Hands-on Lab

Data Visualization with Bokeh

Author: Andrea Giussani

Description

In this Hands-on lab, you will master your knowledge on Bokeh, a very popular Python library for dynamic data visualization. Here, you will build standalone plots, and add interactive tools and features, including dynamic legends and Hover inspectors.

Before starting this lab, you are strongly encouraged to take the following courses:

- Data Wrangling with Pandas (https://cloudacademy.com/course/data-wrangling-with-pandas-1089/?context_resource=lp&context_id=1988).
- Data Visualisation with Python using Matplotlib (https://cloudacademy.com/course/data-visualization-with-python-using-matplotlib-1127/advanced-customization-in-matplotlib/?context_resource=lp&context_id=2148).
- Interactive Data Visualization with Python using Bokeh (https://cloudacademy.com/course/interactive-data-visualization-with-python-using-bokeh-1271/introduction/?context_id=2148&context_resource=lp).

Your data visualization skills will be challenged, and by the end of this lab, you should have a deep understanding of how Bokeh practically works.

Learning Objectives

Upon completion of this lab you will be able to:

- build a standard plot with Bokeh using a line glyph;
- bring interactivity inside a bokeh plot with a dynamic legend;
- create a Column Data Source;
- enrich a bokeh plot with the Hover Inspector;
- visualize cagtegorical variables with a bar chart;
- create a multi-index bar plot for catgorical variables;
- build a histogram with Bokeh.

Intended Audience

This lab is intended for:

- Those interested in performing data visualization with Python.
- Anyone involved in data science and engineering pipelines.

Prerequisites

You should possess:

- An intermediate understanding of Python.
- Basic knowledge of the following libraries: Pandas, Matplotlib, Numpy.

1. Data Preparation

To import the data in our working memory, we use the Python `pandas` library, a very popular data management Python library. We are going to employ the `pd.read_csv` method to ingest the data, and then using simple slicing operations to reduce the dimensionality of the dataset.

In [1]:

```
import pandas as pd
```

In [2]:

```
df_all = pd.read_csv('data/all_fifa_data.zip')
```

Let us print the first two rows of our dataset: we have 107 columns.

In [3]:

```
df_all.head(2)
```

Out[3]:

	sofifa_id	player_url	short_name	long_name	age	dob	height_
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	27	1987-06-24	
1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	29	1985-02-05	

2 rows × 107 columns

Possibly we can reduce the dimensionality of this dataset. We indeed retain a few original columns. This is done for you in the next cell.

In [4]:

```
filtered_df = df_all[[
    'sofifa_id', 'short_name', 'age',
    'nationality', 'club_name', 'overall',
    'potential', 'value_eur', 'wage_eur', 'year'
]]
```

Now the `filtered_df` contains fewer columns than the orginal one. To give you an example, the next cell produces the time series related to the football player `L. Messi` .

In [5]:

```
filtered_df[filtered_df.short_name.str.contains('L. Messi')]
```

Out[5]:

	sofifa_id	short_name	age	nationality	club_name	overall	potential	value_eur	wa
0	158023	L. Messi	27	Argentina	FC Barcelona	93	95	100500000	
16155	158023	L. Messi	28	Argentina	FC Barcelona	94	95	111000000	
31779	158023	L. Messi	29	Argentina	FC Barcelona	93	93	89000000	
49376	158023	L. Messi	30	Argentina	FC Barcelona	93	93	105000000	
67330	158023	L. Messi	31	Argentina	FC Barcelona	94	94	110500000	
85414	158023	L. Messi	32	Argentina	FC Barcelona	94	94	95500000	
103897	158023	L. Messi	33	Argentina	FC Barcelona	93	93	67500000	

2. Basic Plotting with Bokeh

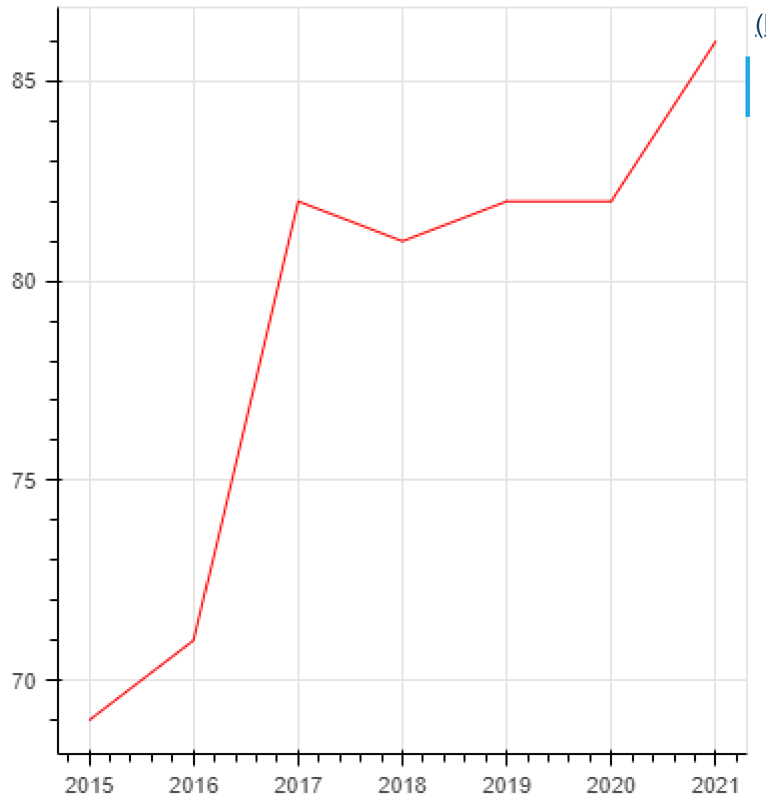
Bokeh has a nice submodule called `plotting` which contains the `figure` method. This is the object you need to create and customize a figure in bokeh. So it is worth you fully understand it before moving to more complex topics. The figure method has several class methods that are used to draw a plot inside a figure object, and those are called `glyphs` : if you are curious, you can watch the course on `Data Visualization with Bokeh` available in our content library. One possibility is to use a *line* glyph:

In this section, we take into account the overall score evolution for `J. Vardy` , a British football player, and we plot it over time. To create a standard line glyph in Bokeh, we need to initialize a `figure` object, and then applying the `line` method on it. This is done for you in the next cell.

In [6]:

```
from bokeh.plotting import figure, show, output_notebook
output_notebook() # we need to specify this to show the plot in the cell
p = figure(plot_width=400, plot_height=400)
p.line(x='year', y='overall', source=df_all.query('short_name=="J. Vardy"'), color='re
d')
show(p)
```

(https://bokeh.org) Loading BokehJS ...



We can also add a legend inside the plot, and we can also specify the location with the attribute `legend.location` .

Task 01: Add a Legend inside the Line Plot

You are asked to add a simple legend to the plot. You are asked to add the command

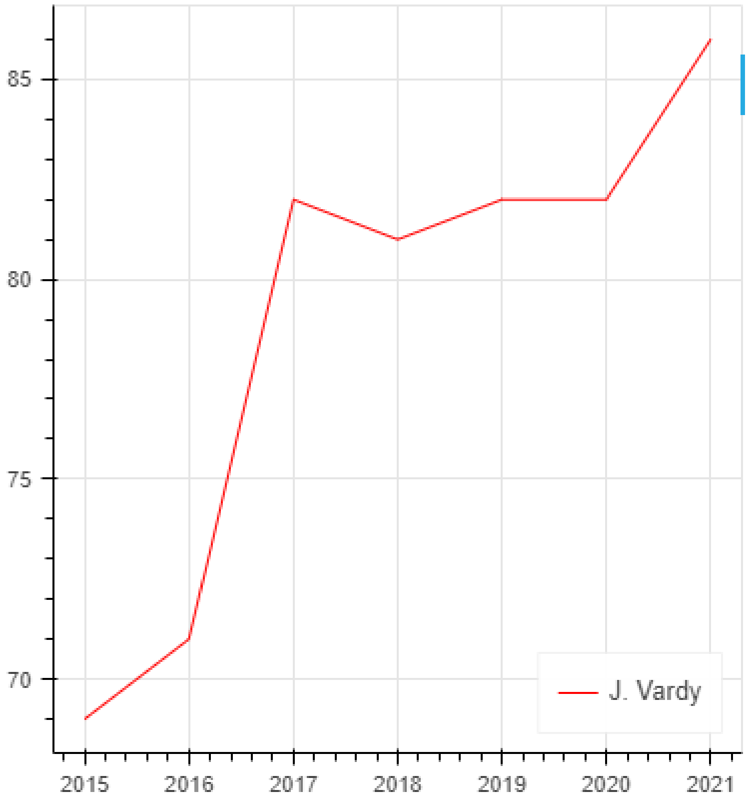
```
p.legend.location = "bottom_right"
```

before showing the plot. Make sure you add to the `line` call the argument `legend_label='J. Vardy'` .

In [7]:

```
from bokeh.plotting import figure, show, output_notebook
output_notebook() # we need to specify this to show the plot in the cell
p = figure(plot_width=400, plot_height=400)
p.line(x='year', y='overall', source=df_all.query('short_name=="J. Vardy"'), color='re
d', legend_label='J. Vardy')
p.legend.location = "bottom_right"
show(p)
```

(https://bokeh.org) BokehJS 2.3.3 successfully loaded.



(http

In [8]:

```
# ====================================
# Validation Check
# DO NOT CHANGE THIS CELL
# ====================================
%history -n 7 -f results/input_vcf_01.txt
with open("results/input_vcf_01.txt", 'r') as my_res:
    lines = my_res.read().splitlines()
    last_line = lines[-2]

with open('results/vcf_01.txt', 'w') as f:
    f.write("%s\n" % last_line)
```

3. Multiple Plots

Ok, so far we have taken into account just one series. Now it's time to plot three distinct series, so that we can easily compare the overall score evolution for the considered players. We first create three distinct pandas dataframes. This is done in the next cell.

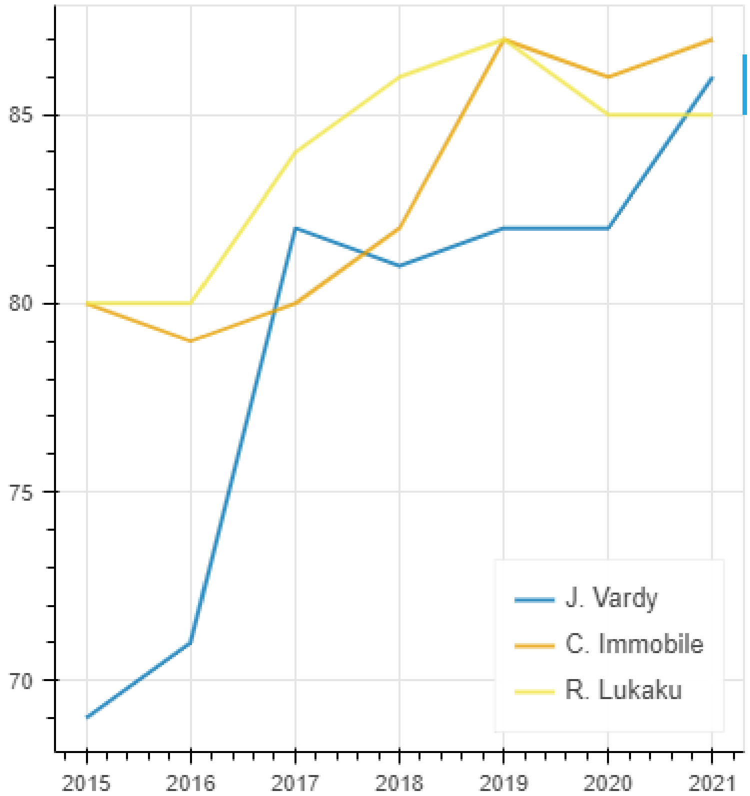
In [10]:

```
vardy_df = filtered_df.query('short_name=="J. Vardy"')
immobile_df = filtered_df.query('short_name=="C. Immobile"')
lukaku_df = filtered_df.query('short_name=="R. Lukaku"')
```

To show three distinct series, we initialize a figure object at first, and then we apply a line glyph for each distinct series. To highlight the distinct patterns, we use a specific color for each player. This is easily done by using the Bokeh `Colorblind3` palette. This is done for you in the next cell.

In [11]:

```
from bokeh.palettes import Colorblind3 # this is a bokeh util that deals the color for
 us
p = figure(plot_width=400, plot_height=400)
for data, name, color in zip([vardy_df, immobile_df, lukaku_df], ["J. Vardy", "C. Immob
ile", "R. Lukaku"], Colorblind3):
    df = pd.DataFrame(data)
    p.line(df['year'], df['overall'], color=color, alpha=0.8, legend_label=name, line_w
idth=2)

p.legend.location = "bottom_right"
show(p)
```



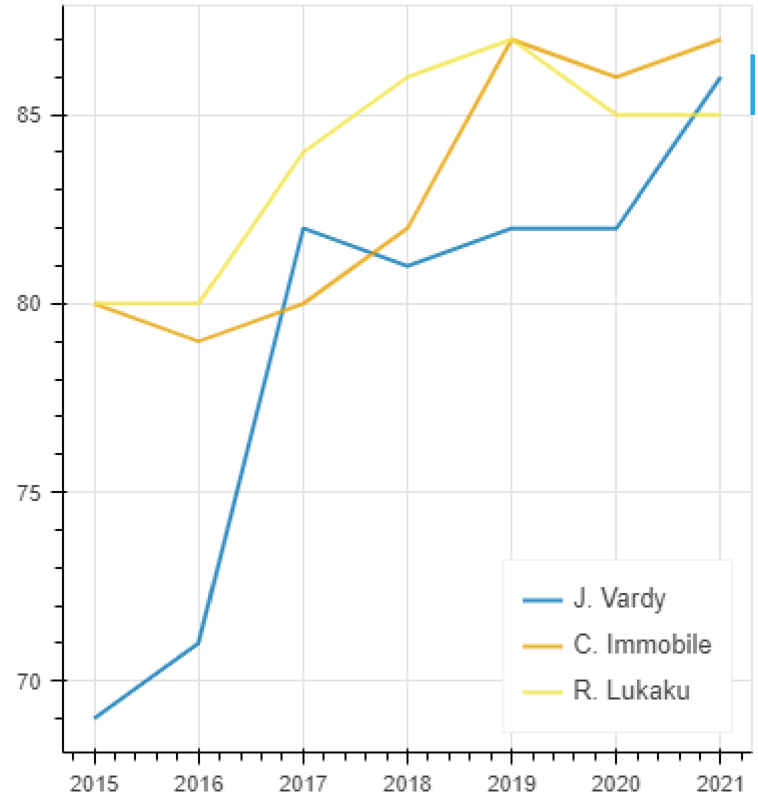
(http

4. Adding Interactivity with a Legend

Legends added to Bokeh plots can be made interactive in case one needs to mute a certain glyph in a plot. These modes are activated by setting the `click_policy` property on a Legend to either `"hide"` or `"mute"` . Once you run the next cell, try to click on the legend: the selected player will be deactivated!

In [12]:

```
p = figure(plot_width=400, plot_height=400)
for data, name, color in zip([vardy_df, immobile_df, lukaku_df], ["J. Vardy", "C. Immob
ile", "R. Lukaku"], Colorblind3):
    df = pd.DataFrame(data)
    p.line(df['year'], df['overall'], line_width=2, color=color, alpha=0.8, legend_labe
l=name)

p.legend.location = "bottom_right"
p.legend.click_policy="hide"
show(p)
```



(http

This is the beauty of Bokeh: with just one simple instruction, we have given a character to our plot. Well done!

5. Adding Interactivity with Inspectors

Bokeh comes with a number of interactive tools that can be used to report information, such as the gestures, which are tools that respond to single gestures. You see them on each single Bokeh plot in its top right location. In particular, for each type of gesture, one tool can be active at any given time, and the active tool is indicated on the toolbar by a highlight next to the tool icon.

But there are other type of tools in bokeh. An example is the family of `Inspectors` .

Inspectors are passive tools that report information about the plot, based on the current cursor position. Any number of inspectors may be active at any given time. The inspectors menu in the toolbar allows users to toggle the active state of any inspector. The most famous member of this familiy is by far the `Hover` tool.

Before getting our hands dirty, let us also introduce the concept of `Column Data Source (CDS)` : this is the corresponding concept of DataFrame in Pandas, but it is more efficient when used to store (and process) data in Bokeh.

Task 02: Create a Column Data Source

You are asked to create a Bokeh `ColumnDataSource` for the `vardy_df` object. You are asked to:

1. import the `ColumnDataSource` class from the `bokeh.models` ;
2. and then applying it to the `vardy_df` . Be sure you store the new object inside the `vardy_src_df` variable.

In [13]:

```
from bokeh.models import ColumnDataSource

vardy_src_df = ColumnDataSource(vardy_df)
```

In [14]:

```
# ====================================
# Validation Check
# DO NOT CHANGE THIS CELL
# ====================================
vcf_02 =  type(vardy_src_df)
with open('results/vcf_02.txt', 'w') as f:
    f.write("%s\n" % vcf_02)
```

We can access to the data with the `data` attribute:

In [15]:

```
vardy_src_df.data
```

Out[15]:

```
{'index': array([  3651,  19780,  32046,  49766,  67643,  85736, 103958]),
 'sofifa_id': array([208830, 208830, 208830, 208830, 208830, 208830, 20883
0]),
 'short_name': array(['J. Vardy', 'J. Vardy', 'J. Vardy', 'J. Vardy', 'J.
Vardy',
        'J. Vardy', 'J. Vardy'], dtype=object),
 'age': array([27, 28, 29, 30, 31, 32, 33]),
 'nationality': array(['England', 'England', 'England', 'England', 'Englan
d', 'England',
        'England'], dtype=object),
 'club_name': array(['Leicester City', 'Leicester City', 'Leicester City',
        'Leicester City', 'Leicester City', 'Leicester City',
        'Leicester City'], dtype=object),
 'overall': array([69, 71, 82, 81, 82, 82, 86]),
 'potential': array([71, 71, 82, 81, 82, 82, 86]),
 'value_eur': array([ 1100000,  1700000, 19500000, 17000000, 20000000, 175
00000,
        28000000]),
 'wage_eur': array([ 15000,  25000, 100000,  90000, 100000, 110000, 16000
0]),
 'year': array([2015, 2016, 2017, 2018, 2019, 2020, 2021])}
```

Let us create three distincts `ColumnDataSource` for the distinct players we have considered so far.

In [16]:

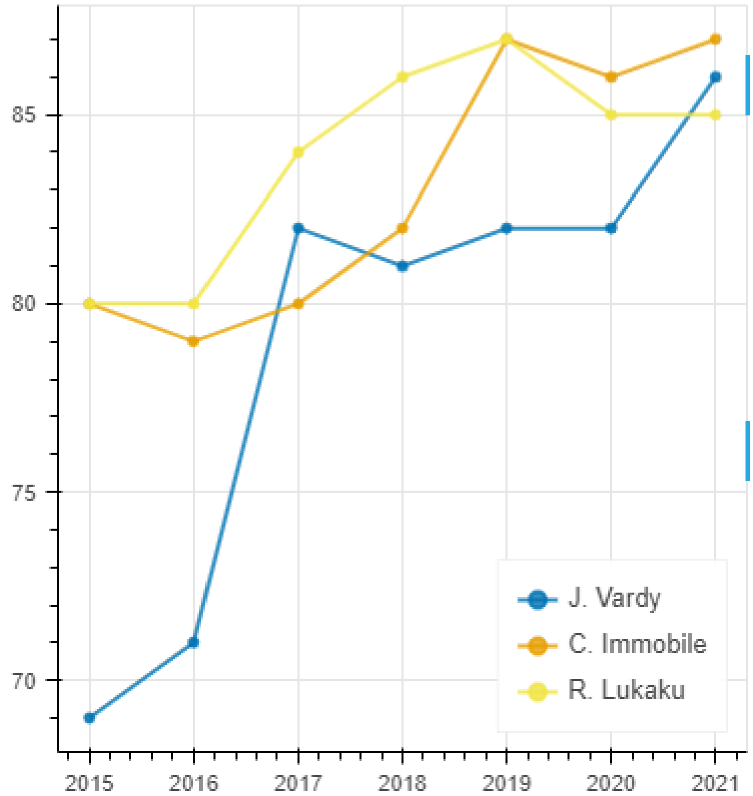
```
vardy_src_df = ColumnDataSource(vardy_df)
immobile_src_df = ColumnDataSource(immobile_df)
lukaku_src_df = ColumnDataSource(lukaku_df)
```

The hover tool is used to generate a tabular tooltip containing information for a particular row of the dataset. Typically, the labels and values we wish to display are supplied as a list of `(label, value)` tuples.

We first import the `HoverTool` class from the `bokeh.models` and then we specify which information we wish to display inside the `tooltips` variable. This is done for you in the next cell.

In [20]:

```
from bokeh.models import HoverTool

tooltips = [
            ('Player', '@short_name'),
            ('Age', '@age'),
            ('Club', '@club_name'),
            ('Mkt Value', '@value_eur'),
            ('Wage', '@wage_eur')
           ]


p = figure(plot_width=400, plot_height=400)
for data, name, color in zip([vardy_src_df, immobile_src_df, lukaku_src_df], ["J. Vard
y", "C. Immobile", "R. Lukaku"], Colorblind3):
    p.line('year', 'overall', source=data, line_width=2, color=color, alpha=0.8, legend
_label=name)
    p.circle('year', 'overall', source=data, line_width=2, color=color, alpha=0.8, lege
nd_label=name)

p.add_tools(HoverTool(tooltips=tooltips, mode='mouse'))

p.legend.location = "bottom_right"
p.legend.click_policy="hide"

show(p)
```



(http

Try to move the mouse on the plot. That's impressive, isn't it? We see that once we hover the mouse over the observed data points, an extra tool appears in the plot: that contains the information we have specified in the variable `tooltips` .

6. Plotting Categorical Data

In this section we investigate different tools to plot categorical data with Bokeh.

The next method is an util that ahs been created to return the number of football players, grouped by country, for a specific year. In particular, it returns only the first `top_n` countries for that specified year.

In [21]:

```
def get_top_countries(df, year_filter, top_n=3):
    df_tmp = df.query('year==@year_filter')[['short_name', 'nationality']].groupby('nat
ionality').count().rename(
        columns={'short_name': 'cnt'}
        ).sort_values(
        by='cnt', ascending=False
    )
    top_countries = df_tmp.head(top_n).reset_index()
    return top_countries
```

We apply the above method for the `year=2021` and with `top_n=10` .

In [22]:

```
top10_countries =  get_top_countries(filtered_df, year_filter=2021, top_n=10)
```

In [23]:

```
top10_countries
```

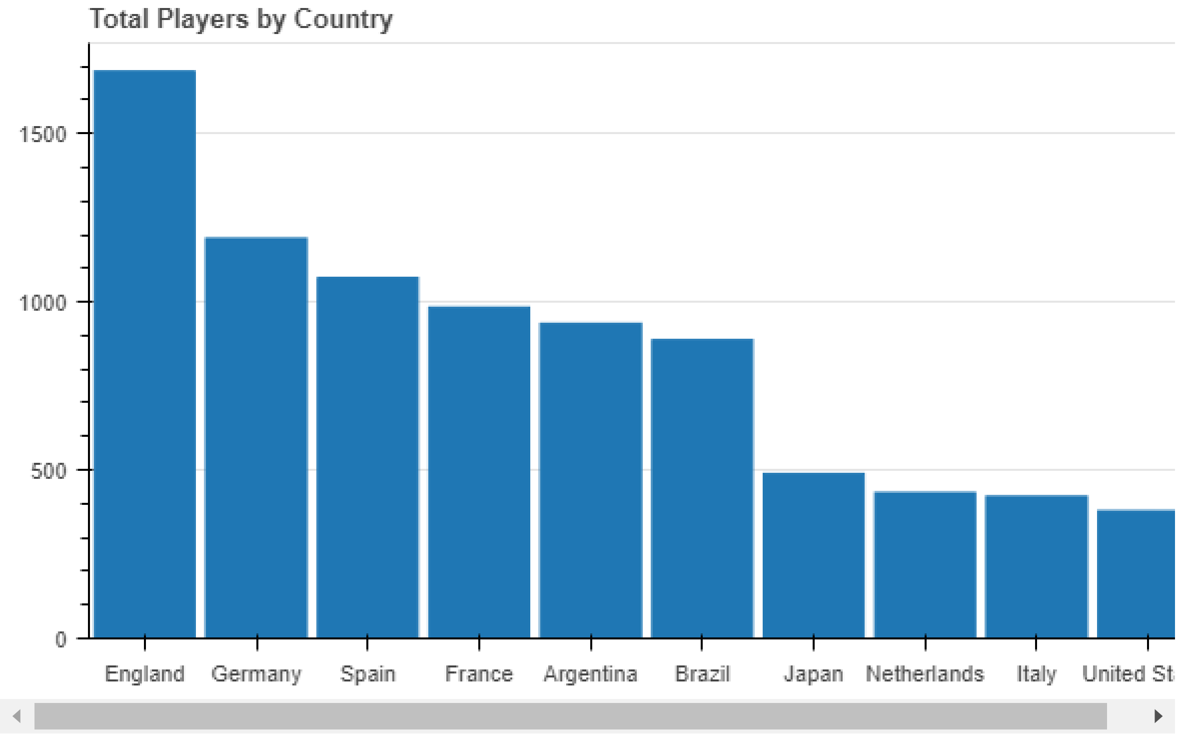
Out[23]:

	nationality	cnt
0	England	1685
1	Germany	1189
2	Spain	1072
3	France	984
4	Argentina	936
5	Brazil	887
6	Japan	489
7	Netherlands	432
8	Italy	421
9	United States	378

We want to plot the top 10 Countries with respect to the number of players. Hence, we plot the total count of players by countries with a `vbar` glyph. We also set the argument `x_range` , inside the figure object, as equals to the series `top10_countries.nationality` , which is already sorted: in this way the data is going to be shown in descending order.

In [26]:

```
p = figure(x_range=top10_countries.nationality,
           plot_height=350,
           title="Total Players by Country",
           toolbar_location=None,
           tools=""
           )

p.vbar(x='nationality', top='cnt', source=top10_countries, width=0.9)

p.xgrid.grid_line_color = None
p.y_range.start = 0

show(p)
```



Now we want to investigate how the average wage for three countries (namely England, Spain and Italy) evolved in the last, say, three years.

To do so we need a little bit of data wrangling. This has been done for you down below here.

In [31]:

```
pivot_table_wages = pd.pivot_table(
    data=filtered_df,
    index='nationality',
    columns='year',
    aggfunc='mean',
    values='wage_eur'
)
pivot_table_wages.columns = pivot_table_wages.columns.map(str)
countries = pivot_table_wages.loc[['England', 'Italy', 'Spain']].index.to_list()
years = pivot_table_wages.loc[['England', 'Italy', 'Spain']].columns.to_list()[4:] # 4:
is the years
```

In [32]:

```
pivot_table_wages.loc[['England', 'Italy', 'Spain'], years]
```

Out[32]:

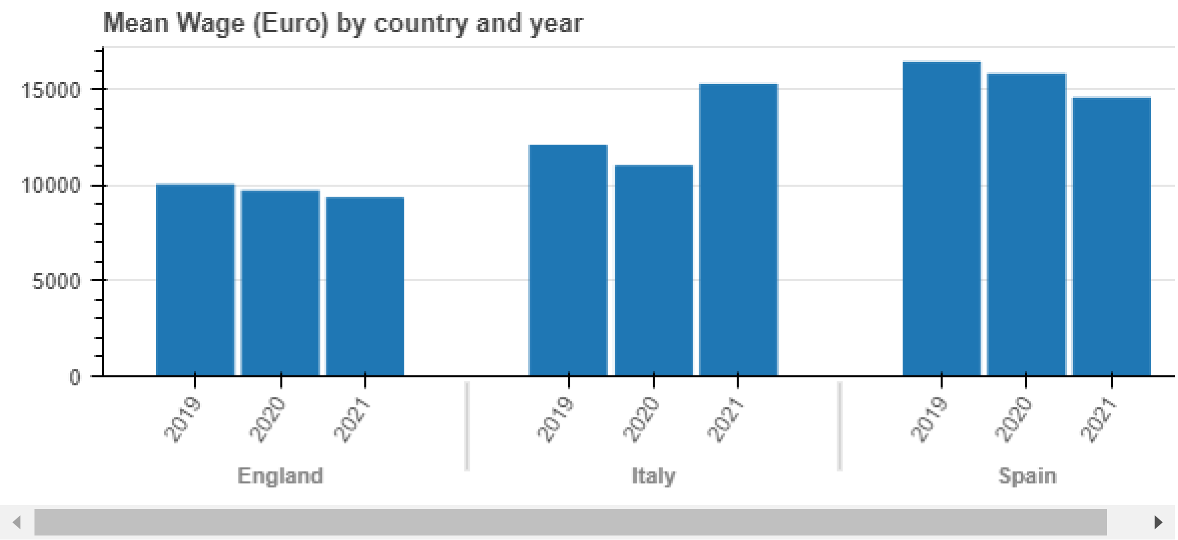
year	2019	2020	2021
nationality			
England	10007.076923	9684.131737	9321.364985
Italy	12071.132597	11000.000000	15233.254157
Spain	16394.985809	15771.291866	14512.033582

From a data visualization perspective, this means we are clustering the average wage by two factors: `year` and `country`. Hence, we need to create a multi-index range for our plot. In Bokeh, this can be easily tackled by using the `FactorRange`, which is a Bokeh class from the `models` submodule to create a range of values for a categorical dimension.

Try to run the next cell: it is going to produced a bar chart clustered on the x-axis with respect to two quantities: `country` and `year`. Everything has been done for you.

In [33]:

```
from bokeh.models import FactorRange


data = {
    'country' : countries,
    '2019'   : pivot_table_wages.loc[['England', 'Italy', 'Spain'], '2019'].to_list(),
    '2020'   : pivot_table_wages.loc[['England', 'Italy', 'Spain'], '2020'].to_list(),
    '2021'   : pivot_table_wages.loc[['England', 'Italy', 'Spain'], '2021'].to_list()
    }

x = [ (country, year) for country in countries for year in years]
counts = sum(zip(data['2019'], data['2020'], data['2021']), ())

source = ColumnDataSource(data=dict(x=x, counts=counts))

p = figure(x_range=FactorRange(*x), plot_height=250,
           title="Mean Wage (Euro) by country and year",
           toolbar_location=None, tools="")

p.vbar(x='x', top='counts', width=0.9, source=source)

p.y_range.start = 0
p.x_range.range_padding = 0.1
p.xaxis.major_label_orientation = 1
p.xgrid.grid_line_color = None

show(p)
```



7. Plot the Normalized Overall Histogram

As a last step for this hands-on lab, we want to create a histogram for the normalized `overall` column. What does normalization mean in this context? Well, from a statistical point of view, we homogenize the data by removing, for each single point, the observed column mean and divide by its standard deviation. This operation is especially useful for those columns which are intrinsically heterogeneous. Think, for example, to the wage: that column might be extremely different across the world for many reasons. Hence, if you want to compare two countries, it is always better to standardize the data.

In [38]:

```
def normalization_data(df, colname):
    normalized_df = (df[colname] - df[colname].mean())/df[colname].std()
    normalized_df = normalized_df[~normalized_df.isna()].to_frame() # to_frame converts
series to df
    return normalized_df
```

In [39]:

```
data_normalized = normalization_data(filtered_df, 'overall')
```

By inspecting the min and max values of the new normalized series, it makes sense to create a grid of 1000 values between -4 and 4.

We also employ the `NumPy` method `histogram` to set the data in the right order to plot a histogram.

In [40]:

```
import numpy as np
hist, edges = np.histogram(data_normalized.overall.to_list(), density=True, bins=50)
```

We now create a new method called `create_histogram` that wraps the `quad()` glyph: this is used to create a histogram by specifying the argument `top=hist` . We also specify down below a few arguments, such as the `xaxis.axis_label` and the `yaxis.axis_label` to improve the readability of the plot.

In [41]:

```
def create_histogram(title, hist, edges):
    p = figure(title=title)
    p.quad(
        top=hist,
        bottom=0,
        left=edges[:-1],
        right=edges[1:],
        fill_color="blue",
        line_color="white",
        alpha=0.5
        )
    p.y_range.start = 0
    p.xaxis.axis_label = 'Normalized Overall (x)'
    p.yaxis.axis_label = 'Pr(x)'
    p.grid.grid_line_color="white"
    return p
```

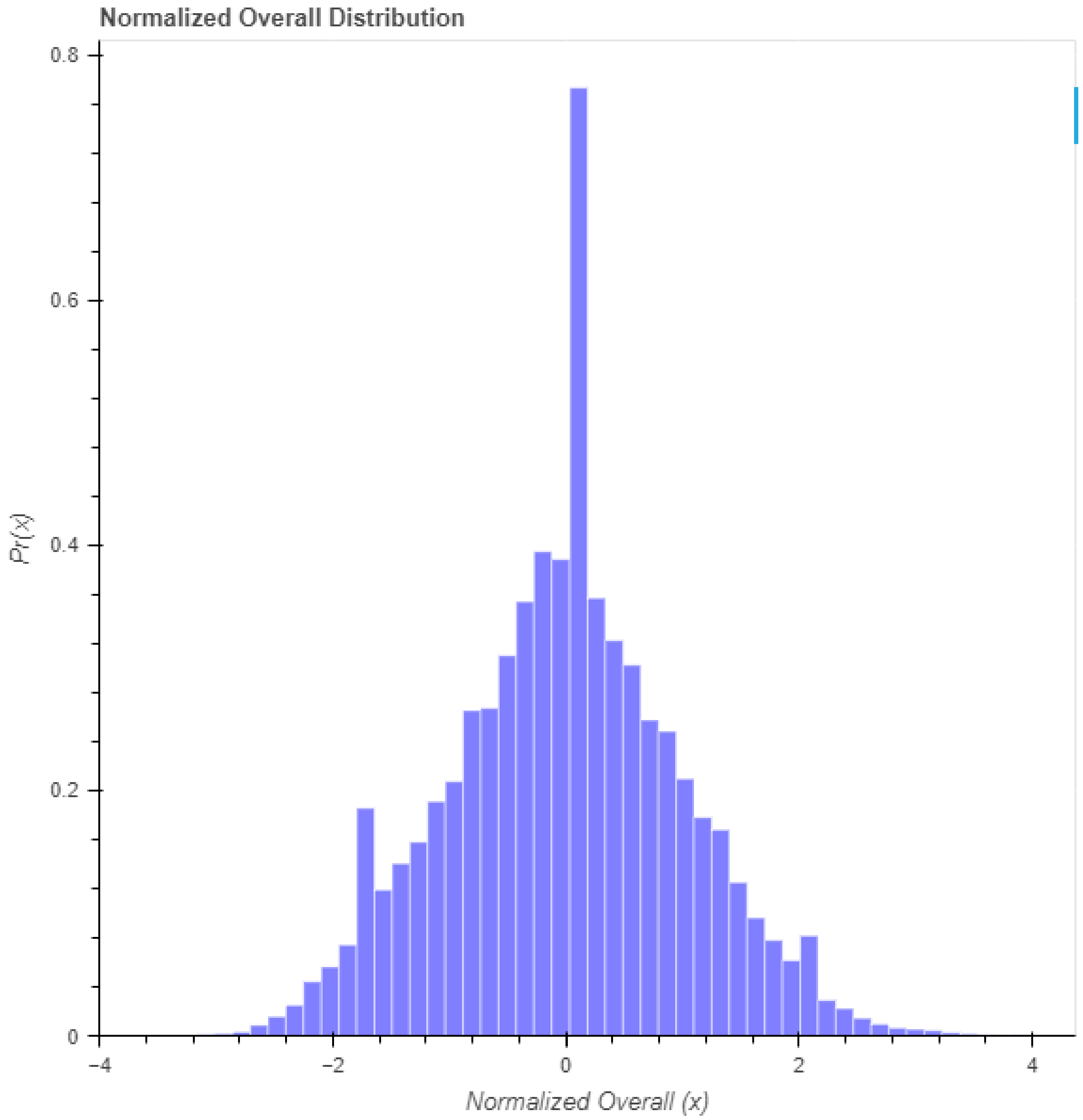
We then call the `create_histogram` method with the following arguments:

- `title="Normalized Overall Distribution"` .
- `hist=hist` .
- `edges=edges`

and we show the plot by calling the `show` method.

In [42]:

```
hist_plot = create_histogram(title="Normalized Overall Distribution", hist=hist, edges=
edges)
show(hist_plot)
```



It looks pretty Gaussian, right? The normalization was successfully performed here, although we have a sifgnificant spike around 0: in general, this makes sense, and so we can easily state that the FIFA video game is populated by average players.

This concludes the hands-on lab on bokeh. If you have any feedback related to this learning path, feel free to contact us at support@cloudacademy.com.

END