Introduction

Google Trends gives us an estimate of search volume. We can explore if search popularity relates to other kinds of data. Perhaps there are patterns in Google's search volume and the price of Bitcoin or a hot stock like Tesla. Perhaps search volume for the term "Unemployment Benefits" can tell us something about the actual unemployment rate?

Data Sources:

- Unemployment Rate from FRED
- Google Trends
- Yahoo Finance for Tesla Stock Price
- Yahoo Finance for Bitcoin Stock Price

Import Statements

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

Reading the Data

In [2]:
```
df_tesla = pd.read_csv('TESLA Search Trend vs Price.csv')
df_btc_search = pd.read_csv('Bitcoin Search Trend.csv')
df_btc_price = pd.read_csv('Daily Bitcoin Price.csv')
df_unemployment = pd.read_csv('UE Benefits Search vs UE Rate 2004-19.csv')
```

Data Exploration

Tesla

Lets take a look at the Tesla dataframe that we created

In [3]:
```
df_tesla
```

Out[3]:

	MONTH	TSLA_WEB_SEARCH	TSLA_USD_CLOSE
0	2010-06-01	3	4.766000
1	2010-07-01	3	3.988000
2	2010-08-01	2	3.896000
3	2010-09-01	2	4.082000
4	2010-10-01	2	4.368000
...	...	...	...
119	2020-05-01	16	167.000000
120	2020-06-01	17	215.962006
121	2020-07-01	24	286.152008
122	2020-08-01	23	498.320007
123	2020-09-01	31	407.339996

124 rows × 3 columns

From the table, we can see different types of information like the months that this data is based on, Tesla's web search index, and Tesla's closing price at the end of each respective month. Now that we know what our dataframe looks like, let's move on to the next one.

Unemployment Data

Let's look at the unemployment data that was obtained

In [4]:
```
df_unemployment
```

Out[4]:

	MONTH	UE_BENEFITS_WEB_SEARCH	UNRATE
0	2004-01	34	5.7
1	2004-02	33	5.6
2	2004-03	25	5.8
3	2004-04	29	5.6
4	2004-05	23	5.6
...	...	...	...
176	2018-09	14	3.7
177	2018-10	15	3.8
178	2018-11	16	3.7
179	2018-12	17	3.9
180	2019-01	21	4.0

181 rows × 3 columns

Again, we can see the months, search index, and unemployment rate in the dataframe.

Bitcoin

Here is the infamous Bitcoin. Let's first look at our search data, then at our price data

In [5]:
```
display(df_btc_search, df_btc_price)
```

	MONTH	BTC_NEWS_SEARCH
0	2014-09	5
1	2014-10	4
2	2014-11	4
3	2014-12	4
4	2015-01	5
...	...	...
68	2020-05	22
69	2020-06	13
70	2020-07	14
71	2020-08	16
72	2020-09	13

73 rows × 2 columns

	DATE	CLOSE	VOLUME
0	2014-09-17	457.334015	2.105680e+07
1	2014-09-18	424.440002	3.448320e+07
2	2014-09-19	394.795990	3.791970e+07
3	2014-09-20	408.903992	3.686360e+07
4	2014-09-21	398.821014	2.658010e+07
...	...	...	...
2199	2020-09-24	10745.548828	2.301754e+10
2200	2020-09-25	10702.290039	2.123255e+10
2201	2020-09-26	10754.437500	1.810501e+10
2202	2020-09-27	10774.426758	1.801688e+10
2203	2020-09-28	10912.536133	2.122653e+10

2204 rows × 3 columns

These two dataframes provide us with various data like search index, closing price, and volume. Keep in mind that the indices for both dataframes are different.

We can see from our DataFrames that Google's search interest ranges between 0 and 100. Specifically, the number represents search interest relative to the highest point. A value of 100 is the peak popularity for the search term, while a value of 50 means that the search term is half as popular compared to its peak, and even lower numbers represent weak search popularity compared to peak results. This is a scaled, normalized version of aggregated search volume data - Interesting right?

Okay let's continue. After briefly examining all of our dataframes, we will do some work on cleaning up this data.

Data Cleaning

Checking for Missing Values

Using the pandas library, we can use the `.isNull()` function to check if there are any missing values in the dataframe and returns a boolean. Specifically the function checks if there are any `None` or `NaN` or even `NaT` values. Because we are performing this operation on a DataFrame object, we don't want to check every individual value to see if return value is valid. Instead, we can use pandas `.values` to convert the dataframe into an array and then chain it together with the `.any()` method, which returns a clean True or False boolean value.

You might be asking, why do we need to chain so many different methods together? Well, if we had instead decided to do `DataFrame.isnull().any()` or `DataFrame.isnull()`, neither of the results would be clean enough for our needs. Using the first option, the result is a 3x2 table where the rows represent the values of the previous DF columns and their resulting boolean returns. With the second option, we would receive the original dataframe, but the resulting data values would be boolean. No, instead we want to quickly know if there are any invalid or missing values in the original dataframe through a single `True` or `False` result. If the result is True, then we can further investigate, otherwise we should probably just move on.

Like in mathematical operations, order matters! As we have learned from this little investigation, we can't just chain random functions together that we think will work. Instead, we have to take a more methodical approach and combine functions in a more meaningful way. I quickly discovered this to be particularly true when working with data using Pandas library...

In [6]:
```
print(f'Missing values for Tesla?: {df_tesla.isnull().values.any()}')
print(f'Missing values for U/E?: {df_unemployment.isnull().values.any()}')
print(f'Missing values for BTC Search?: {df_btc_search.isnull().values.any()}')
```

```
Missing values for Tesla?: False
Missing values for U/E?: False
Missing values for BTC Search?: False
```

Hey look at that, no missing values in our each of our above three DataFrames!

Unfortunately, if we perform the operation on our last DF, we do indeed see that there is at least one missing or invalid value:

In [7]:
```
print(f'Missing values for BTC price?: {df_btc_price.isnull().values.any()}')
```

```
Missing values for BTC price?: True
```

Luckily, there is an easy way for us to check how many missing values are present by using the `.count()` method:

In [8]:
```
print(f'Number of missing values: {df_btc_price.CLOSE.isnull().count()}')
```

```
Number of missing values: 2204
```

Further, we can remove all of the missing values by using the `.dropna()` function which returns a new Series with the removed data. However, using the argument `inplace=True` lets us replace the old DataFrame with the new one instead of creating a who new data structure.

In [9]:
```
df_btc_price.dropna(inplace=True)
```

Now our all of our data is clean, nice!

Convert Strings to DateTime Objects

To prepare our data visualization charts, we should convert the `DATE` and `MONTH` columns in our DataFrames to DateTime objects. This way, Matplotlib will be able to read the dates and properly output a beautiful graph that we can appreciate.

Simply, we just have to use Pandas `to_datetime()` method on the column that we want to convert, and then assign this result to itself (replacing the original column):

In [10]:
```
df_btc_price.DATE = pd.to_datetime(df_btc_price.DATE)
df_tesla.MONTH = pd.to_datetime(df_tesla.MONTH)
df_unemployment.MONTH = pd.to_datetime(df_unemployment.MONTH)
df_btc_search.MONTH = pd.to_datetime(df_btc_search.MONTH)
```

Nice! Looks like our data type conversion worked. Let's take a look at one of our DataFrames. On the surface, it looks like nothing has changed, but date values themselves are now DateTime objects that we can use!

In [11]:
```
display(df_btc_price)
```

	DATE	CLOSE	VOLUME
0	2014-09-17	457.334015	2.105680e+07
1	2014-09-18	424.440002	3.448320e+07
2	2014-09-19	394.795990	3.791970e+07
3	2014-09-20	408.903992	3.686360e+07
4	2014-09-21	398.821014	2.658010e+07
...	...	...	...
2199	2020-09-24	10745.548828	2.301754e+10
2200	2020-09-25	10702.290039	2.123255e+10
2201	2020-09-26	10754.437500	1.810501e+10
2202	2020-09-27	10774.426758	1.801688e+10
2203	2020-09-28	10912.536133	2.122653e+10

2203 rows × 3 columns

Converting from Daily to Monthly Data

What we have now are a bunch of daily values. Instead, I think a larger timeframe between data values will prove to be more beneficial to us. It allows us to step back and look at the bigger picture. Additionally, any small outliers or noise in the data will be muted out, which allows us to paint a smoother graph.

So, lets use the `.resample()` function to correctly turn our daily data into monthly data. For our first argument, we have "M", this tells the function that we want to convert the timeframe into months. Next we have `on="DATE"` which tells the function that the column that we want to change is the DATE column. The `.mean()` function is chained at the end, so that our new monthly values represent the average values across the ~30 days of data.

Refer to the documentation for more info about `.resample()` : Pandas .resample() documentation

In [12]:
```
df_btc_price_monthly = df_btc_price.resample("M", on="DATE").mean()
display(df_btc_price_monthly)
```

	CLOSE	VOLUME
DATE		
2014-09-30	407.182428	2.934864e+07
2014-10-31	364.148873	2.912885e+07
2014-11-30	366.099799	2.199111e+07
2014-12-31	341.267871	1.784201e+07
2015-01-31	248.782547	3.544555e+07
...	...	...
2020-05-31	9263.151745	4.149575e+10
2020-06-30	9489.227214	2.169711e+10
2020-07-31	9589.899729	1.760688e+10
2020-08-31	11667.277572	2.290423e+10
2020-09-30	10655.702218	3.039781e+10

73 rows × 2 columns

Data Visualisation

Notebook Formatting & Style Helpers

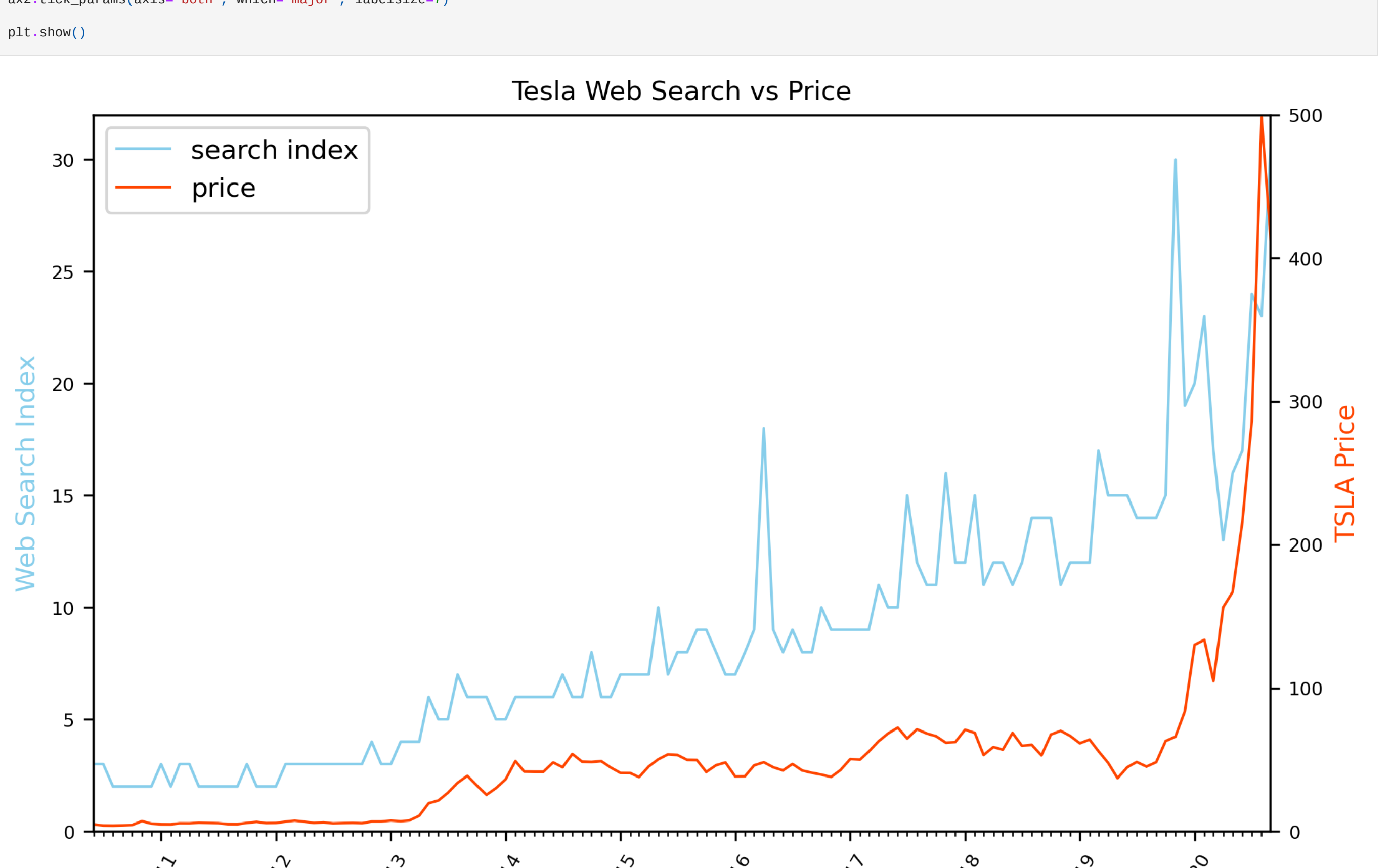
In [13]:
```
# Creating locators for ticks on the time axis
years = mdates.YearLocator()
months = mdates.MonthLocator()
years_fmt = mdates.DateFormatter("%Y")
```

Here we created locator for ticks on the time axis. This makes it easier to see the time values when we graph the data.

Tesla Stock Price v.s. Search Volume

Now, let's jump right into it. First, we will plot Tesla's stock price against its search volume:

In [14]:
```
plt.figure(figsize=(8,5), dpi=300)
plt.xticks(rotation=60)
plt.title("Tesla Web Search vs Price", fontsize=10)

ax1 = plt.gca()
ax2 = ax1.twinx()

ax1.set_ylabel("Web Search Index", color="skyblue", fontsize=10)
ax2.set_ylabel("TSLA Price", color="orangered", fontsize=10)
ax1.set_xlabel("Year", fontsize=10)
ax1.set_xlim(pd.Timestamp("2010-05-30"), pd.Timestamp("2020-08-28"))
ax1.set_ylim(0, 32)
ax2.set_ylim(0, 500)
ax1.xaxis.set_major_locator(years)
ax1.xaxis.set_major_formatter(years_fmt)
ax1.xaxis.set_minor_locator(months)


lns1 = ax1.plot(df_tesla.MONTH, df_tesla.TSLA_WEB_SEARCH, color="skyblue", lw=1, label="search index")
lns2 = ax2.plot(df_tesla.MONTH, df_tesla.TSLA_USD_CLOSE, color="orangered", lw=1, label="price")
leg = lns1 + lns2
labs = [l.get_label() for l in leg]
ax1.legend(leg,labs,loc=0)
ax1.tick_params(axis='both', which='major', labelsize=7)
ax2.tick_params(axis='both', which='major', labelsize=7)

plt.show()
```



In order to plot two different datasets with different y-values, we need to create two different axes. We can plot the two lines by giving them a set of values on both the x-axis and y-axis. Keep in mind, they share the same x-axis, but have different y-axes.

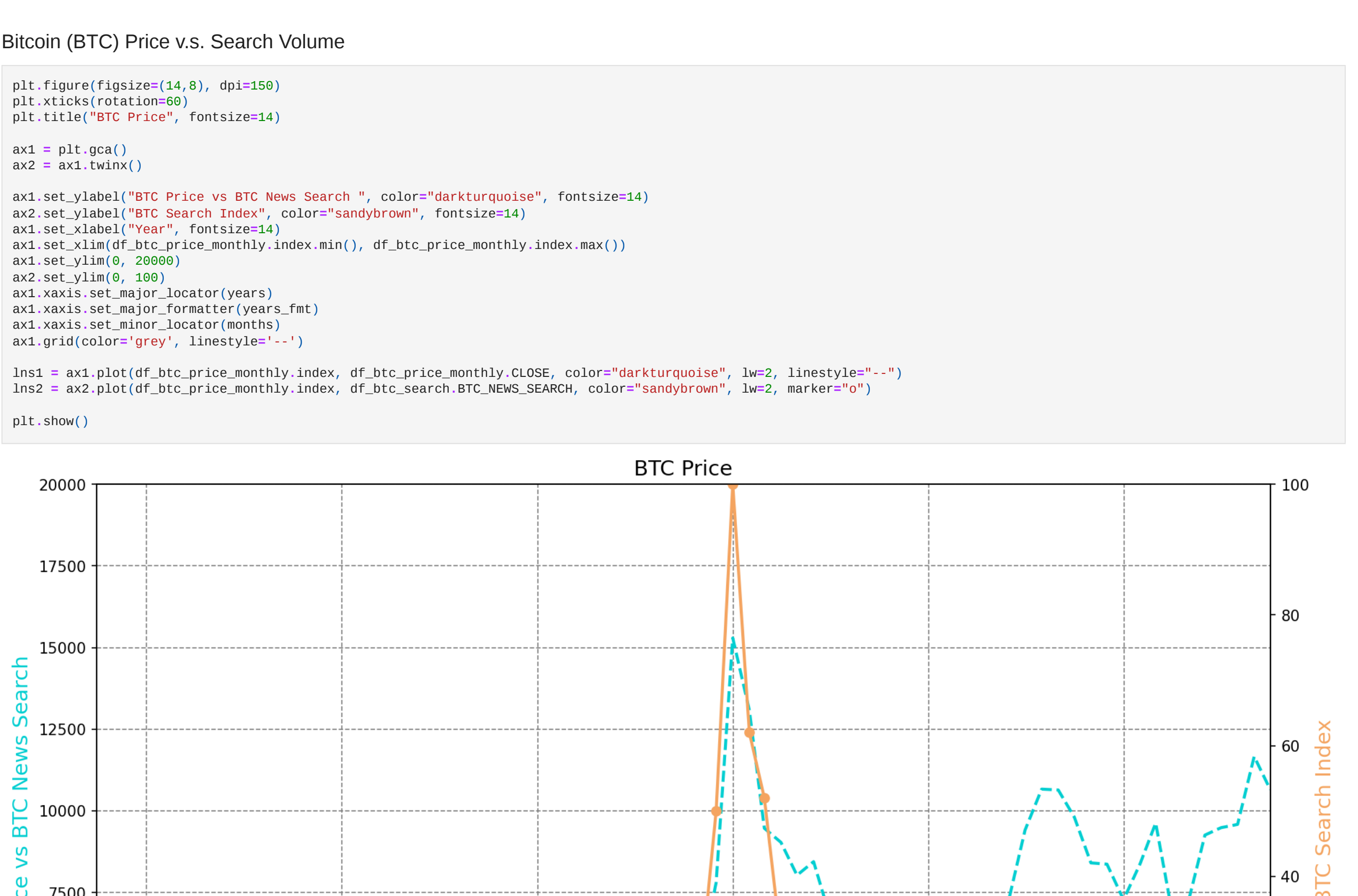
Now to distinguish the datasets from each other, we add color (skyblue and red), give them labels, and set their font sizes. We can limit both the x-axis and y-axis values to ensure that we are displaying the relevant data and not empty data values. We can also add xticks so that we aren't just looking at the time at yearly increments, but monthly. After sizing adjustments, adding finishing touches, and including a legend, our graph is done!

So what does our graph tell us?

We can clearly see a correlation between the stock's price and its Google Trends search index!

Bitcoin (BTC) Price v.s. Search Volume

In [15]:
```
plt.figure(figsize=(14,8), dpi=150)
plt.xticks(rotation=60)
plt.title("BTC Price", fontsize=14)

ax1 = plt.gca()
ax2 = ax1.twinx()

ax1.set_ylabel("BTC Price vs BTC News Search ", color="darkturquoise", fontsize=14)
ax2.set_ylabel("BTC Search Index", color="sandybrown", fontsize=14)
ax1.set_xlabel("Year", fontsize=14)
ax1.set_xlim(df_btc_price_monthly.index.min(), df_btc_price_monthly.index.max())
ax1.set_ylim(0, 20000)
ax2.set_ylim(0, 100)
ax1.xaxis.set_major_locator(years)
ax1.xaxis.set_major_formatter(years_fmt)
ax1.xaxis.set_minor_locator(months)
ax1.grid(color='grey', linestyle='--')

lns1 = ax1.plot(df_btc_price_monthly.index, df_btc_price_monthly.CLOSE, color="darkturquoise", lw=2, linestyle="--")
lns2 = ax2.plot(df_btc_price_monthly.index, df_btc_search.BTC_NEWS_SEARCH, color="sandybrown", lw=2, marker="o")

plt.show()
```



Here we investigate Bitcoin's price compared to its search volume, and we can see that in early 2018, the search volume index was at its peak. Unsurprisingly, this corresponds to the surge in Bitcoin price as it reached a new all time high by many multiples during that time period.

In the years after 2018, we can see Bitcoin's price slowly increase, while the search index slowly decreased leading to a negative correlation which may seem odd. But there are many factors at play, as Bitcoin started becoming more widely adopted, people knew more about the digital currency.

Similarly, things like Ethereum, Alt-coins, and other crypto assets became increasingly popular. As a whole, I wouldn't be surprised to see a clear, strong, positive correlation between Crypto's entire market cap and its search index.

Unemployement Benefits Search vs. Actual Unemployment in the U.S.

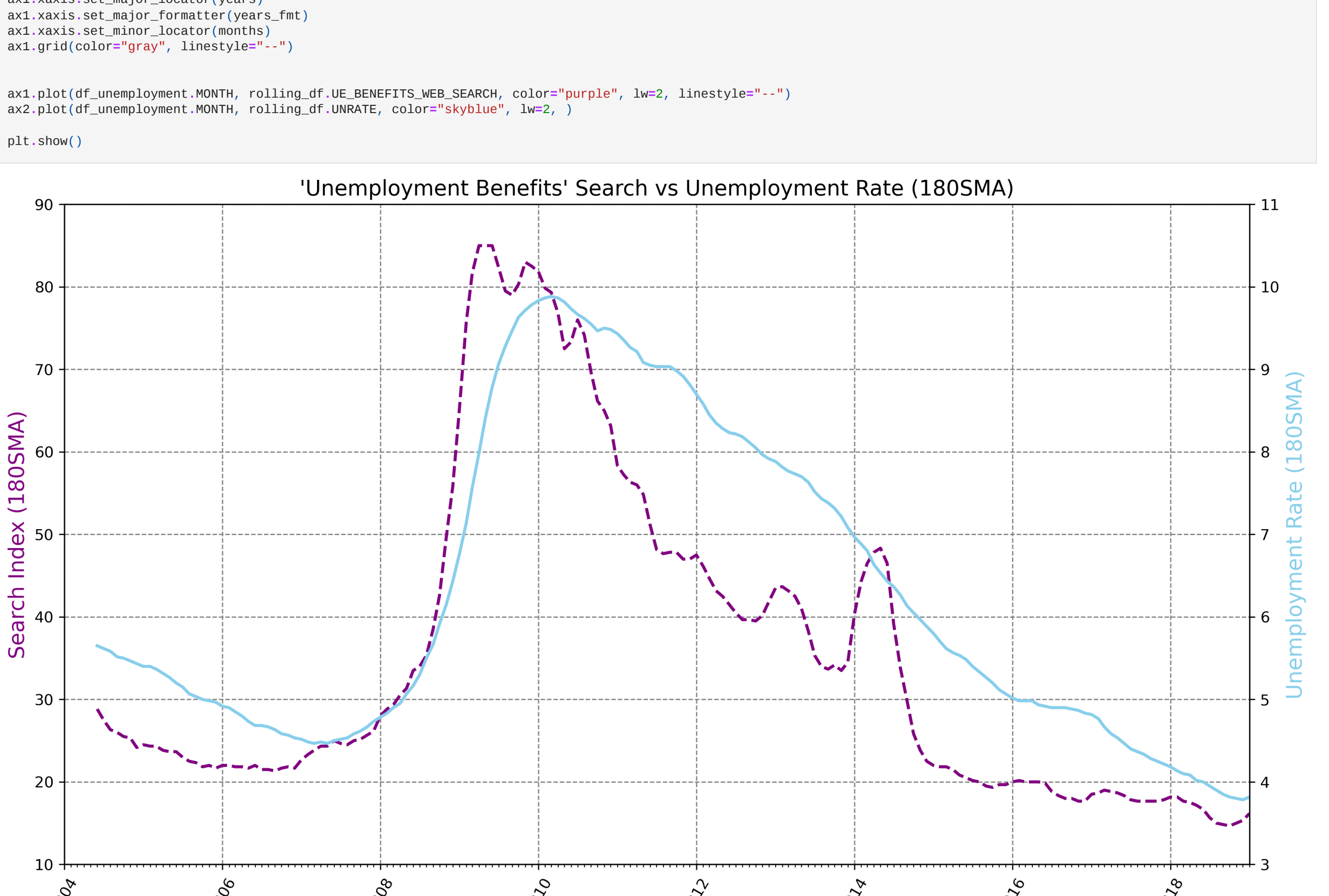
In [16]:
```
plt.figure(figsize=(14,8), dpi=400)
plt.xticks(rotation=60)
plt.title("Monthly 'Unemployment Benefits' Search vs Unemployment Rate", fontsize=14)

ax1 = plt.gca()
ax2 = ax1.twinx()

ax1.set_ylabel("Search Index", color="purple", fontsize=14)
ax2.set_ylabel("Unemployment Rate", color="skyblue", fontsize=14)
ax1.set_xlabel("Year", fontsize=14)
ax1.set_xlim(df_unemployment.MONTH.min(), df_unemployment.MONTH.max())
ax1.set_ylim(10, 100)
ax2.set_ylim(3, 11)
ax1.xaxis.set_major_locator(years)
ax1.xaxis.set_major_formatter(years_fmt)
ax1.xaxis.set_minor_locator(months)
ax1.grid(color='grey', linestyle='--')


ax1.plot(df_unemployment.MONTH, df_unemployment.UE_BENEFITS_WEB_SEARCH, color="purple", lw=1)
ax2.plot(df_unemployment.MONTH, df_unemployment.UNRATE, color="skyblue", lw=2, linestyle="--")

plt.show()
```

If we take a close look at the plot, we can see something interesting. What happens to the lines near the end of each year? Well, there seems to be a spike in the unemployment search index at every vertical grid line as well as between each vertical grid-line. This tells us that there does appear to be a seasonality in the job market.

We can also see that the financial crisis in 2008 drove unemployment rates to more than double what it was a few years ago, as well as causing a significant spike in the search index. It took over 8 years for unemployment rates to go back down under 5%, around the same levels before the crisis.

The search volume plot has a lot of spikes, but we can smooth the data by calculating the 6-month rolling average for it. Because our time data is already incremented by months, we can use the `.rolling(6)` method to calculate the rolling average for the last 6 observations which in our case is 6 months.

In [17]:
```
rolling_df = df_unemployment[["UE_BENEFITS_WEB_SEARCH","UNRATE"]].rolling(6).mean()

plt.figure(figsize=(14,8), dpi=300)
plt.xticks(rotation=60)
plt.title("'Unemployment Benefits' Search vs Unemployment Rate (180SMA)", fontsize=14)


ax1 = plt.gca()
ax2 = ax1.twinx()

ax1.set_ylabel("Search Index (180SMA)", color="purple", fontsize=14)
ax2.set_ylabel("Unemployment Rate (180SMA)", color="skyblue", fontsize=14)
ax1.set_xlabel("Year", fontsize=14)
ax1.set_xlim(df_unemployment.MONTH.min(), df_unemployment.MONTH.max())
ax1.set_ylim(10, 90)
ax2.set_ylim(3, 11)
ax1.xaxis.set_major_locator(years)
ax1.xaxis.set_major_formatter(years_fmt)
ax1.xaxis.set_minor_locator(months)
ax1.grid(color="gray", linestyle="--")


ax1.plot(df_unemployment.MONTH, rolling_df.UE_BENEFITS_WEB_SEARCH, color="purple", lw=2, linestyle="--")
ax2.plot(df_unemployment.MONTH, rolling_df.UNRATE, color="skyblue", lw=2, )

plt.show()
```



Now the graph looks much cleaner, and trends are easier to see with less perturbations. We can observe how the search index seems to spike before the unemployment rate; and decreases before the unemployment rate does. This tells us that the search index is a *leading indicator*, while the the unemployment rate is a *lagging indicator*

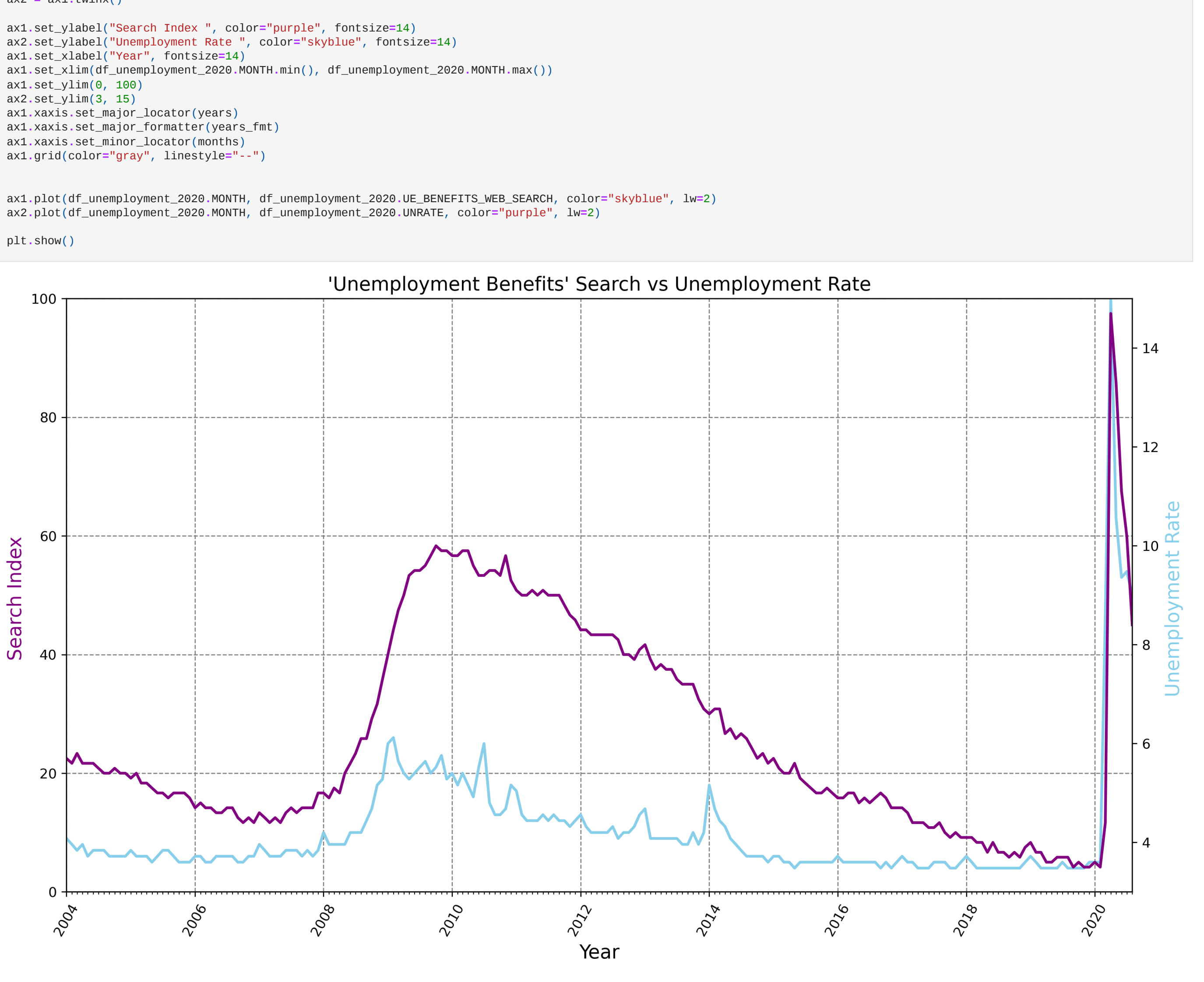
Including 2020 in Unemployment Charts

In [18]:
```
df_unemployment_2020 = pd.read_csv("UE Benefits Search vs UE Rate 2004-20.csv")
df_unemployment_2020.MONTH = pd.to_datetime(df_unemployment_2020.MONTH)
display(df_unemployment_2020)
```

	MONTH	UE_BENEFITS_WEB_SEARCH	UNRATE
0	2004-01-01	9	5.7
1	2004-02-01	8	5.6
2	2004-03-01	7	5.8
3	2004-04-01	8	5.6
4	2004-05-01	6	5.6
...	...	...	...
195	2020-04-01	100	14.7
196	2020-05-01	63	13.3
197	2020-06-01	53	11.1
198	2020-07-01	54	10.2
199	2020-08-01	50	8.4

200 rows × 3 columns

In [19]:
```
plt.figure(figsize=(14,8), dpi=300)
plt.xticks(rotation=60)
plt.title("'Unemployment Benefits' Search vs Unemployment Rate", fontsize=14)


ax1 = plt.gca()
ax2 = ax1.twinx()

ax1.set_ylabel("Search Index ", color="purple", fontsize=14)
ax2.set_ylabel("Unemployment Rate ", color="skyblue", fontsize=14)
ax1.set_xlabel("Year", fontsize=14)
ax1.set_xlim(df_unemployment_2020.MONTH.min(), df_unemployment_2020.MONTH.max())
ax1.set_ylim(0, 100)
ax2.set_ylim(3, 15)
ax1.xaxis.set_major_locator(years)
ax1.xaxis.set_major_formatter(years_fmt)
ax1.xaxis.set_minor_locator(months)
ax1.grid(color="gray", linestyle="--")


ax1.plot(df_unemployment_2020.MONTH, df_unemployment_2020.UE_BENEFITS_WEB_SEARCH, color="skyblue", lw=2)
ax2.plot(df_unemployment_2020.MONTH, df_unemployment_2020.UNRATE, color="purple", lw=2)

plt.show()
```



Factoring in 2020 data from the pandemic, we see something that is much more alarming than what we previously observed. The spike in unemployment rate and its corresponding "unemployment benefits" search index is much steeper than what happened during the financial crisis in 2008. The unemployment rate reached unprecendented levels during that time, greatly overshadowing the financial crisis. Because an event like this has never happened before in modern history, we can only hope for a swift recovery...