Welcome to my News Analyzer Project:

Overview: This project involves building a news analyzer that leverages the New York Times API to fetch and analyze news articles based on user-defined criteria. The focus is on extracting keywords, tracking trends over time, and presenting insights through visualizations

This project demonstrates the following skills:

1. **Python Programming**: Apply Python programming to manage data retrieval, processing, and visualization.
2. **API Integration**:Utilize API connection to pull news data automatically.
3. **Config File Management**: Implement secure API connection by passing credentials through a configuration file
4. **Data Cleaning**:Organize and clean data, addressing missing information and preparing it for analysis.
5. **Keyword Analysis**:Perform keyword frequency analysis to identify popular topics in news articles.
6. **Time Series Analysis**:Track keyword occurrences over time to detect trends.
7. **User Input Handling**:Implement customizable user options for selecting news sections and date ranges.
8. **Data Visualization**:Create visual representations such as charts or graphs to display trends.
9. **Project Documentation**:Write clear and concise instructions to explain code functionality.

By: Shaquille Jones

In [2]:
```
import requests as r
import pandas as pd
from io import StringIO
import seaborn as sns
from config import API_KEY
import matplotlib.pyplot as plt
import json
```

Data Collection:

To begin, our data collection steps, the below cell will do the following steps:

-**User Welcome Prompt**: Start by welcoming the user and prompts them to select a section of the New York Times they wish to analyze. This is done by the print statement asking for section to be viewed.

-**Choice and Validation**: To ensure choices are within the expected range, we'll limit user input by declaring a while loop that will break and ask user to input the expected.

-**User Choice Mapping**: To dynamically reflect the user choices, we will, declare a dictionary with key/value pairs in line with choices 1 - 4 based on avaible section to be analyzed.

-**Dynamic API Call**: Based on user input and choice mapping above, we can now make a dynamic API call using the selected section. The API URL will be modified by passing through the declared chosen section, and will be retrived from the New York Times API.

In [143…
```
#User Welcome Prompt:

print("Welcome, thanks for using the News Analyzer. Please start by slecting a New York Times section you'll like to begin

#User Choice and Validation:
"""While loop to ensure users enter values associated with sections, but also does
not allow them to ensure values out of range"""
while True:
    print("1. Tech")
    print("2. Health")
    print("3. Politics")
    print("4. Automobiles")

    user_choice = input("Please select (1 - 4)")

    if user_choice in ['1', '2', '3', '4']:
        break
    else: print("Not a valid choice. Please choose from 1 - 4")


#User Choice Mapping: Key/Value pairs
"""Initiate choice mapping based on choices"""
choice_map ={
    '1': 'technology',
    '2': 'health',
    '3': 'politics',
    '4': 'automobiles'
}
```

```
"""The section variable is what is going to be the section of the api, that will dynamically
select based on user choice"""
section = choice_map[user_choice]

#Dynamic API Call:
nyt_top_story = r.get(f"https://api.nytimes.com/svc/topstories/v2/{section}.json?api-key={API_KEY}").content.decode()
print(f"You've chosen {section.capitalize()}, let's dive deeper...")
```

```
Welcome, thanks for using the News Analyzer. Please start by slecting a New York Times section you'll like to begin analyzin
g:
1. Tech
2. Health
3. Politics
4. Automobiles
You've chosen Automobiles, let's dive deeper...
```

Data Processing and Transformation:

Below cell will do the following:

-**Create DataFrame & Error Handling**: Structure the JSON data and extract the relevant data into a DataFrame.

-**Date Column Adjustment**: Goal is to Standardize the published_date column:

- Remove Time: Convert the date into a format that Python can read w/o throwing an error.
- Format Adjustment: Change the date format to a string
- Reshape to final format: Convert the date back into ISO format (YYYY-MM-DD)

-**Column Selection**: Focus on key columns such as title, published_date, des_facet, and byline by using .loc[] to filter the DataFrame.

-**Date Availability**: Display the minimum and maximum dates available in the data for the user to select a range.

-**Date Range Input**: Use of the function get_date_range() to prompt the user for a minimum and maximum date within the available range. The dates are then validated for format and range.

-**Filter Data**: Use the date range the user selected to filter the data.

-**Keyword Expansion**: Split keywords (des_facet) into individual rows.

-**Data Grouping**: Group the filtered data by published_date, des_facet, title, and byline, and calculate the frequency of keywords.

-**Column Renaming**: Rename the columns to more user-friendly names, such as Date, Keywords, Headline, Frequency, and Journalist.

In [144…

```
#Create DataFrame & Error Handling:
if isinstance(nyt_top_story, str):
    nyt_top_story = json.loads(nyt_top_story)
df = pd.json_normalize(nyt_top_story['results'])

"""Given the API have thrown an error stating variable nyt_top_story can't be a string,
utilize the Json loads to read proper JSON structure, if not then the Json_normalize function shall work"""



#Date Column Adjustment:
df['published_date'] = pd.to_datetime(df['published_date'])
df['published_date'] = df['published_date'].dt.strftime('%m/%d/%y')
df['published_date'] = pd.to_datetime(df['published_date'], format='%m/%d/%y')

"""Given the API reads in JSON format, date field may not be structured to the format in which we will be able to use
without throwing an error in Pyhton. The time field is included, can be useful, but in this case for clarity, we should omi

#Column Selection:
df = df.loc[:, ['title', 'published_date', 'des_facet', 'byline']]


#Data Availability: Set Variable
avail_min_date = df['published_date'].min().date()
avail_max_date = df['published_date'].max().date()

"""The above is to showcase to the user the available date range in the data, which will be showcased to the user"""

#Data Availbility: Print to user
print(f"Data Availability: {avail_min_date} - {avail_max_date}")
print("Please select the date range you'd like to see (format: m/d/y)")


#Define function
def get_date_range(avail_min_date, avail_max_date):
    while True:
        try:
            date_min = input("1. Select the minimum date to view (m/d/y): ")
            date_max = input("2. Now select the maximum date to view (m/d/y): ")
            date_min_parsed = pd.to_datetime(date_min, format='%m/%d/%y')
            date_max_parsed = pd.to_datetime(date_max, format='%m/%d/%y')

            """1. The above has date_min and date_max, which will be an input from the user
            2. date_min_parsed and date_max_parsed pulls from the user input, but reformats the date"""

            #Valid Range Check:
            if avail_min_date <= date_min_parsed.date() <= avail_max_date and avail_min_date <= date_max_parsed.date() <= a
                return date_min_parsed, date_max_parsed
            else:
                print(f"Dates out of range. Please enter dates between {min_date} and {max_date}.")
        except ValueError:
            print("Invalid date format. Please enter the date in m/d/y format (example: 11/01/24).")

        """To ensure that users enter a valid date and one in range, this funtion will be passed,
        if not, the first fail safe is to ask for an in range date, and the next being a valid date"""




#Calling the function and passing through min and max date:
date_min, date_max = get_date_range(avail_min_date, avail_max_date)

#Apply filter based on user input:
filtered_data = df[(df['published_date'] >= date_min) & (df['published_date'] <= date_max)]

#Handle concatenation fo des_facet:
filtered_data = filtered_data.explode('des_facet')

"""des_facet which is the keyword column we'll be using is usually nested with many keywords together,
so the goal is to view them as a stand alone """

#Grouping Data:
filtered_data = filtered_data.groupby(['published_date', 'des_facet', 'title', 'byline']).size().reset_index(name='count')

#Column Renaming:
filtered_data = filtered_data.rename(columns={'published_date': 'Date', 'des_facet': 'Keywords',
    'title': 'Headline',
    'count': 'Frequency',

    'byline': 'Journalist'
})
```

```
Data Availability: 2024-07-04 - 2024-10-15
Please select the date range you'd like to see (format: m/d/y)
```

Keyword analysis and ranking:

Below cell will do the following table generation:

-**Keyword Frequency Calculation**: Segment data to aggregate the total frequency of each keyword.

-**Display Keyword Frequency**: Display final dataframe along with the keyword corresponding frequencies.

In [145…

```
"""This is showcase the frequency of keywords by using the sum pandas function
but also, only looking at top 20, since there's lots of keywords"""

#Keyword Frequency Calculation:
kw_cnt= filtered_data.groupby('Keywords')['Frequency'].sum().sort_values(ascending=False).iloc[:21]
kw_cnt= kw_cnt.reset_index()

#Showcase the most frequently used Kewwords:
kw_cnt
```

Out[145…

	Keywords	Frequency
0	Automobiles	29
1	Electric and Hybrid Vehicles	24
2	Factories and Manufacturing	10
3	Batteries	9
4	Labor and Jobs	7
5	Driverless and Semiautonomous Vehicles	6
6	Layoffs and Job Reductions	5
7	Traffic Accidents and Safety	5
8	Taxicabs and Taxicab Drivers	5
9	Customs (Tariff)	4
10	Prices (Fares, Fees and Rates)	4
11	Automobile Racing	4
12	Global Warming	4
13	Protectionism (Trade)	4
14	Organized Labor	3
15	International Relations	3
16	International Trade and World Market	3
17	Fuel Emissions (Transportation)	3
18	Company Reports	3
19	Collectors and Collections	3
20	Antique and Classic Cars	3

Data visualization: Keywords Frequency

Below cell will do the following visualiztion:

-**Goal**: Visualize the top 20 most frequently mentioned keywords using a bar chart to better understand their distribution.

-**Chart Dimensions and Setup**:

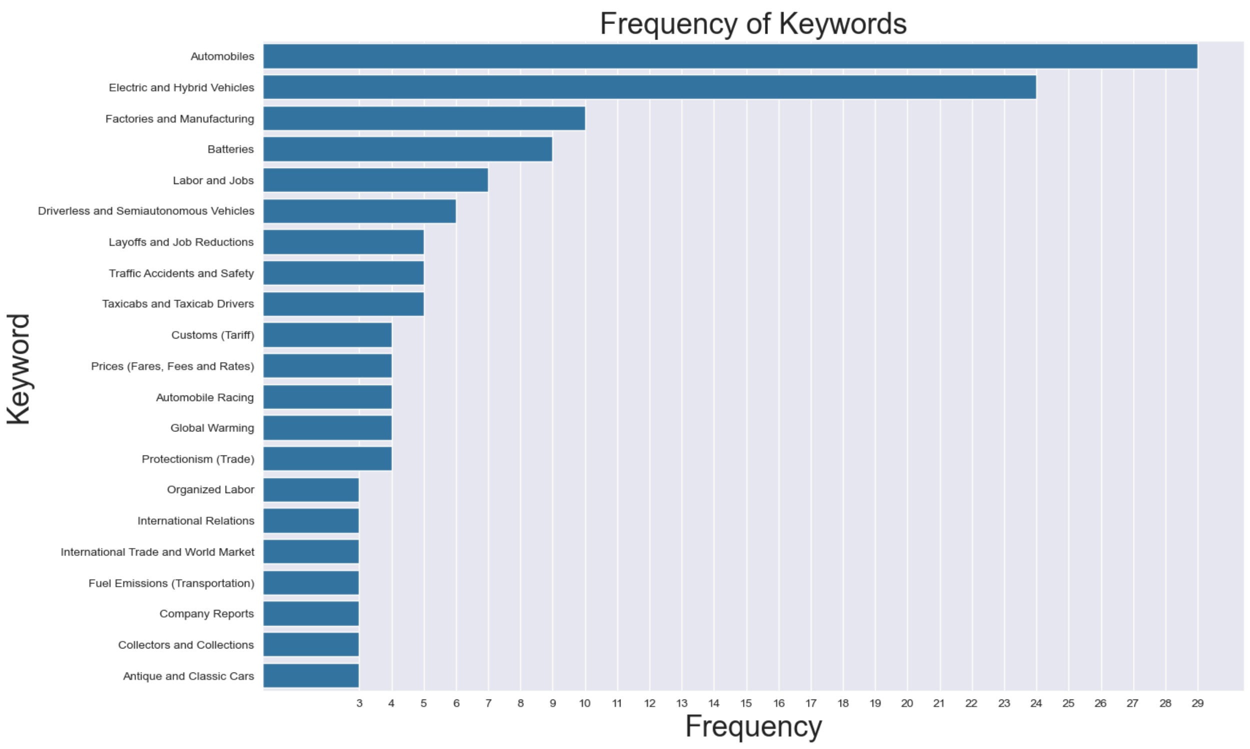
- Set Plot Style: Apply the darkgrid style for professional appearance.
- Viz Size: Structure a readable chart by declaring width and height.
- X-axis Ticks: Base x-axis ticks on min and max of frequency values.
- Bar Chart Creation: Plot the keyword frequencies, by declaring x and y axis.

-**Chart Labels and Title**:

- Title: Add a descriptive title to describe what is being shown.
- X-label: Declare what the X-axis label will be.
- Y-Label: Declare what the y-axis label will be.
- Show final chart.
- Display final version of the bar chart.

In [146…

```
"""Showcase the same above, but more stylised in a barchart"""

#Chart Dimensions and Setup:
sns.set_style('darkgrid')
plt.figure(figsize=(15, 10))
plt.xticks(range(min(kw_cnt['Frequency']), max(kw_cnt['Frequency'])+1))
sns.barplot(data=kw_cnt, y='Keywords', x='Frequency')

#Set Chart Labels and Title:
plt.title('Frequency of Keywords',fontsize=25)
plt.xlabel('Frequency', fontsize=25)
plt.ylabel('Keyword', fontsize=25)
plt.show()
```



Data visualization: Keyword Frequency Word Cloud

Below cell will do the following visualization:

Goal: Visualize the keyword frequency in a word cloud.

-Generate Word Cloud:

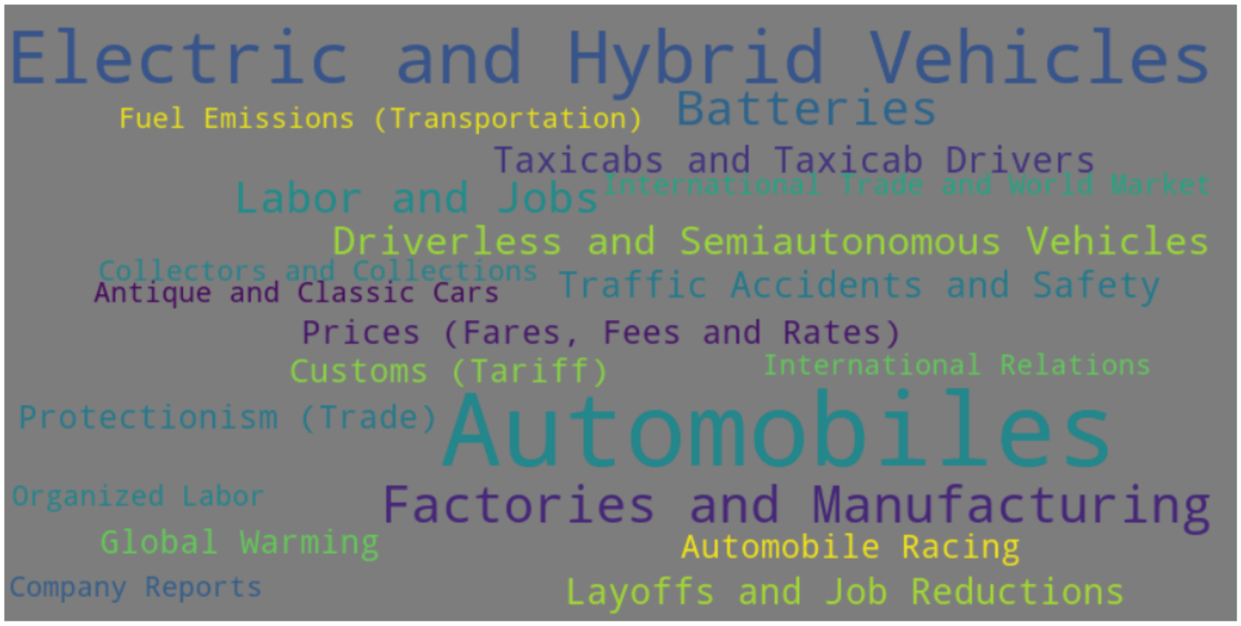
- Set Word Cloud Dimensions/Color.
- Background Color: Use grey as the background for contrast.
- Set Value and Elements: Set the column that will be used to display along with it's associated value.

-Design Elements:

- Figure Size: Adjust size for best viewing.
- Generate Figure: Declare the word cloud for a cleaner look.
- Axis: Turn off axis to clean up the display.
- Layout: Adjust word cloud padding to fill out space.
- Show final word cloud.

In [147…

```
#Import Relevant Package:
from wordcloud import WordCloud

#Generate Word Cloud:
wordcloud = WordCloud(width=800, height=400, background_color='grey').generate_from_frequencies(kw_cnt.set_index('Keywords'
"""For another perspective, a wordcloud is great for conceptualizing frequency just based on size"""

#Design elements of wordcloud:
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



Data visualization: Count Articles by Journalist

Below cell will do the following:

Goal: Count the number of articles written by each journalist and visualize the top 20 using a bar chart.

-String Cleaning and Grouping:

- Cleaning: Remove leading "By" from the Journalist column and strip any excess whitespace.
- Grouping: Group by target column, and count the number of articles, by filtering to top 20.

-Bar Chart Setup:

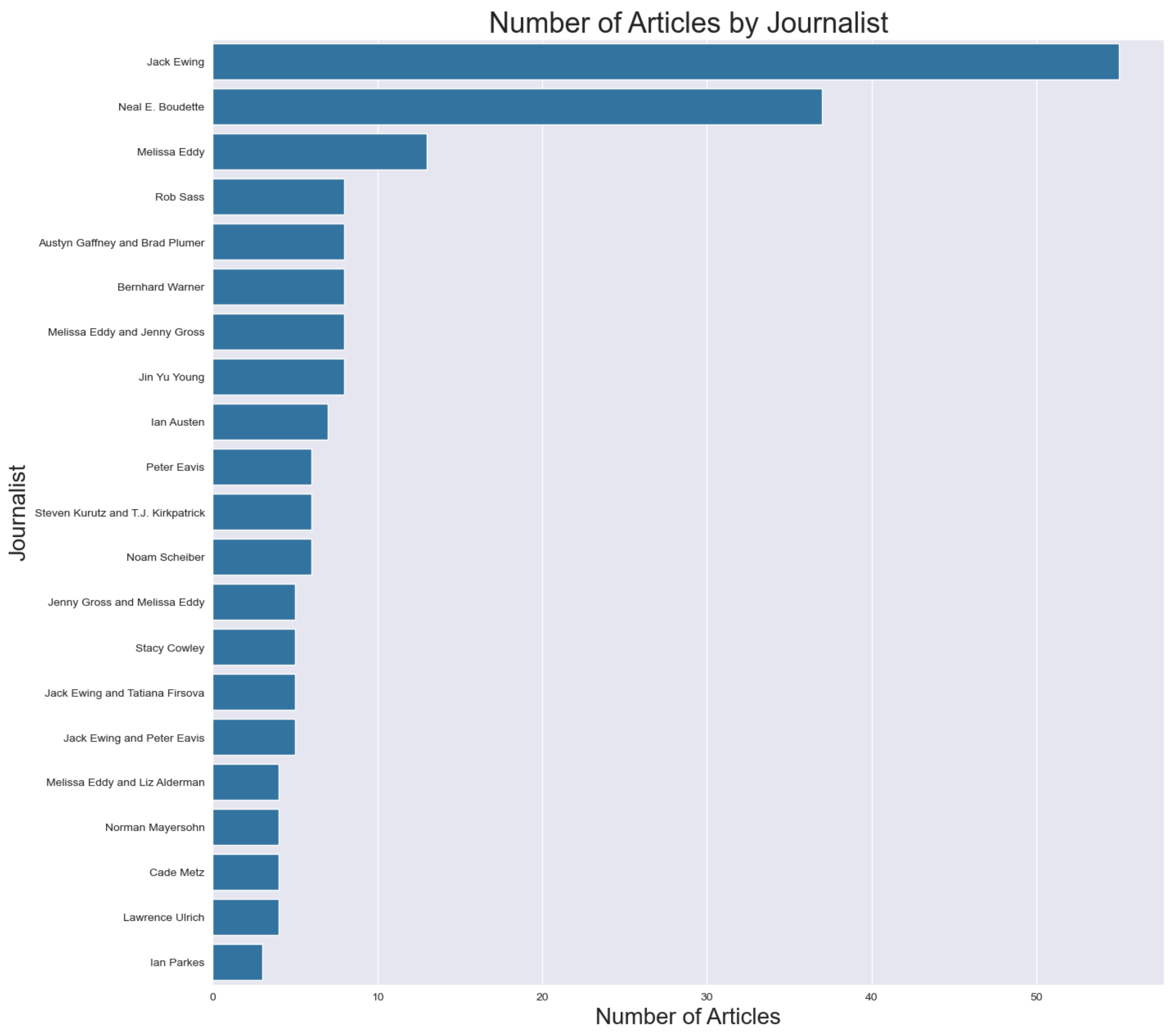
- Plot Style: Apply the darkgrid style for visual clarity.
- Viz Size: Structure a readable chart by declaring width and height.
- Bar Chart Creation: Plot the journalists on the y-axis and the number of articles on the x-axis.

-Chart Labels and Title:

- Title: Add a descriptive title to describe what is being shown.
- X-label: Declare the X-axis "Number of Articles."
- Y-label: Declare the y-axis "Journalist."
- Display Chart: Display the final version of the bar chart.

In [148…

```
#Count articles by Journalists:

#String Cleaning and Grouping:
filtered_data['Journalist'] = filtered_data['Journalist'].str.strip().str.replace('By ', '', regex=False)
journ_cnt = filtered_data.groupby('Journalist')['Headline'].count().sort_values(ascending=False).iloc[:21]

"""Since the Journalist column often starts with By, we use the string strip to omit if matches the string,
leave a blank space to ensure no leading spaces"""

#Chart Dimensions and Setup:
sns.set_style('darkgrid')
plt.figure(figsize=(15, 15))
sns.barplot(y=journ_cnt.index, x=journ_cnt.values)

#Chart Labels and Title:
plt.title('Number of Articles by Journalist', fontsize=25)
plt.xlabel('Number of Articles', fontsize=20)
plt.ylabel('Journalist', fontsize=20)
plt.show()
```



User Input: Date Aggregation Level

Below cell will do the following:

Goal: Allow the user to select the desired level of date aggregation for the data (daily, weekly, or monthly).

-User Input: Prompt user asking what level data they will like to see (1 = Daily, 2 = Weekly, 3 = Monthly)

-User Input: Initiate while loop to ensure the user inputs a valid choice (1-3)

-Initial Grouping: Group the filtered data by date and keywords to prepare for aggregation.

-Adjust Data Based on User Choice:

- Monthly Aggregation: If the user chooses "Monthly," group the data by Keywords and use the pd.Grouper function to aggregate by month.
- Weekly Aggregation: If the user chooses "Weekly," group the data by Keywords and aggregate by week.
- Daily Aggregation: If the user chooses "Daily," no changes are needed to the original grouping.

In [149…

```
#User Input: Date peiod
print("Finally, what level will you like to roll up the data: 1/2/3")
print("1. Daily")
print("2. Weekly")
print("3. Monthly")

#User Input: Loop Validation
"""This is to initiate how the user would like to aggregate the date field based on input, but ensuring
only correct values are inputted"""
while True:
    user_choice_period = input("Please select (1 - 3)")
    if user_choice_period in ['1', '2', '3']:
        break
    else:
        print("Not a valid choice. Please choose from 1 - 3")

#Group Data:
grouped_period_freq = filtered_data.groupby(['Date', 'Keywords']).sum().reset_index()


#Adjust data to suit user date aggregation:

#Monthly Aggregation:
if user_choice_period == '3':
    grouped_period_freq = grouped_period_freq.groupby(['Keywords', pd.Grouper(key='Date', freq='M')]).sum().reset_index()
    print("You've chosen Monthly, let's dive deeper...")

#Weekly Aggregation:
elif user_choice_period == '2':
    grouped_period_freq = grouped_period_freq.groupby(['Keywords', pd.Grouper(key='Date', freq='W')]).sum().reset_index()
    print("You've chosen Weekly, let's dive deeper...")

#Daily, no changes needed: Confirm message
else:
    print("You've chosen Daily, let's dive deeper...")
```

```
Finally, what level will you like to roll up the data: 1/2/3
1. Daily
2. Weekly
3. Monthly
You've chosen Weekly, let's dive deeper...
```

Data visualization: Articles Published Over Time

Below cell will do the following:

Goal: Visualize the number of articles published over the selected time period (daily, weekly, or monthly) based on user choice.

Dictionary User choice-Key/Value pairs: Map the user's aggregation choice (daily, weekly, monthly) to the corresponding time period ('D', 'W', 'M').

Article Count: Group data by date and count of headline

Resample Data: Resample the data based on the user's selected aggregation level (daily, weekly, or monthly).

Line Chart Dimensions and Setup:

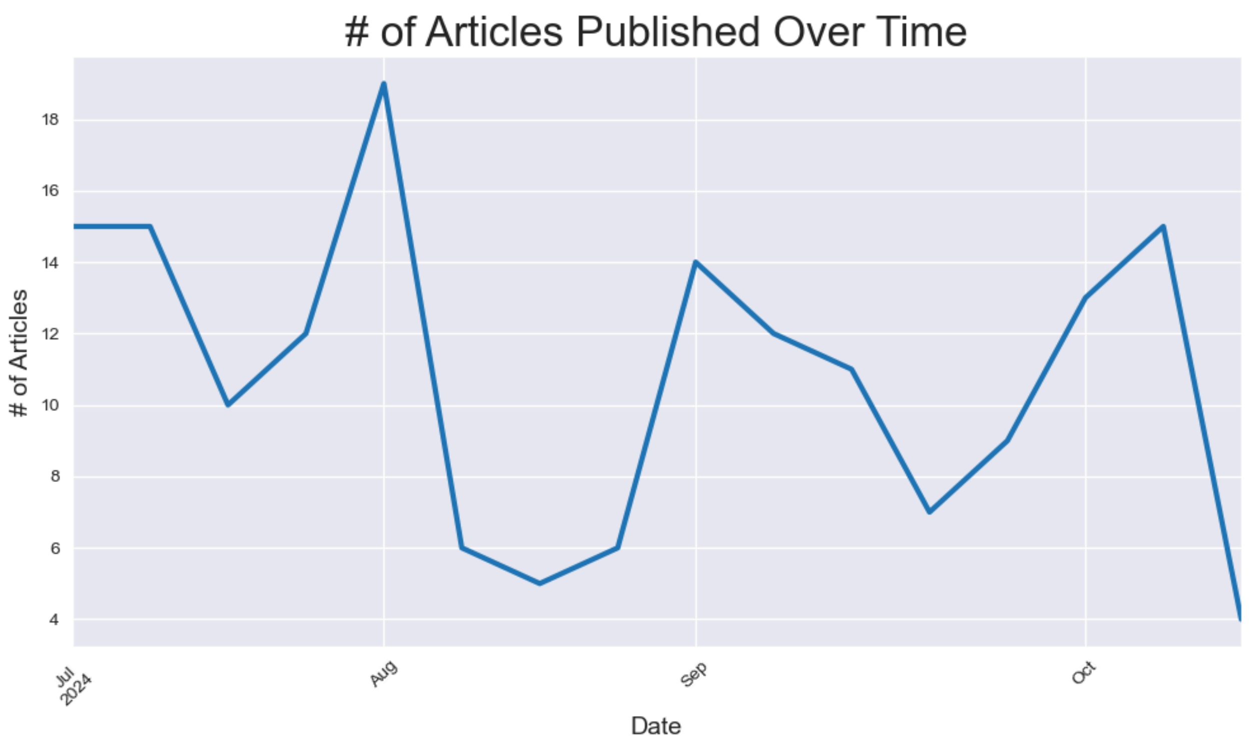
- Chart Size: Structure a readable chart by declaring width and height
- Line Width: Set the line to an appropriate better visibility.

Chart Labels and Title:

- Title: Add a descriptive title to describe what is being shown.
- X-label: Declare the X-axis "Date."
- Y-label: Declare the y-axis as "# of Articles."
- X-Ticks: Adjust the rotation on labels on the x-axis
- Tight layout: Automatically adjust spacing
- Display Chart: Display the final version of the line chart.

In [150…

```
#Look at articles Published over the period:

#Dictionary User choice: Key/Value pairs
period_mapping = {'1': 'D', '2': 'W', '3': 'M'}
agg_period = period_mapping.get(user_choice_period)
""""This uses the user choice and passes in the dictionary, if not Weekly, or Monthly then it defaults to daily value"""

#Count the number of articles:
article_cnt = grouped_period_freq.groupby('Date')['Headline'].count()

#Resample the data: based on the selected period (daily, weekly, monthly)
article_cnt= article_cnt.resample(agg_period).sum()
"""Given aggregating data over daily, weekly, or monthly, we need to call the resample function, to get the associated dist
per the date aggregation"""


#Line Chart Dimensions and Setup:
plt.figure(figsize=(10, 6))
article_cnt.plot(kind='line', linewidth=3)

#Set title and labels
plt.title('# of Articles Published Over Time', fontsize=24)
plt.xlabel('Date', fontsize=14)
plt.ylabel('# of Articles', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Data visualization: Frequency of Keywords Over Time

Below cell will do the following:

Goal: Visualize the frequency of the top 10 keywords over time using a line graph.

Pivot DataFrame: Pivot data where keyword becomes a column and dates as an index

Handle Missing Data: Fill any missing values with 0 for continuous lines

Select Top keywords: Limit to top 10 keywords based on their frequency.

Line Chart Dimensions and Setup:

- Chart Size: Structure a readable chart by declaring width and height and line width

Chart Labels and Title:

- Title: Add a descriptive title to describe what is being shown.
- X-label: Declare the x-axis as "Date."
- Y-label: Declare the y-axis as "Frequency."
- X-ticks: Rotate
- Tight layout: Automatically adjust spacing

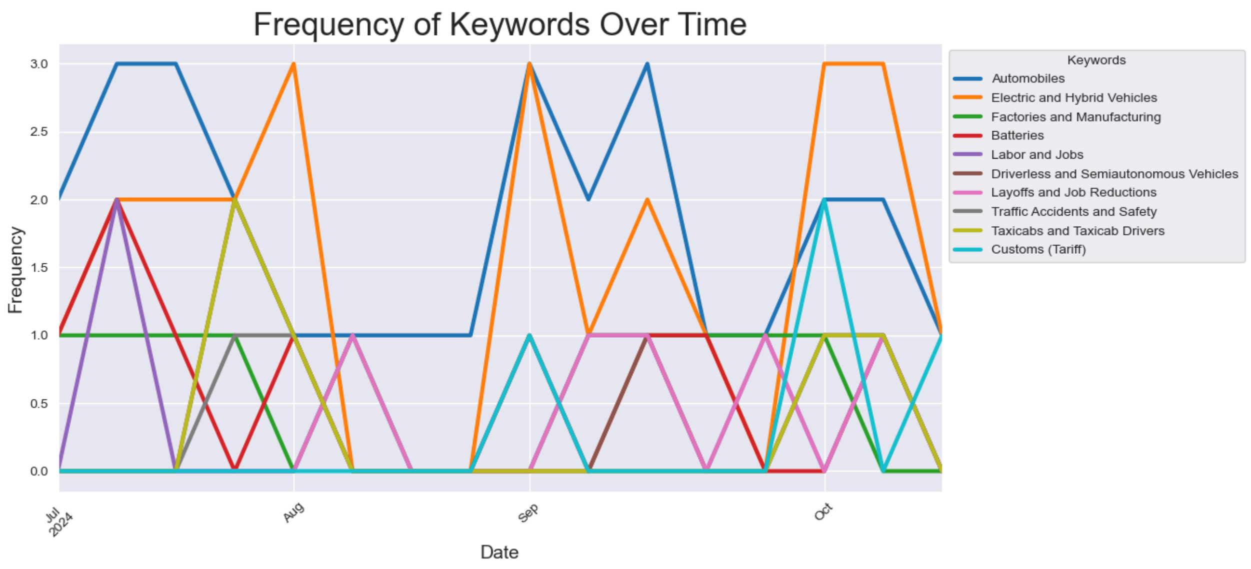
Legend Position: Adjust legend to ensure not overlapping with line chart.

Display Plot: Show final line chart

In [151…

```
#Pivot the DataFrame
plot = grouped_period_freq.pivot(index='Date', columns='Keywords', values='Frequency')
"""Conduct a pivot so that each Keyword becomes a column, with Dates as the index"""

#Handle Missing Data:
plot = plot.fillna(0)

"""To combat missing values for a certain date, this makes the line continous"""

#Select Top keywords:
plot = plot[plot.sum().sort_values(ascending=False).index[:10]]
"""To maintain aesthetic and allow for optimal view, limit amout keywords shown """
```

```
#Line Chart Dimensions and Setup:
plot.plot(figsize=(10, 6), linewidth=3)

#Display the plot:
plt.title("Frequency of Keywords Over Time", fontsize=24)
plt.xlabel("Date", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()


#Legend:
plt.legend(title="Keywords", loc='upper left', bbox_to_anchor=(1, 1))
"""Given the legend has a tendency to be over the visual, let's lock it's location to upper left"""

#Show the plot
plt.show()
```


Frequency of Keywords Over Time
Frequency
Jul 2024
Aug
Sep
Oct
Date
Keywords
Automobiles
Electric and Hybrid Vehicles
Factories and Manufacturing
Batteries
Labor and Jobs
Driverless and Semiautonomous Vehicles
Layoffs and Job Reductions
Traffic Accidents and Safety
Taxicabs and Taxicab Drivers
Customs (Tariff)