Bokeh гордится тем, что является библиотекой для интерактивной визуализации данных. В отличие от популярных аналогов в области визуализации Python, таких как Matplotlib и [Seaborn](https://seaborn.pydata.org/" \t "_blank), Bokeh отображает свою графику с помощью HTML и JavaScript. Это делает его отличным кандидатом для создания веб-панелей и приложений. Тем не менее, это не менее мощный инструмент для изучения и понимания ваших данных или создания красивых пользовательских диаграмм для проекта, или отчета.

**Создание первой фигуры**

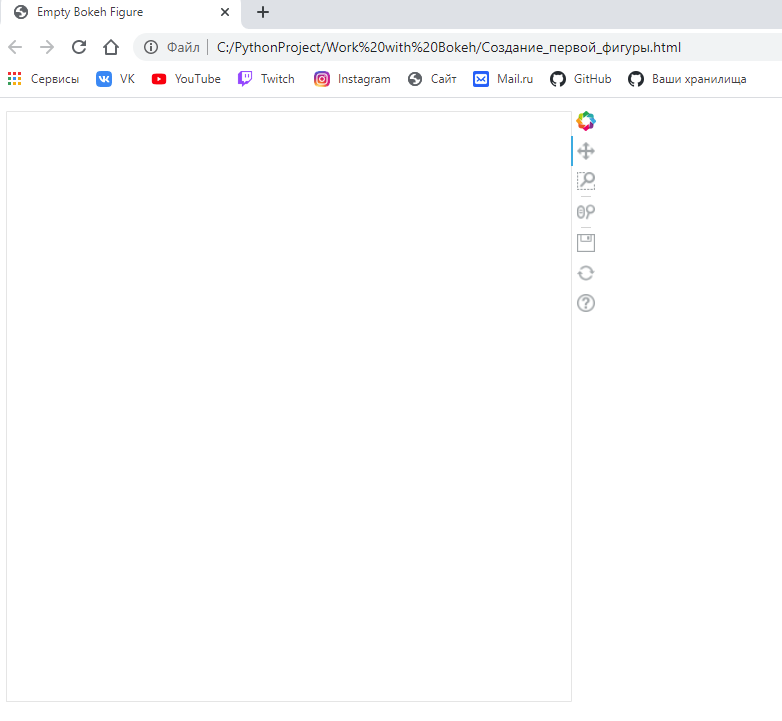
Прежде чем приступить к работе, нам необходимо, соответственно, установить определенные библиотеки, а именно: pandas, numpy и bokeh. Всего есть несколько способов сделать инфографику в Bokeh:

1. **output\_file(‘filename.html’)** - запишет инфографику в статический HTML-файл.
2. **output\_notebook()** - отобразит инфографику непосредственно в Jupyter Notebook.

Важно отметить, что ни одна из функций не отображает визуализацию. Этого не произойдет, пока не будет вызвана функция show(). Однако они гарантируют, что при вызове show() визуализация появится там, где вы хотите. И так как я буду работать в привычном для меня PyCharm, то я буду использовать первый способ. Давайте же создадим первую фигуру:

from bokeh.io import output\_file  
from bokeh.plotting import figure, show  
  
# Рисунок будет отображен в статическом HTML-файле  
output\_file('Создание\_первой\_фигуры.html',  
 title='Empty Bokeh Figure')  
  
# Настроить общий объект figure()  
fig = figure()  
  
# Посмотрите, как это выглядит  
show(fig)

Результат:



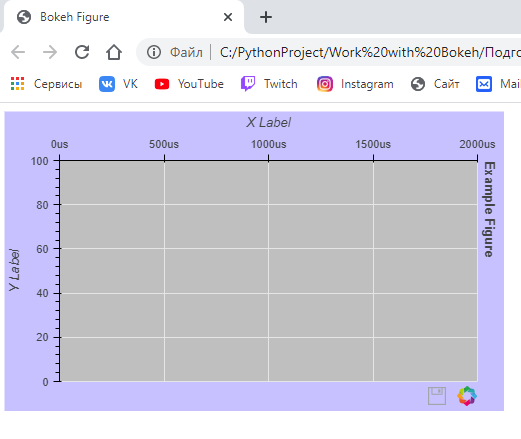
Как видите, новое окно браузера открылось с вкладкой под названием **Empty Bokeh Figure** и пустой фигурой, а сам созданный файл с именем «Создание\_первой\_фигуры.html» появился в папке с проектом.

## Подготовка фигуры к данным

Теперь, когда вы знаете, как создавать и просматривать общую фигуру Bokeh в браузере, пора узнать больше о том, как настроить объект figure(). Объект figure() — это не только основа визуализации данных, но и объект, который открывает все доступные инструменты Bokeh для инфографики. Figure Bokeh является подклассом объекта [Bokeh Plot](https://bokeh.pydata.org/en/latest/docs/reference/models/plots.html" \l "bokeh.models.plots.Plot" \t "_blank), который предоставляет многие параметры, которые позволяют настраивать эстетические элементы инфографики. Чтобы вы могли получить представление о доступных параметрах настройки, давайте создадим самую уродливую инфографика на свете:

from bokeh.plotting import figure, show  
  
output\_file('Подготовка\_фигуры\_к\_данным.html',  
 title='Bokeh Figure')  
  
  
fig = figure(background\_fill\_color='gray',  
 background\_fill\_alpha=0.5,  
 border\_fill\_color='blue',  
 border\_fill\_alpha=0.25,  
 plot\_height=300,  
 plot\_width=500,  
 x\_axis\_label='X Label',  
 x\_axis\_type='datetime',  
 x\_axis\_location='above',  
 x\_range=('2018-01-01', '2018-06-30'),  
 y\_axis\_label='Y Label',  
 y\_axis\_type='linear',  
 y\_axis\_location='left',  
 y\_range=(0, 100),  
 title='Example Figure',  
 title\_location='right',  
 toolbar\_location='below',  
 tools='save')  
  
  
show(fig)

Результат:



## Рисование данных с помощью глифов

## Пустая фигура не так уж и интересна, поэтому давайте посмотрим на глифы. Глиф — это векторизованная графическая форма или маркер, который используется для представления ваших данных в виде круга или квадрата. Начнем с очень простого примера, нарисовав несколько точек на координатной сетке x-y:

#РИСОВАНИЕ ДАННЫХ С ПОМОЩЬЮ ГЛИФОВ  
#Создание первых данных  
from bokeh.io import output\_file  
from bokeh.plotting import figure, show  
  
# Мои данные о координатах x-y  
x = [1, 2, 1]  
y = [1, 1, 2]  
  
  
output\_file('Рисование\_данных\_с\_помощью\_глифов1.html', title='First Glyphs')  
  
# Создайте фигуру без панели инструментов и диапазонов осей  
fig = figure(title='My Coordinates',  
 plot\_height=300, plot\_width=300,  
 x\_range=(0, 3), y\_range=(0, 3),  
 toolbar\_location=None)  
  
# Нарисуйте координаты в виде кругов  
fig.circle(x=x, y=y,  
 color='green', size=10, alpha=0.5)  
  
# Показать сюжет  
show(fig)

## Результат:

## 

## Как только ваша фигура будет создана, вы сможете увидеть, как ее можно использовать для рисования данных координат x-y с использованием настраиваемых круговых глифов. Помимо этого, глифы можно комбинировать по мере необходимости, чтобы соответствовать вашим потребностям в визуализации. Допустим, я хочу создать визуализацию, которая показывает, сколько слов я написал за день, чтобы сделать этот руководство, с наложенной линией тренда совокупного количества слов:

#Создание вторых данных  
import numpy as np  
from bokeh.plotting import figure, show  
  
# Мои данные о подсчете слов  
day\_num = np.linspace(1, 10, 10)  
daily\_words = [450, 628, 488, 210, 287, 791, 508, 639, 397, 943]  
cumulative\_words = np.cumsum(daily\_words)  
  
  
output\_file('Рисование\_данных\_с\_помощью\_глифов2.html', title='First Glyphs')  
  
# Создаем фигуру с осью x типа datetime  
fig = figure(title='My Tutorial Progress',  
 plot\_height=400, plot\_width=700,  
 x\_axis\_label='Day Number', y\_axis\_label='Words Written',  
 x\_minor\_ticks=2, y\_range=(0, 6000),  
 toolbar\_location=None)  
  
# Ежедневные слова будут представлены в виде вертикальных полос (столбцов)  
fig.vbar(x=day\_num, bottom=0, top=daily\_words,  
 color='blue', width=0.75,  
 legend='Daily')  
  
# Накопленная сумма будет линией тренда  
fig.line(x=day\_num, y=cumulative\_words,  
 color='gray', line\_width=1,  
 legend='Cumulative')  
  
# Поместите легенду в левый верхний угол  
fig.legend.location = 'top\_left'  
  
# Давайте проверим  
show(fig)

## Результат:

## 

Чтобы объединить столбцы и линии на рисунке, они просто создаются с использованием одного и того же объекта figure (). Кроме того, вы можете увидеть выше, как легко можно создать легенду, задав свойство легенды для каждого глифа. Затем легенда была перемещена в верхний левый угол графика путем присвоения ‘top\_left’ значению fig.legend.location.

## Небольшая заметка о данных

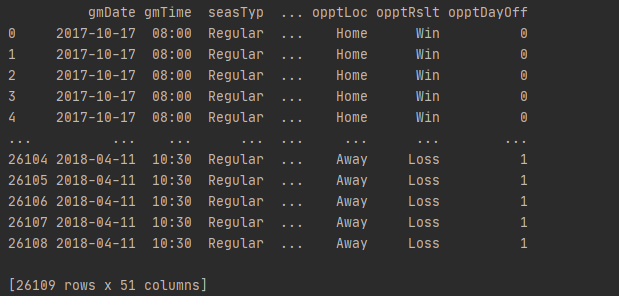
Каждый раз, когда вы изучаете новую библиотеку визуализации, рекомендуется начинать с некоторых данных из знакомой вам области. Поэтому, в остальных примерах будут использоваться общедоступные данные из Kaggle, который содержит [информацию о сезоне 2017-18 Национальной баскетбольной ассоциации (НБА)](https://www.kaggle.com/pablote/nba-enhanced-stats), а именно:

* 2017-18\_playerBoxScore.csv: снимки статистики игроков по играм;
* 2017-18\_teamBoxScore.csv: снимки статистики команд по играм;
* 2017-18\_standings.csv: ежедневное командное положение и рейтинги.

Эти файлы я, соответственно, скачал и разместил в папку с проектом. После чего, они были прочитаны с помощью Pandas DataFrame, используя следующие команды:

import pandas as pd  
# Прочитать csv файлы  
player\_stats = pd.read\_csv('2017-18\_playerBoxScore.csv', parse\_dates=['gmDate'])  
team\_stats = pd.read\_csv('2017-18\_teamBoxScore.csv', parse\_dates=['gmDate'])  
standings = pd.read\_csv('2017-18\_standings.csv', parse\_dates=['stDate'])  
print(player\_stats)

Этот фрагмент кода считывает данные из трех файлов CSV и автоматически интерпретирует столбцы даты как объекты datetime. Ну и print был применен для того, чтобы проверить, что он выведет, да и считывает ли данные в целом или нет. Результат:

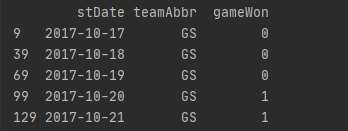


## Использование объекта ColumnDataSource

В приведенных выше примерах для представления данных использовались списки Python и массивы Numpy и Bokeh хорошо оснащено для обработки этих типов данных. Однако, когда дело доходит до данных в Python, вы, скорее всего, встретите словари Python и [Pandas DataFrames](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html" \l "pandas.DataFrame" \t "_blank), особенно если вы читаете данные из файла или внешнего источника данных. Bokeh хорошо оснащен для работы с этими более сложными структурами данных и даже имеет встроенные функции для их обработки, а именно ColumnDataSource. Вы можете спросить себя: «Зачем использовать ColumnDataSource, если Bokeh может напрямую взаимодействовать с другими типами данных?» Во-первых, ссылаетесь ли вы напрямую на список, массив, словарь или DataFrame, Bokeh за кулисами все равно превратит его в ColumnDataSource. Что еще более важно, ColumnDataSource значительно упрощает реализацию интерактивных возможностей Bokeh. ColumnDataSource лежит в основе передачи данных глифам, которые вы используете для визуализации.Его основная функция — отображать имена столбцам ваших данных. Это упрощает использование ссылок на элементы данных при построении визуализации. Это также упрощает то же самое для Bokeh при создании визуализации. Давайте начнем с визуализации гонки за первое место в Западной конференции НБА в 2017–2018 годах между действующим чемпионом Golden State Warriors и претендентом Houston Rockets. Ежедневные записи о победах и поражениях этих двух команд хранятся в DataFrame с именем west\_top\_2:

#ИСПОЛЬЗОВАНИЕ ОБЪЕКТА ColumnDataSource  
#Пример 1  
west\_top\_2 = (standings[(standings['teamAbbr'] == 'HOU') | (standings['teamAbbr'] == 'GS')]  
 .loc[:, ['stDate', 'teamAbbr', 'gameWon']]  
 .sort\_values(['teamAbbr','stDate']))  
print(west\_top\_2.head())

Результат:



## Отсюда вы можете загрузить этот DataFrame в два объекта ColumnDataSource и визуализировать гонку:

from bokeh.plotting import figure, show  
from bokeh.io import output\_file  
from bokeh.models import ColumnDataSource  
  
# Output to file  
output\_file('west-top-2-standings-race1.html',  
 title='Western Conference Top 2 Teams Wins Race')  
  
# Isolate the data for the Rockets and Warriors  
rockets\_data = west\_top\_2[west\_top\_2['teamAbbr'] == 'HOU']  
warriors\_data = west\_top\_2[west\_top\_2['teamAbbr'] == 'GS']  
  
# Create a ColumnDataSource object for each team  
rockets\_cds = ColumnDataSource(rockets\_data)  
warriors\_cds = ColumnDataSource(warriors\_data)  
  
# Create and configure the figure  
fig = figure(x\_axis\_type='datetime',  
 plot\_height=300, plot\_width=600,  
 title='Western Conference Top 2 Teams Wins Race, 2017-18',  
 x\_axis\_label='Date', y\_axis\_label='Wins',  
 toolbar\_location=None)  
  
# Render the race as step lines  
fig.step('stDate', 'gameWon',  
 color='#CE1141', legend='Rockets',  
 source=rockets\_cds)  
fig.step('stDate', 'gameWon',  
 color='#006BB6', legend='Warriors',  
 source=warriors\_cds)  
  
# Move the legend to the upper left corner  
fig.legend.location = 'top\_left'  
  
# Show the plot  
show(fig)

## Результат:

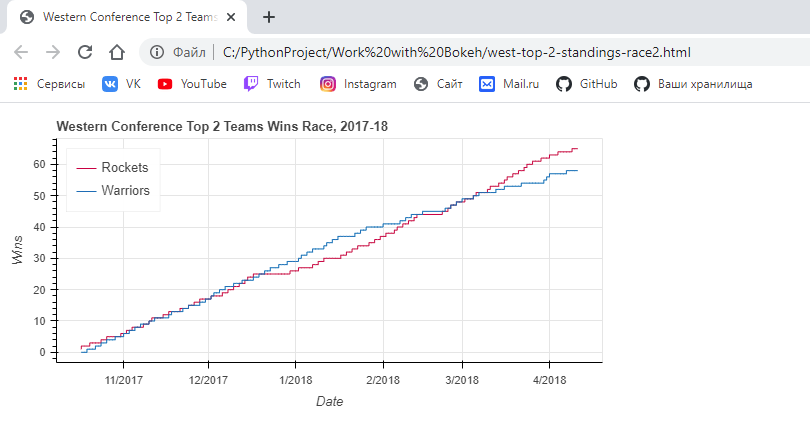
## 

Обратите внимание на ссылки на соответствующие объекты ColumnDataSource при создании двух строк. Вы просто передаете исходные имена столбцов в качестве входных параметров и указываете, какой ColumnDataSource использовать через свойство источника. Визуализация показывает напряженную гонку на протяжении всего сезона, Warriors создают довольно большую подушку в середине сезона. Тем не менее, небольшой спад в конце сезона позволил Rockets наверстать упущенное и в конечном итоге превзойти защищающихся чемпионов и завершить сезон посевным номером один в Западной конференции.

В предыдущем примере были созданы два объекта ColumnDataSource, по одному из подмножества фрейма данных west\_top\_2. В следующем примере будет воссоздан тот же вывод из одного ColumnDataSource на основе всего west\_top\_2 с использованием GroupFilter, который создает представление для данных:

#Пример 2  
from bokeh.plotting import figure, show  
from bokeh.io import output\_file  
from bokeh.models import ColumnDataSource, CDSView, GroupFilter  
  
# Вывод в файл  
output\_file('west-top-2-standings-race2.html',  
 title='Western Conference Top 2 Teams Wins Race')  
  
# Создать ColumnDataSource  
west\_cds = ColumnDataSource(west\_top\_2)  
  
# Создайте представления для каждой команды  
rockets\_view = CDSView(source=west\_cds,  
 filters=[GroupFilter(column\_name='teamAbbr', group='HOU')])  
warriors\_view = CDSView(source=west\_cds,  
 filters=[GroupFilter(column\_name='teamAbbr', group='GS')])  
  
# Создаем и настраиваем фигуру  
west\_fig = figure(x\_axis\_type='datetime',  
 plot\_height=300, plot\_width=600,  
 title='Western Conference Top 2 Teams Wins Race, 2017-18',  
 x\_axis\_label='Date', y\_axis\_label='Wins',  
 toolbar\_location=None)  
  
# Отрисовываем гонку в виде ступенчатых линий  
west\_fig.step('stDate', 'gameWon',  
 source=west\_cds, view=rockets\_view,  
 color='#CE1141', legend='Rockets')  
west\_fig.step('stDate', 'gameWon',  
 source=west\_cds, view=warriors\_view,  
 color='#006BB6', legend='Warriors')  
  
# Переместите легенду в верхний левый угол  
west\_fig.legend.location = 'top\_left'  
  
# Показать сюжет  
show(west\_fig)

Результат:



## Обратите внимание, как GroupFilter передается в CDSView в виде списка. Это позволяет вам комбинировать несколько фильтров вместе, чтобы при необходимости изолировать нужные данные от ColumnDataSource.

## Организация нескольких визуализаций с помощью макетов

В турнирной таблице Восточной конференции в Атлантическом дивизионе остались два соперника: Boston Celtics и Toronto Raptors. Прежде чем повторить шаги, использованные для создания west\_top\_2, давайте попробуем еще раз протестировать ColumnDataSource, используя то, что вы узнали выше. В этом примере вы увидите, как передать весь DataFrame в ColumnDataSource и создать представления для изоляции соответствующих данных:

#ОРГАНИЗАЦИЯ НЕСКОЛЬКИХ ВИЗУАЛИЗАЦИЙ С ПОМОЩЬЮ МАКЕТОВ  
from bokeh.plotting import figure, show  
from bokeh.io import output\_file  
from bokeh.models import ColumnDataSource, CDSView, GroupFilter  
  
# Вывод в файл  
output\_file('Организация\_нескольких\_визуализаций\_с\_помощью\_макетов.html',  
 title='Eastern Conference Top 2 Teams Wins Race')  
  
# Создать ColumnDataSource  
standings\_cds = ColumnDataSource(standings)  
  
# Create views for each team  
celtics\_view = CDSView(source=standings\_cds,  
 filters=[GroupFilter(column\_name='teamAbbr',  
 group='BOS')])  
raptors\_view = CDSView(source=standings\_cds,  
 filters=[GroupFilter(column\_name='teamAbbr',  
 group='TOR')])  
  
# Создаем и настраиваем фигуру  
east\_fig = figure(x\_axis\_type='datetime',  
 plot\_height=300, plot\_width=600,  
 title='Eastern Conference Top 2 Teams Wins Race, 2017-18',  
 x\_axis\_label='Date', y\_axis\_label='Wins',  
 toolbar\_location=None)  
  
# Отрисовываем гонку в виде ступенчатых линий  
east\_fig.step('stDate', 'gameWon',  
 color='#007A33', legend='Celtics',  
 source=standings\_cds, view=celtics\_view)  
east\_fig.step('stDate', 'gameWon',  
 color='#CE1141', legend='Raptors',  
 source=standings\_cds, view=raptors\_view)  
  
# Переместите легенду в верхний левый угол  
east\_fig.legend.location = 'top\_left'  
  
# Показать сюжет  
show(east\_fig)

## Результат:

## 

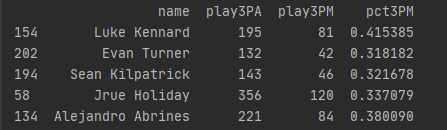
## Добавление взаимодействия

### **Выбор точек данных**

Реализовать поведение выбора так же просто, как добавить несколько конкретных ключевых слов при объявлении ваших глифов. В следующем примере создается диаграмма разброса, которая связывает общее количество попыток трехочковых бросков игроком с их процентом (для игроков, сделавших не менее 100 трехочковых бросков). Данные могут быть агрегированы из DataFrame player\_stats:

#ДОБАВЛЕНИЕ ВЗАИМОДЕЙСТВИЯ  
#ВЫБОР ТОЧЕК ДАННЫХ  
# Найдите игроков, которые сделали хотя бы 1 трехочковый бросок за сезон  
three\_takers = player\_stats[player\_stats['play3PA'] > 0]  
  
# Убрать имена игроков, поместив их в один столбец  
three\_takers['name'] = [f'{p["playFNm"]} {p["playLNm"]}'  
 for \_, p in three\_takers.iterrows()]  
  
# Суммируйте общее количество трехочковых попыток и количество попыток для каждого игрока  
three\_takers = (three\_takers.groupby('name')  
 .sum()  
 .loc[:,['play3PA', 'play3PM']]  
 .sort\_values('play3PA', ascending=False))  
  
# Отфильтровать всех, кто не сделал хотя бы 100 трехочковых бросков  
three\_takers = three\_takers[three\_takers['play3PA'] >= 100].reset\_index()  
  
# Добавить столбец с рассчитанным трехбалльным процентом (выполнено / предпринято)  
three\_takers['pct3PM'] = three\_takers['play3PM'] / three\_takers['play3PA']  
print(three\_takers.sample(5))

Результат:



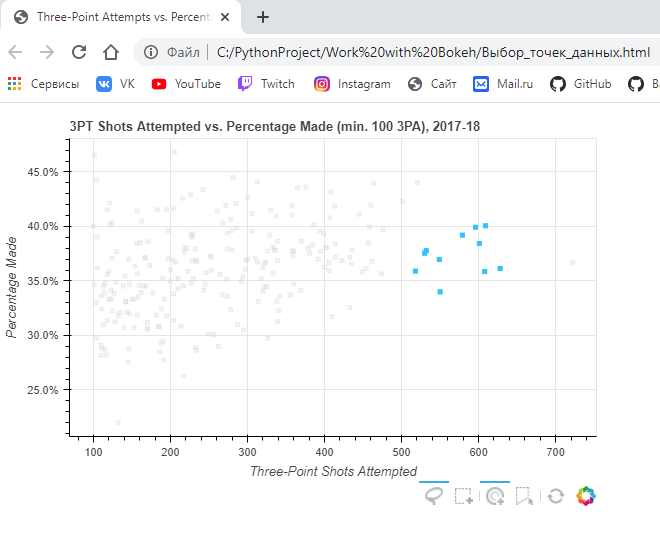
Допустим, вы хотите выбрать группы игроков в раздаче и при этом отключить цвет глифов, представляющих невыбранных игроков:

# Выбор группы игроков в раздаче из полученного DataFrame, при этом отключить цвет глифов  
from bokeh.plotting import figure, show  
from bokeh.io import output\_file  
from bokeh.models import ColumnDataSource, NumeralTickFormatter  
  
# Файл вывода  
output\_file('Выбор\_точек\_данных.html',  
 title='Three-Point Attempts vs. Percentage')  
  
# Сохраним данные в ColumnDataSource  
three\_takers\_cds = ColumnDataSource(three\_takers)  
  
# Укажите инструменты выбора, которые будут доступны  
select\_tools = ['box\_select', 'lasso\_select', 'poly\_select', 'tap', 'reset']  
  
# Создание картинки  
fig = figure(plot\_height=400,  
 plot\_width=600,  
 x\_axis\_label='Three-Point Shots Attempted',  
 y\_axis\_label='Percentage Made',  
 title='3PT Shots Attempted vs. Percentage Made (min. 100 3PA), 2017-18',  
 toolbar\_location='below',  
 tools=select\_tools)  
  
# Отформатируйте метки делений оси Y в процентах  
fig.yaxis[0].formatter = NumeralTickFormatter(format='00.0%')  
  
# Добавить квадрат, представляющий каждого игрока  
fig.square(x='play3PA',  
 y='pct3PM',  
 source=three\_takers\_cds,  
 color='royalblue',  
 selection\_color='deepskyblue',  
 nonselection\_color='lightgray',  
 nonselection\_alpha=0.3)  
  
# Показать результат  
show(fig)

Сначала укажите инструменты выбора, которые вы хотите сделать доступными. В приведенном выше примере «box\_select», «lasso\_select», «poly\_select» и «tap» (плюс кнопка сброса) были указаны в списке под названием select\_tools. Когда фигура создается, панель инструментов располагается «below» («ниже») графика, и список передается в инструменты, чтобы сделать инструменты, выбранные выше, доступными. Каждый игрок изначально представлен синим квадратным глифом, но при выборе игрока или группы игроков устанавливаются следующие конфигурации:

* Превратите выбранных игроков в темно-синий
* Измените глифы всех невыделенных игроков на светло-серый цвет с непрозрачностью 0,3

Результат:



### **Добавление действий при наведении курсора**

Таким образом, реализована возможность выбора конкретных точек данных игроков, которые кажутся интересными на моем графике разброса, но что, если вы хотите быстро увидеть, каких отдельных игроков представляет глиф? Один из вариантов — использовать Bokeh HoverTool (), чтобы показывать всплывающую подсказку, когда курсор пересекает пути с глифом. Все, что вам нужно сделать, это добавить в приведенный выше фрагмент кода следующее:

#ДОБАВЛЕНИЕ ДЕЙСТВИЙ ПРИ НАВЕДЕНИИ КУРСОРА  
from bokeh.models import HoverTool  
  
output\_file('Добавление\_действий\_при\_наведении\_курсора.html',  
 title='Three-Point Attempts vs. Percentage')  
  
# Отформатируйте всплывающую подсказку  
# Отформатируйте всплывающую подсказку  
tooltips = [  
 ('Player','@name'),  
 ('Three-Pointers Made', '@play3PM'),  
 ('Three-Pointers Attempted', '@play3PA'),  
 ('Three-Point Percentage','@pct3PM{00.0%}'),  
 ]  
  
# Настроить рендерер, который будет использоваться при наведении  
hover\_glyph = fig.circle(x='play3PA', y='pct3PM', source=three\_takers\_cds,  
 size=15, alpha=0,  
 hover\_fill\_color='black', hover\_alpha=0.5)  
  
# Добавляем HoverTool к фигуре  
fig.add\_tools(HoverTool(tooltips=tooltips, renderers=[hover\_glyph]))  
  
# Показать  
show(fig)

HoverTool() немного отличается от инструментов выбора, которые вы видели выше, тем, что у него есть свойства, в частности всплывающие подсказки. Во-первых, вы можете настроить отформатированную всплывающую подсказку, создав список кортежей, содержащих описание и ссылку на ColumnDataSource. Этот список был передан в качестве входных данных в HoverTool(), а затем просто добавлен к рисунку с помощью add\_tools(). Для большего подчеркивания игроков при наведении, был создан новый глиф, в данном случае кружков вместо квадратов, и присвоения ему hover\_glyph. Обратите внимание, что начальная непрозрачность установлена на ноль, поэтому она невидима, пока курсор не коснется ее. Свойства, которые появляются при наведении курсора, фиксируются путем установки hover\_alpha значения 0,5 вместе с hover\_fill\_color. Теперь вы увидите, как маленький черный кружок появляется над исходным квадратом при наведении курсора на различные маркеры. Вот что произошло:

## 

### **Выделение данных с помощью легенды**

В этом примере вы увидите две идентичные диаграммы разброса, на которых сравниваются игровые очки и подборы Леброна Джеймса и Кевина Дюранта. Единственная разница будет заключаться в том, что один будет использовать hide в качестве своей click\_policy, а другой — mute. Первый шаг — настроить вывод и настроить данные, создав представление для каждого игрока из кадра данных player\_stats:

#ВЫДЕЛЕНИЕ ДАННЫХ С ПОМОЩЬЮ ЛЕГЕНДЫ  
from bokeh.plotting import figure, show  
from bokeh.io import output\_file  
from bokeh.models import ColumnDataSource, CDSView, GroupFilter  
from bokeh.layouts import row  
  
# Вывести в записной книжке  
output\_file('lebron-vs-durant.html',  
 title='LeBron James vs. Kevin Durant')  
  
# Хранить данные в ColumnDataSource  
player\_gm\_stats = ColumnDataSource(player\_stats)  
  
# Создайте представление для каждого игрока  
lebron\_filters = [GroupFilter(column\_name='playFNm', group='LeBron'),  
 GroupFilter(column\_name='playLNm', group='James')]  
lebron\_view = CDSView(source=player\_gm\_stats,  
 filters=lebron\_filters)  
  
durant\_filters = [GroupFilter(column\_name='playFNm', group='Kevin'),  
 GroupFilter(column\_name='playLNm', group='Durant')]  
durant\_view = CDSView(source=player\_gm\_stats,  
 filters=durant\_filters)

Перед созданием рисунков общие параметры рисунка, маркеры и данные могут быть объединены в словари и повторно использованы. Это не только сохраняет избыточность на следующем шаге, но и предоставляет простой способ настроить эти параметры позже, если потребуется:

# Объединить общие аргументы ключевых слов в dicts  
common\_figure\_kwargs = {  
 'plot\_width': 400,  
 'x\_axis\_label': 'Points',  
 'toolbar\_location': None,  
}  
common\_circle\_kwargs = {  
 'x': 'playPTS',  
 'y': 'playTRB',  
 'source': player\_gm\_stats,  
 'size': 12,  
 'alpha': 0.7,  
}  
common\_lebron\_kwargs = {  
 'view': lebron\_view,  
 'color': '#002859',  
 'legend': 'LeBron James'  
}  
common\_durant\_kwargs = {  
 'view': durant\_view,  
 'color': '#FFC324',  
 'legend': 'Kevin Durant'  
}

Теперь, когда различные свойства установлены, два графика разброса можно построить гораздо более кратко:

# Создайте две фигуры и нарисуйте данные  
hide\_fig = figure(\*\*common\_figure\_kwargs,  
 title='Click Legend to HIDE Data',  
 y\_axis\_label='Rebounds')  
hide\_fig.circle(\*\*common\_circle\_kwargs, \*\*common\_lebron\_kwargs)  
hide\_fig.circle(\*\*common\_circle\_kwargs, \*\*common\_durant\_kwargs)  
  
mute\_fig = figure(\*\*common\_figure\_kwargs, title='Click Legend to MUTE Data')  
mute\_fig.circle(\*\*common\_circle\_kwargs, \*\*common\_lebron\_kwargs,  
 muted\_alpha=0.1)  
mute\_fig.circle(\*\*common\_circle\_kwargs, \*\*common\_durant\_kwargs,  
 muted\_alpha=0.1)

Обратите внимание, что у mute\_fig есть дополнительный параметр под названием muted\_alpha. Этот параметр управляет непрозрачностью маркеров, когда отключение звука используется как click\_policy. Наконец, устанавливается click\_policy для каждой фигуры, и они отображаются в горизонтальной конфигурации:

# Добавляем интерактивности в легенду  
hide\_fig.legend.click\_policy = 'hide'  
mute\_fig.legend.click\_policy = 'mute'  
  
# Визуализировать  
show(row(hide\_fig, mute\_fig))

Результат:



После того, как легенда размещена, все, что вам нужно сделать, это назначить либо скрыть, либо отключить для свойства фигуры click\_policy. Это автоматически превратит вашу базовую легенду в интерактивную легенду. Также обратите внимание, что специально для отключения звука дополнительное свойство muted\_alpha было установлено в соответствующих круговых глифах для Леброна Джеймса и Кевина Дюранта. Это определяет визуальный эффект, обусловленный взаимодействием легенды.