UFC Fighter Performance Metric Exploration

I wrestled back in highschool, and since then I kept up with UFC since I think it's very entertaining at the high level. I haven't explored the data provided before, so this is my first time doing so. I dove into the data set (link) provided and we can see trends based on fighters, their wins and losses, different divisions, and the method of winning or losing.

Technique: Scatterplot

I opted for the scatterplot since it made the most sense with this data. Using something like a bar chart will get pretty cluttered really fast, especially on the initialization of the plot. Using a line plot was also an option, but when selecting wins or loss or both, it didn't really make sense to me at the time of constructing this visualization.

Scatterplots plot an X vs. Y relationship where each data point is a unique bout in this case. On initialization you may realize that some points may overlay on each other, which means that the fights took place on the same day and the fighters had similar performance metrics in their fights. In this visualization the X-axis will always be the date since it didn't make sense to use any other data type since the goal is show fighter improvement/comparison. The Y-axis will be the up to the user to select

Library: Bokeh

Bokeh is an open source library for creating interactive Java-Script powered visualizations in web browswers from simple plots, like scatter plots, to dashboards which can be found in their demo tab on their website link. Bokeh is maintained on GitHub and is recommended if a user finds a bug to report it to the GitHub for developers to fix. website

I looked at other alternatives, mainly Plotly and Altair/Vega, but I chose this one since I wanted experience with a new library I've never seen before, and I would've liked to make it interactive. I didn't choose Altair/Vega due to my previous experience with in is 522. Another plus that I saw to Bokeh is that I could practice JavaScript/CSS, but I only used it in one line in this project. In future projects, I think a more complex dashboard would require more of these languages.

In this project I really only used bokeh.circle, which does as one would expect, which would be draw circles on a grid, making a scatter plot. There is a caveat though, this is a low level version of the bokeh.scatter, but offers more customizability by changing the shape, and we can pass through arguments for different colors on different data points.

Install Bokeh

Pip:

> $ pip install bokeh

Conda:

> $ conda install bokeh

Import Libraries and Dependencies

In [1]:
```
import pandas as pd
from bokeh.plotting import figure, output_file, show
from bokeh.layouts import column, row
from bokeh.models.tools import HoverTool
from bokeh.models.widgets import Select
from bokeh.models import (ColumnDataSource, RadioButtonGroup, GroupFilter, Column, Row, Div, HoverTool, MultiChoice)
from bokeh.io import curdoc, output_notebook
from bokeh.application.handlers.function import FunctionHandler
from bokeh.application import Application
from bokeh.resources import INLINE
```

Data Manipulation and Collection

The data is from the following link, and is a pretty clean file already. All we have to do is read, turn the dates into pandas datetime, make a colors column, map wins and losses, and fix time formats. All we have to do it go to the page and click download. After that a zip file will be downloaded. Extract the files and notice that one is a very large file, we don't care about that so you can go ahead and delete it. We only care about the CSV labeled masterdataframe.csv.

We need a colors column due to the information we are going to be passing into bokeh.circle in the future. We want to differentiate between wins and losses of fighters, otherwise it all be 1 homogenous color. The data frame contains a 0 for losing and a 1 for winning, we should map this to text since we want our legent to be readable and not a 0 and 1. Time formats are based on the current UFC ruling where main events and championship fights are 5 rounds, and all other are 3 rounds. Before the UFC adopted this rule there were a multitude of different time formats, some of them unsafe like a no time limit. I chose to omit all other time formats that are not 3 or 5 rounds due to transfering this knowledge into the modern day and not some arbitrary time.

In [2]:
```
def get_dataframe() -> pd.DataFrame:
    """Gets the data we will be using and cleans it for use"""

    data = pd.read_csv('masterdataframe.csv')
    data['date'] = pd.to_datetime(data['date'])
    data['colors'] = ['#FF3131' if category == 0 else '#1F51FF' for category in data['result']]
    data['result'].replace({0:'Loss', 1:'Win'}, inplace=True)
    data = data[data['time_format'].isin(['5-5-5-5-5', '5-5-5'])]

    return data
```

In [3]:
```
data = get_dataframe()
data
```

Out[3]:

	date	fight_url	event_url	result	fighter	opponent	division	stance	dob	method	...	precomp_recent_avg_clinch_strikes_attempts_per_min	avg_grou
364	1999-07-16	http://ufcstats.com/fight-details/693e4a0bed9a...	http://ufcstats.com/event-details/1a1a4d7a2904...	Loss	Royce Alger	Eugene Jackson	Middleweight	Orthodox	NaN	KO/TKO	...	NaN	
365	1999-07-16	http://ufcstats.com/fight-details/693e4a0bed9a...	http://ufcstats.com/event-details/1a1a4d7a2904...	Win	Eugene Jackson	Royce Alger	Middleweight	Orthodox	1966-09-23	KO/TKO	...	NaN	
366	1999-07-16	http://ufcstats.com/fight-details/3e4199f9a69b...	http://ufcstats.com/event-details/1a1a4d7a2904...	Loss	Tim Lajcik	Tsuyoshi Kohsaka	Heavyweight	Orthodox	1965-06-21	KO/TKO	...	NaN	
367	1999-07-16	http://ufcstats.com/fight-details/3e4199f9a69b...	http://ufcstats.com/event-details/1a1a4d7a2904...	Win	Tsuyoshi Kohsaka	Tim Lajcik	Heavyweight	Orthodox	1970-03-06	KO/TKO	...	0.113450	
368	1999-07-16	http://ufcstats.com/fight-details/82cf32aea3d7...	http://ufcstats.com/event-details/1a1a4d7a2904...	Loss	Flavio Luiz Moura	Paul Jones	Middleweight	Open Stance	NaN	SUB	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
13317	2022-06-25	http://ufcstats.com/fight-details/c57c8e22a3e8...	http://ufcstats.com/event-details/eb42d4febfaf...	Win	Josh Parisian	Alan Baudot	Heavyweight	Orthodox	1989-06-28	KO/TKO	...	1.812958	
13318	2022-06-25	http://ufcstats.com/fight-details/3c98739eb42f...	http://ufcstats.com/event-details/eb42d4febfaf...	Loss	Neil Magny	Shavkat Rakhmonov	Welterweight	Orthodox	1987-08-03	SUB	...	0.600000	
13319	2022-06-25	http://ufcstats.com/fight-details/3c98739eb42f...	http://ufcstats.com/event-details/eb42d4febfaf...	Win	Shavkat Rakhmonov	Neil Magny	Welterweight	Orthodox	1994-10-23	SUB	...	0.614618	
13320	2022-06-25	http://ufcstats.com/fight-details/1f5f59924b59...	http://ufcstats.com/event-details/eb42d4febfaf...	Loss	Arman Tsarukyan	Mateusz Gamrot	Lightweight	Orthodox	1996-10-11	U-DEC	...	0.177778	
13321	2022-06-25	http://ufcstats.com/fight-details/1f5f59924b59...	http://ufcstats.com/event-details/eb42d4febfaf...	Win	Mateusz Gamrot	Arman Tsarukyan	Lightweight	Southpaw	1990-12-11	U-DEC	...	0.314136	

12886 rows × 531 columns

You can see that we have 12886 data points with a whopping 531 columns! We don't care about a majority of these columns since the raw numbers are in the dataframe, like number of significant strikes. Said columns are some sort of aggregation the the raw columns, for example, significant strike accuracy is significant strikes landed divided by significant strikes attempted. We also have urls, but those just go to a webpage containing the metrics, so we are also not going to use this either.

Plotting the Data

First I'm going to show the method of plotting using bokeh.circle, point out the room for improvement, then give a step by step on how to make the interactive plot. So Bokeh is declaritive so we must be pretty verbose with what we want.

Our first step is to output to the notebook, which tells the library we are using Jupyter Notebooks, so the plot must be inline.

Next we have to make a figure on which we will plot, this is similar to MatplotLib where we plot on figures. We must declare our x-axis is a datetime, otherwise Bokeh will make some nasty numbers that don't make any sense.

We have to initialize a ColumnDataSource object so that Bokeh can easily read data points and plot. This comes in handy later when we want to make the plot interactive.

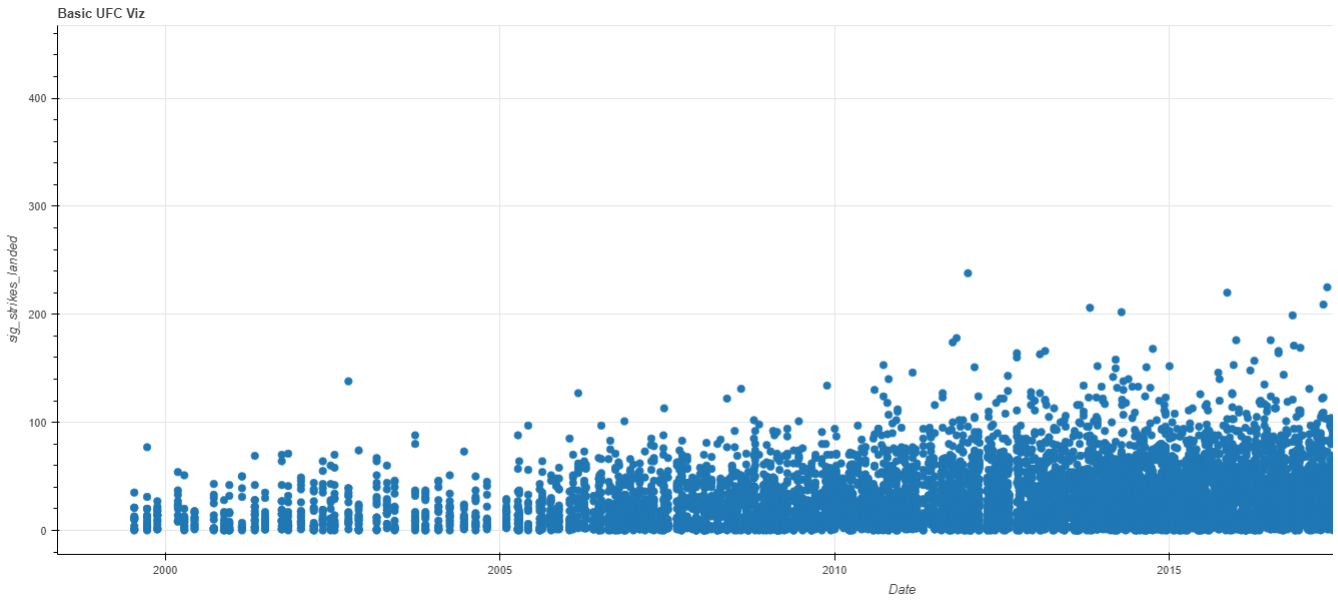
Lastly we plot. This is similar to Altair where we have to declare the columns for x and y axes, then declare our source.

We use curdoc for handling, and then we show using show(p)

In [4]:
```
# list(data.columns)
```

In [5]:
```
output_notebook()
p = figure(height=600, title="Basic UFC Viz", sizing_mode="stretch_width", x_axis_type='datetime')
p.xaxis.axis_label = "Date"
p.yaxis.axis_label = "sig_strikes_landed"
source = ColumnDataSource(data)
p.circle(x="date", y="sig_strikes_landed", source=source, size=7)
curdoc()
show(p)
```

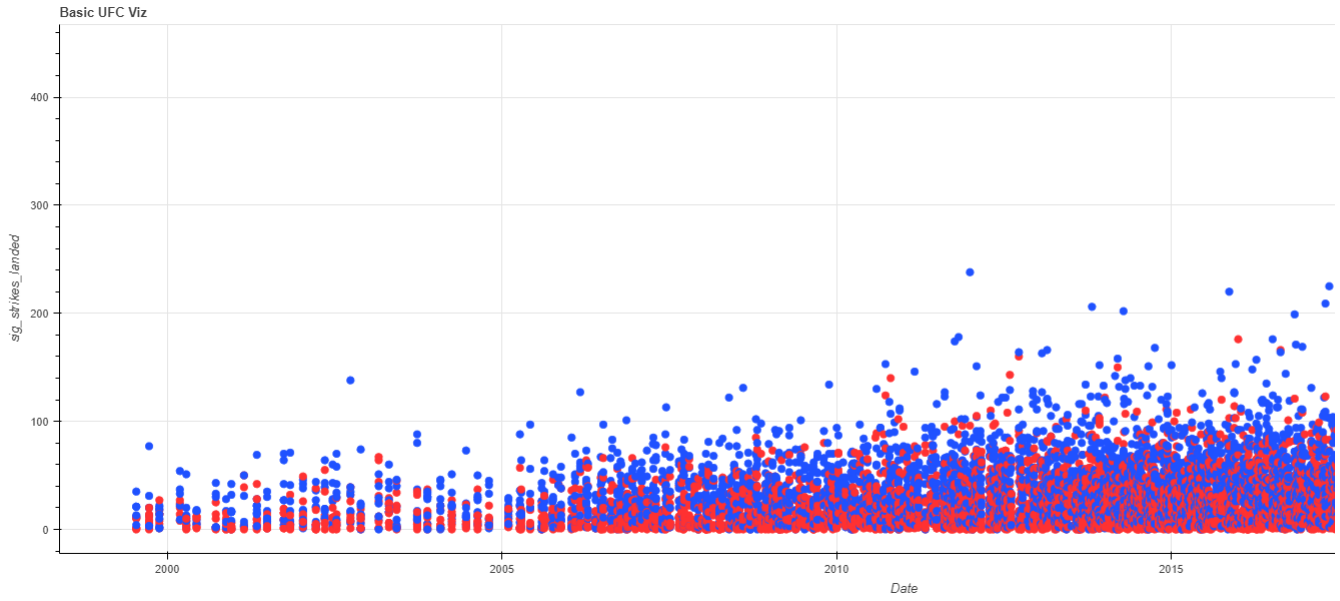

BokehJS 3.3.0 successfully loaded.



Nice, but we didn't include the colors, lets add that and see what happens with a legend field and see what happens.

In [6]:
```
output_notebook()
p = figure(height=600, title="Basic UFC Viz", sizing_mode="stretch_width", x_axis_type='datetime')
p.xaxis.axis_label = "Date"
p.yaxis.axis_label = "sig_strikes_landed"
source = ColumnDataSource(data)
p.circle(x="date", y="sig_strikes_landed", source=source, size=7, color="colors", legend_field='result')
curdoc()
show(p)
```

BokehJS 3.3.0 successfully loaded.

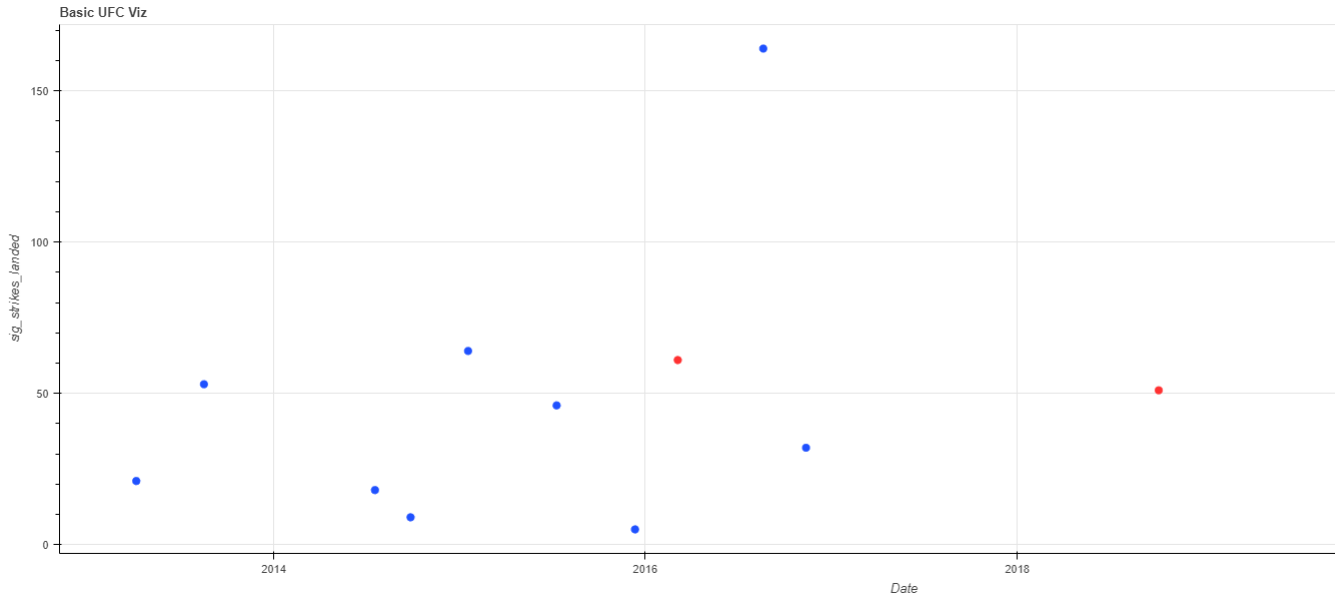


So this doesn't really tell us much. What if we look at one particular fighter?

Lets look at Connor McGregor

In [7]:
```
output_notebook()
p = figure(height=600, title="Basic UFC Viz", sizing_mode="stretch_width", x_axis_type='datetime')
p.xaxis.axis_label = "Date"
p.yaxis.axis_label = "sig_strikes_landed"
source = ColumnDataSource(data[data["fighter"] == "Conor McGregor"])
p.circle(x="date", y="sig_strikes_landed", source=source, size=7, color="colors", legend_field='result')
curdoc()
show(p)
```

Loading BokehJS ...



The Interactive Plot

So this is where I got the idea for an interactive visualization. I'd like to see different categories of fighters with certain criteria on how they won or lost. I'd like to filter based on fighters, metrics, methods of winning/losing, division/weight class, and if the fighter wins or loses.

Widgets

First we need to get our choice for our widgets.

The list of fighters, methods, and divisions are the most intuitive where it's the unique value in each column.

The metrics we can choose from are only available in a certain subset of columns, so we use slicing techniques to get them.

I didn't think anyone would want to see underscores in the choices for metrics, so I made a dictionary to map choices onto the column. We will split the string using the underscores, then when we join them we will capitalize each word once again.

In [8]:
```
# vars
fighter_list = sorted(list(data['fighter'].unique()))
stats = list(data.columns[21:42])  # we don't care about accuracies, everything else is some sort of aggregation of these columns
method_options = list(data['method'].unique())
division_list = list(data['division'].unique())
stat_map = {}
for stat in stats:
    stat_map[" ".join(word.capitalize() for word in stat.split("_"))] = stat
```

Next we have to make the widgets themselves.

Our title is a div element in HTML, since this is a web based plotting library. If you don't know what a div is, it's just a division of a website. You can think of it like the cells in a notebook, but on a webpage they are much more compartmentalized. Think of the different parts of your Google search, you may be able to draw out different segments of the web pages and those could be called divs.

Multichoice allows users to select or type of the fighter they want to see. A dropdown didn't make much sense here since there are 500+ fighters to choose from, so we should type.

The metrics choice is a dropdown since it's much more manageable to see what we can choose. The same goes for method and division.

Lastly we have the choice of winning, lossing, or all. This comes in the form of a radio button. Radio buttons give the users a choice where only one element maybe selected. If given the choice apples, oranges, and bananas, I must choose apples, oranges, or bananas.

In [9]:
```
# make input objects
title = Div(text='<h1 style="text-align: center">UFC Interactive Scatterplot Viz</h1>')
fighter_button = MultiChoice(title="Fighter(s)", options=fighter_list, option_limit=3, width=250)
stats_axis = Select(title="Y-Axis", options=sorted(stat_map.keys()), value='Sig Strikes Landed')
method = Select(title="Method", options=["Any"] + method_options, value="Any")
division = Select(title="Division", options=["All"] + division_list, value="All")
win_or_loss = RadioButtonGroup(labels=["Any", "Wins", "Losses"], active=0, styles={"padding-top": "20px"})

TOOLTIPS = [
        ('Fighter', '@fighter'),
        ('Opponent', '@opponent'),
        ('Result', '@result'),
        ('Method', '@method'),
        ('Date', '@date'),
        ('Round', '@round'),
        ('Time', '@time')
        ]
```

Initialization

We have an empty source since we will be adding to it. These different columns in our source will be used for our tool tips.

Creation of the figure and plot are similar to before.

In [10]:
```
source = ColumnDataSource(data=dict(date=[], y=[], color=[], fighter=[], opponent=[], result=[], method=[], round=[], time=[]))
```

In [11]:
```
output_notebook()
p = figure(height=600, title="", toolbar_location=None, sizing_mode="stretch_width", x_axis_type='datetime')
p.add_tools(HoverTool(tooltips=TOOLTIPS))
p.circle(x="date", y="y", source=source, size=7, color="color", legend_field='result')
```

BokehJS 3.3.0 successfully loaded.

Out[11]: **GlyphRenderer**(id = 'p2772', ...)

Selecting Points and Updating

Having a select function will allow use to easily put repeatable code into one line. This makes our update function more readable. This essentially tells our widgets to filter our data. For the **MultiChoice** widget, we have a value attribute which output in the form of a list. In our **Select** widgets, we also have value, and when they are set to Any or All, there is not filtering. Lastly is the **radio button** where the active is the number in the list. Since we initialized the list as (all, win, loss) the buttons value will return and index on where the pressed button is. When all is selected the value is 0 and there is no filtering.

Update just updates the chart and source data based on the selected points. Notice that the Y-axis title will always change when the new metric is chosen.

In [12]:
```
def select_points() -> pd.DataFrame():
    if fighter_button.value:
        df = data[data['fighter'].isin(fighter_button.value)] # in a select, but we only are selecting fighters
    else:
        df = data

    if method.value != "Any":
        df = df[df['method'] == method.value]

    if division.value != "All":
        df = df[df['division'] == division.value]

    if win_or_loss.active == 1:
        df = df[df['result'] == 'Win']

    if win_or_loss.active == 2:
        df = df[df['result'] == 'Loss']

    return df
```

In [13]:
```
def update():
    df = select_points()
    x_name = "Date"
    y_name = stat_map[stats_axis.value]

    p.xaxis.axis_label = "Date"
    p.yaxis.axis_label = stats_axis.value

    if fighter_button.value:
        p.title.text = f"{len(df)} Bouts of {stats_axis.value} by " + " ".join(fighter for fighter in fighter_button.value) + f" by {method.value} in {division.value} division"
    else:
```

```
        p.title.text = f"{len(df)} Bouts of {stats_axis.value} by all Fighters by {method.value} in {division.value} division"
    source.data = dict(
        date=df['date'],
        y=df[y_name],
        color=df['colors'],
        fighter=df['fighter'],
        opponent=df['opponent'],
        result=df['result'],
        method=df['method'],
        round=df['round'],
        time=df['time']
    )
```

Final Steps

Lastly we have to set the widgets to actually work and be responsive, so we must connect them to the update function. We do this in the following cell below.

We are going to put the widgets into a row where each is horizontal to each other. The line of code can be read as literal left to right in the order we set the controls. Update is there for initialization, and our final layout will be a column where it is also a literal from top to bottom.

The modify doc function allows us to look at the visualization inline after passing through a handler, then we pass through the Application object to show the visualization.

In [14]:
```
controls = [fighter_button, stats_axis, method, division]
for control in controls:
    control.on_change('value', lambda attr, old, new: update())
win_or_loss.on_change('active', lambda attr, old, new: update())
```

In [15]:
```
inputs = row(*controls, win_or_loss,  width=320)
update()
layout = column(title, p, inputs, sizing_mode="stretch_width", height=800)
```

In [16]:
```
def modify_doc(doc):
    doc.add_root(row(layout, width=800))
    doc.title = "test"
```

In [17]:
```
handler = FunctionHandler(modify_doc)
app = Application(handler)
show(app)
```

In [ ]:

In [ ]: