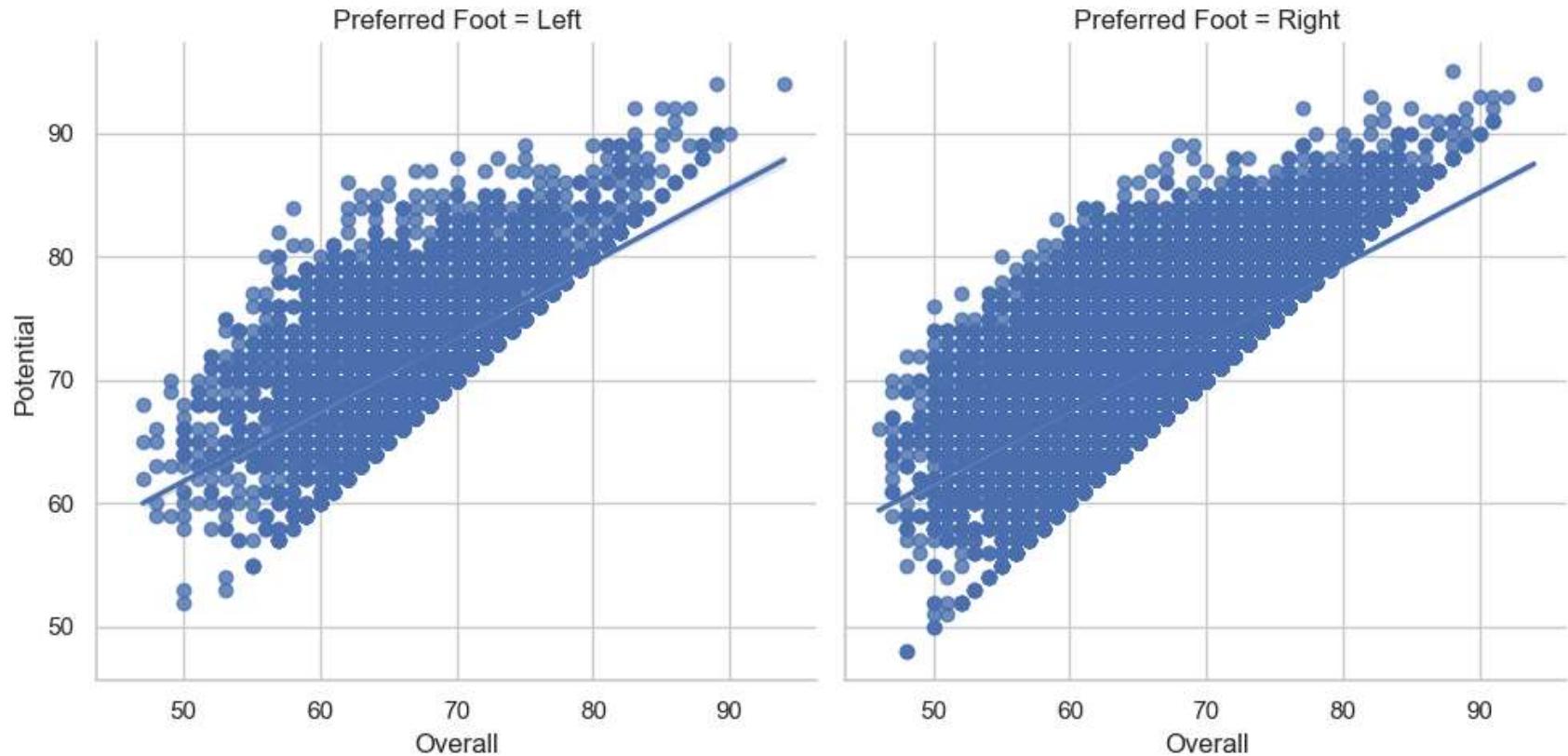
```
In [71]: f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="Overall", y="Potential", data=df);
```



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

```
Out[72]: <seaborn.axisgrid.FacetGrid at 0x19459037fb0>
```



## Seaborn figure styles

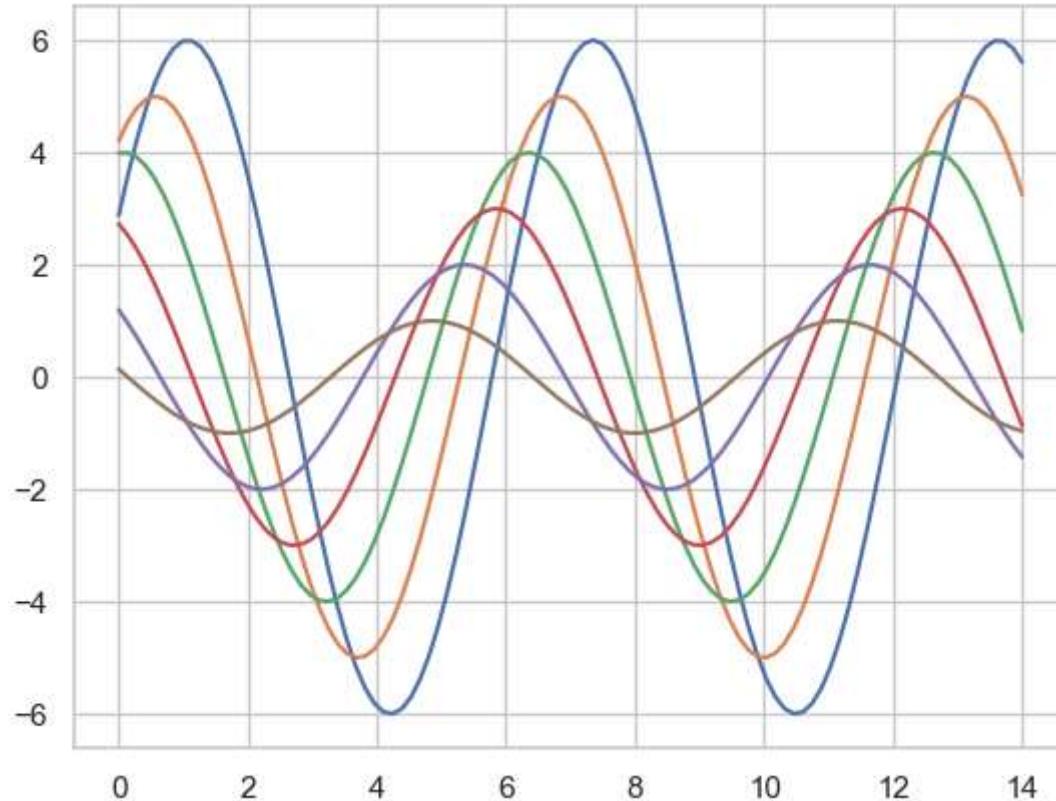
- There are five preset seaborn themes: `darkgrid`, `whitegrid`, `dark`, `white` and `ticks`.
- They are each suited to different applications and personal preferences.
- The default theme is `darkgrid`.

- 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:

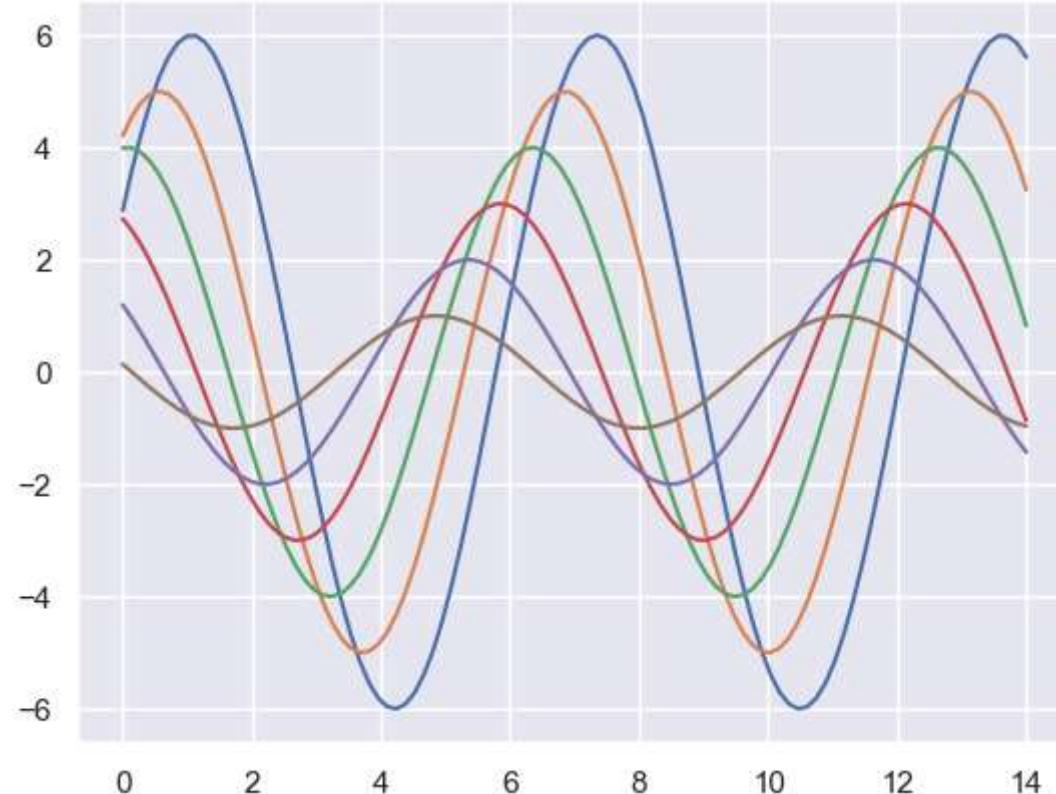
I will define a simple function to plot some offset sine waves, which will help us see the different stylistic parameters as follows -

```
In [73]: 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 [74]: sinplot()
```

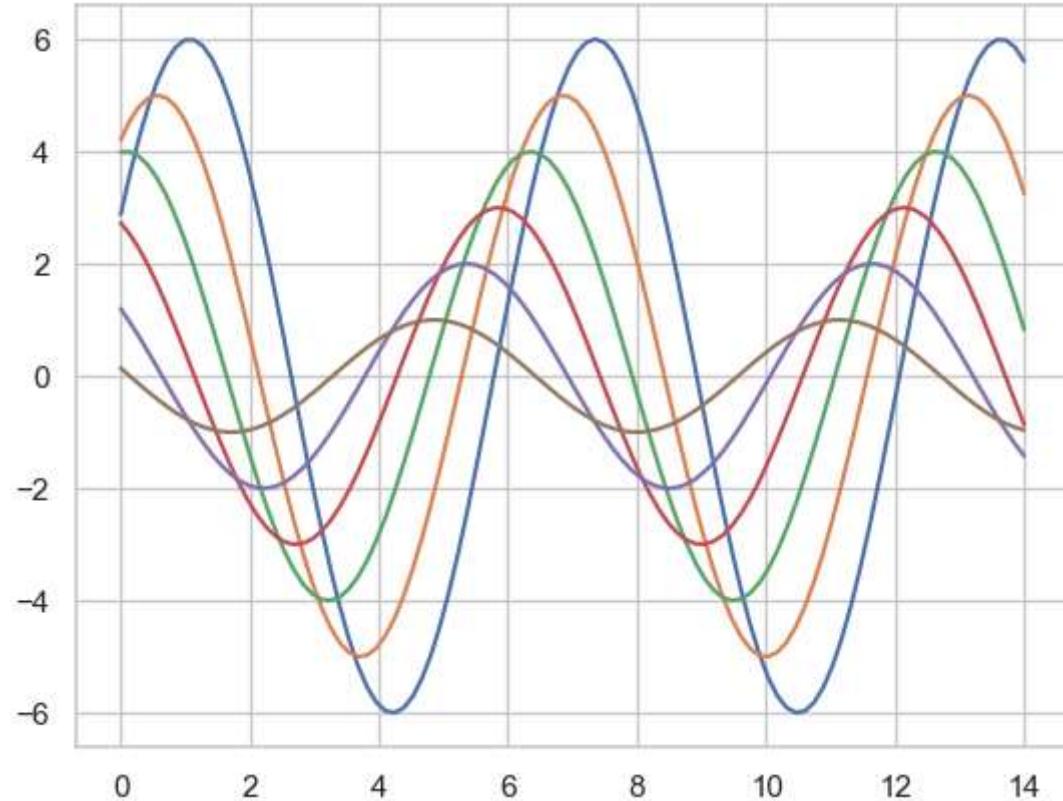


```
In [75]: sns.set()  
sinplot()
```

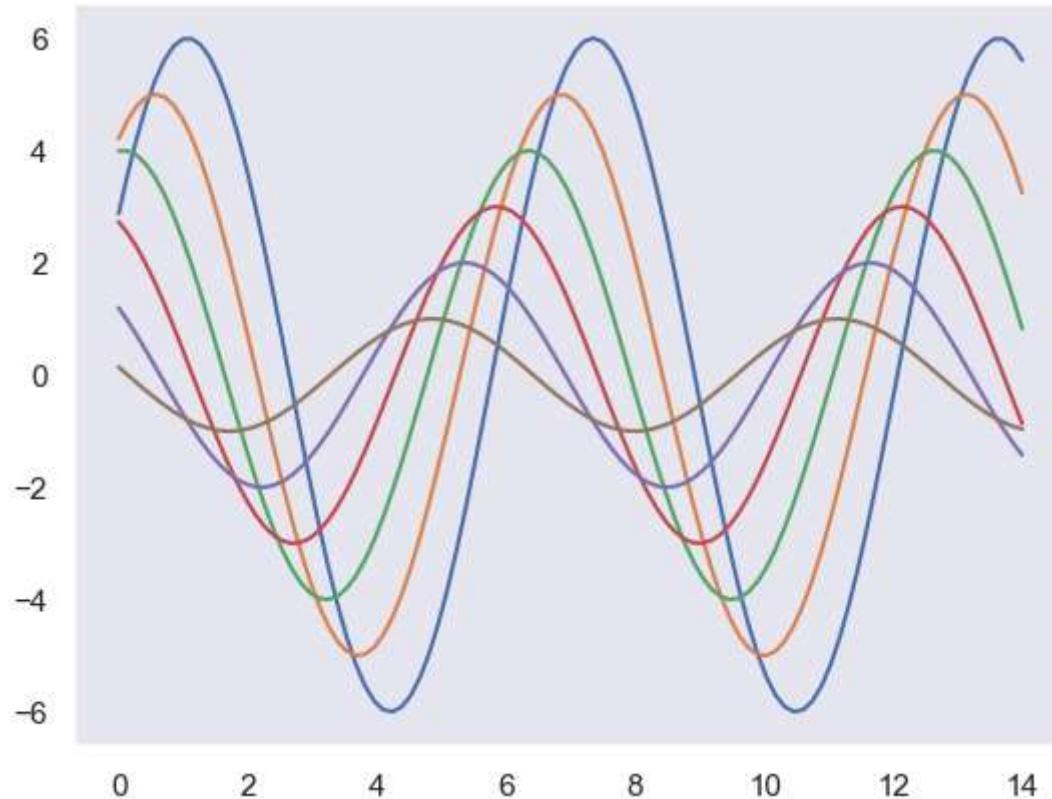


- We can set different styles as follows -

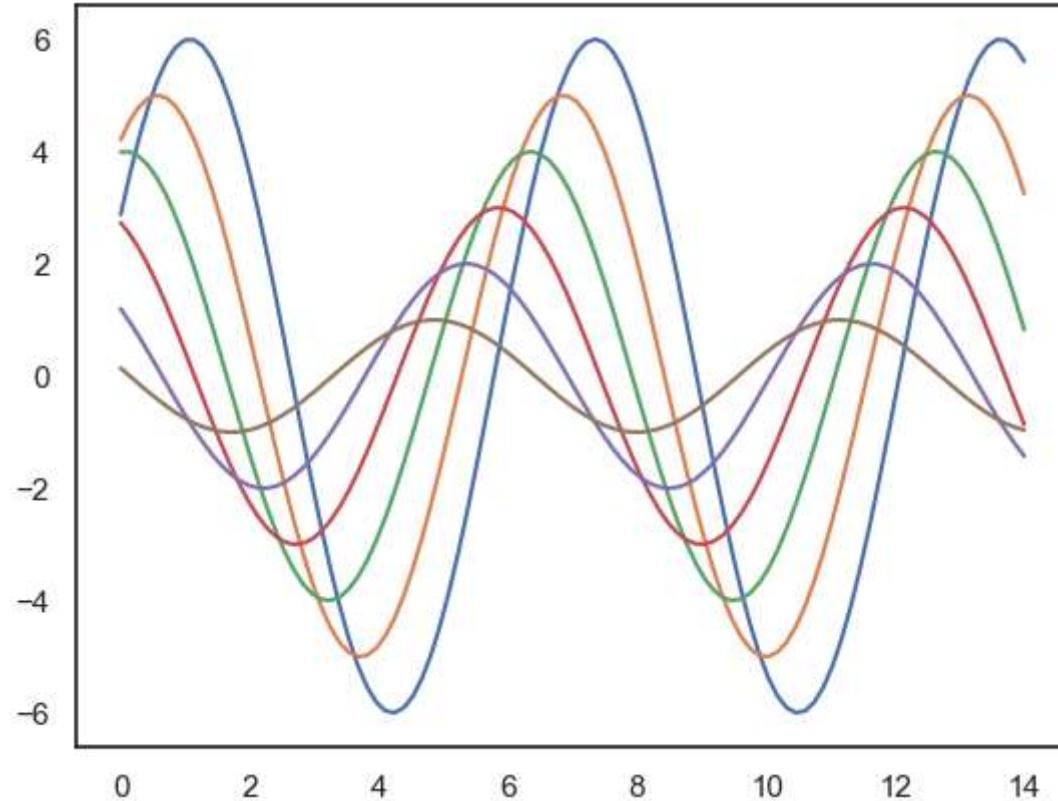
```
In [76]: sns.set_style("whitegrid")  
sinplot()
```



```
In [77]: sns.set_style("dark")
sinplot()
```



```
In [78]: sns.set_style("white")
sinplot()
```



```
In [79]: sns.set_style("ticks")
sinplot()
```

