Annotated follow-along guide_Date string manipulations with Python

May 19, 2023

1 Annotated follow-along guide: Date string manipulations with Python

This notebook contains the code used in the following instructional video: Date string manipulations with Python

1.1 Introduction

Throughout the following programming activity, you will practice manipulating date strings in Python.Before beginning the activity, watch the associated instructional video and complete the in-video question. All the information you need to complete the activity and all the code you will be implementing is in this notebook.

1.2 Install packages and libraries

Before you begin the activity, import all the required libraries and extensions. Throughout the course, you will be using using pandas for operations, and matplotlib and seaborn for plotting.

[1]:
```
# Import statements
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

1.3 Activity overview

In this activity, you will work with 2016–2018 lightning strike data from the National Oceanic and Atmospheric Association (NOAA) to calculate weekly sums of lightning strikes and plot them on a bar graph. Then, you will calculate quarterly lightning strike totals and plot them on bar graphs.

1.4 Explore the data

[2]:
```
# Read in the data.
df = pd.read_csv('eda_manipulate_date_strings_with_python.csv')
df.head()
```

[2]:

	date	number_of_strikes	center_point_geom
0	2016-08-05	16	POINT(-101.5 24.7)
1	2016-08-05	16	POINT(-85 34.3)
2	2016-08-05	16	POINT(-89 41.4)
3	2016-08-05	16	POINT(-89.8 30.7)
4	2016-08-05	16	POINT(-86.2 37.9)

1.5 Create new time columns

As with similar datasets you have worked with, the `date` column data type is a string object, which limits what you can do with the data in this column. Converting the column to datetime will enable you to work with this data much more easily.

[3]:
```
# Convert the `date` column to datetime.
df['date'] = pd.to_datetime(df['date'])
```

Next, create four new columns: `week`, `month`, `quarter`, and `year`. You can do this by using the `datetime.strftime()` method of your datetime object. `strftime` is short for "string format time." You will use this method on the datetime data in the `week` column, and it will extract the information you specify, formatted as a string.

To specify the information to extract, use strftime format codes. You can find a full list of available codes to use in the strftime format codes documentation. In this case, you will use `%Y` for year, `%V` for week number, `%q` for quarter.

NOTE: The following process might take a minute or two to complete.

[4]:
```
# Create four new columns.
df['week'] = df['date'].dt.strftime('%Y-W%V')
df['month'] = df['date'].dt.strftime('%Y-%m')
df['quarter'] = df['date'].dt.to_period('Q').dt.strftime('%Y-Q%q')
df['year'] = df['date'].dt.strftime('%Y')
```

Use `head()` to check that the columns were created as intended.

[5]:
```
df.head(10)
```

[5]:

	date	number_of_strikes	center_point_geom	week	month	\
0	2016-08-05	16	POINT(-101.5 24.7)	2016-W31	2016-08	
1	2016-08-05	16	POINT(-85 34.3)	2016-W31	2016-08	
2	2016-08-05	16	POINT(-89 41.4)	2016-W31	2016-08	
3	2016-08-05	16	POINT(-89.8 30.7)	2016-W31	2016-08	

```
4 2016-08-05                16   POINT(-86.2 37.9)  2016-W31  2016-08
5 2016-08-05                16   POINT(-97.8 38.9)  2016-W31  2016-08
6 2016-08-05                16     POINT(-81.9 36)  2016-W31  2016-08
7 2016-08-05                16   POINT(-90.9 36.7)  2016-W31  2016-08
8 2016-08-05                16  POINT(-106.6 26.1)  2016-W31  2016-08
9 2016-08-05                16    POINT(-108 31.6)  2016-W31  2016-08
```

```
   quarter  year
0  2016-Q3  2016
1  2016-Q3  2016
2  2016-Q3  2016
3  2016-Q3  2016
4  2016-Q3  2016
5  2016-Q3  2016
6  2016-Q3  2016
7  2016-Q3  2016
8  2016-Q3  2016
9  2016-Q3  2016
```

1.6 Plot the number of weekly lightning strikes in 2018

Next, plot the number of weekly lightning strikes. Start by filtering the original dataset to 2018. Use the `groupby()` and `sum()` functions to get the number of strikes per week.

[6]:
```
# Create a new dataframe view of just 2018 data, summed by week.
df_by_week_2018 = df[df['year'] == '2018'].groupby(['week']).sum().reset_index()
df_by_week_2018.head()
```

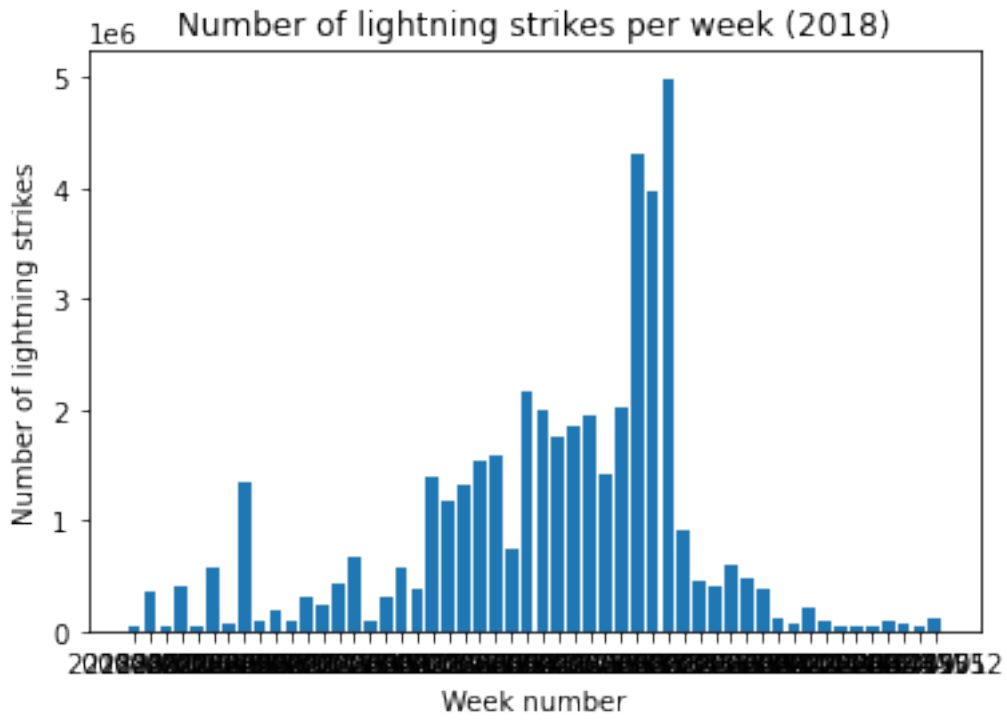
[6]:
```
        week  number_of_strikes
0  2018-W01              34843
1  2018-W02             353425
2  2018-W03              37132
3  2018-W04             412772
4  2018-W05              34972
```

Now you have a table of exactly what you need to plot the weekly lightning strike totals of 2018. You will use the `plt.bar()` function to plot the bar graph. Within the argument field, input the x-axis (the `week` column), then input the y-axis (or height) as the `number_of_strikes` column.

Use `plt.xlabel()`, `plt.ylabel()`, and `plt.title()` to add appropriate text to the graph.

[7]:
```
# Plot a bar graph of weekly strike totals in 2018.
plt.bar(x = df_by_week_2018['week'], height =␣
 ↪df_by_week_2018['number_of_strikes'])
plt.plot()
plt.xlabel("Week number")
plt.ylabel("Number of lightning strikes")
```

```
plt.title("Number of lightning strikes per week (2018)");
```

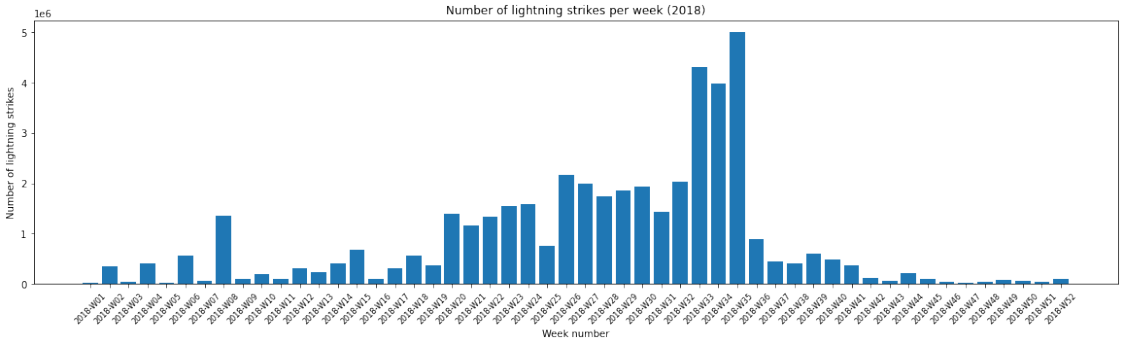


You have a graph, but you can't read the x-axis labels. To fix this problem, first make it bigger using `plt.figure(figsize=(20, 5))`. This will change the size to 20 inches wide by 5 inches tall.

Next, use the `plt.xticks()` function to access the tick labels on the x-axis. Using the `rotation` and `fontsize` keyword arguments, rotate the labels and make the font smaller.

[8]:
```
plt.figure(figsize = (20, 5)) # Increase output size
plt.bar(x = df_by_week_2018['week'], height =␣
 ↪df_by_week_2018['number_of_strikes'])
plt.plot()
plt.xlabel("Week number")
plt.ylabel("Number of lightning strikes")
plt.title("Number of lightning strikes per week (2018)")
plt.xticks(rotation = 45, fontsize = 8) # Rotate x-axis labels and decrease␣
 ↪font size

plt.show()
```



1.7 Plot the number of quarterly lightning strikes from 2016–2018

Next, plot lightning strikes by quarter for the full date range of available data. For a visualization, it will be easiest to work with numbers in millions, such as 25.2 million. As an example, the following code will divide the `number_of_strikes` column by one million.

[9]:
```
df_by_quarter = df['number_of_strikes'].div(1000000)
df_by_quarter.head()
```

[9]:
```
0    0.000016
1    0.000016
2    0.000016
3    0.000016
4    0.000016
Name: number_of_strikes, dtype: float64
```

This alone does not help much, because it just moved the decimal to the left. We'll need to format the numbers as well. We'll begin by grouping the data by quarter and summing.

Then, we'll create `number_of_strikes_formatted` column in the resulting dataframe by dividing by one million (like we did above), but also rounding to one digit after the decimal point, converting it to a string, and adding 'M' to the end, to represent millions.

[10]:
```
# Group 2016-2018 data by quarter and sum
df_by_quarter = df.groupby(['quarter']).sum().reset_index()

# Format as text, in millions
df_by_quarter['number_of_strikes_formatted'] =␣
 ↪df_by_quarter['number_of_strikes'].div(1000000).round(1).astype(str) + 'M'

df_by_quarter.head()
```

[10]:

	quarter	number_of_strikes	number_of_strikes_formatted
0	2016-Q1	2683798	2.7M

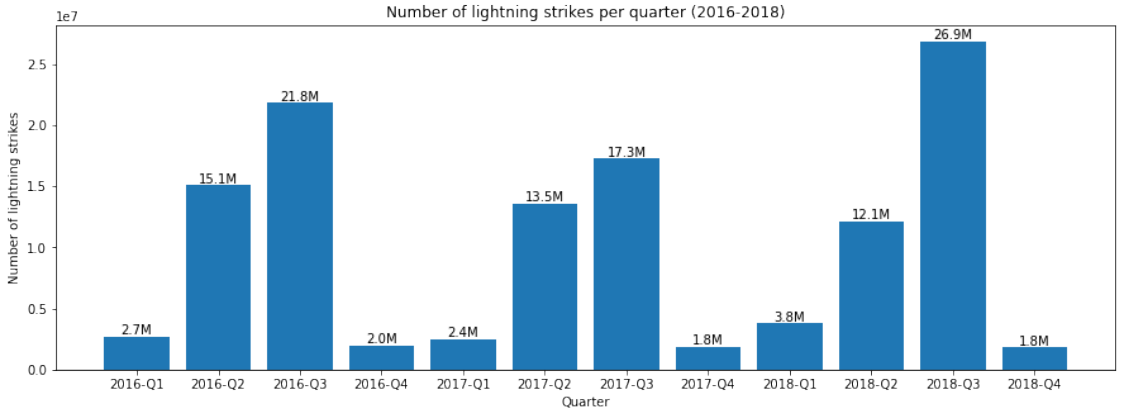
1	2016-Q2	15084857	15.1M
2	2016-Q3	21843820	21.8M
3	2016-Q4	1969754	2.0M
4	2017-Q1	2444279	2.4M

Before we start plotting, we'll write a function that will help label each bar in the plot with its corresponding number_of_strikes_formatted text. The function uses `plt.text()`, which is a pyplot function whose positional arguments are `x`, `y`, and `s`. `x` represents the x axis coordinates, `y` represents the y axis coordinates, and `s` is for the text that you want to appear at these coordinates. For more information, refer to the pyplot documentation.

[11]:
```
def addlabels(x, y, labels):
    '''
    Iterates over data and plots text labels above each bar of bar graph.
    '''
    for i in range(len(x)):
        plt.text(i, y[i], labels[i], ha = 'center', va = 'bottom')
```

Now we'll plot the bar graph.

[12]:
```
plt.figure(figsize = (15, 5))
plt.bar(x = df_by_quarter['quarter'], height =␣
 ↪df_by_quarter['number_of_strikes'])
addlabels(df_by_quarter['quarter'], df_by_quarter['number_of_strikes'],␣
 ↪df_by_quarter['number_of_strikes_formatted'])
plt.plot()
plt.xlabel('Quarter')
plt.ylabel('Number of lightning strikes')
plt.title('Number of lightning strikes per quarter (2016-2018)')
plt.show()
```



We can create a grouped bar chart to better compare year-over-year changes each quarter. We'll do this by creating two new columns that break out the quarter and year from the `quarter` column.

To do this, we use the `quarter` column and take the last two characters to get `quarter_number`, and take the first four characters to get `year`.

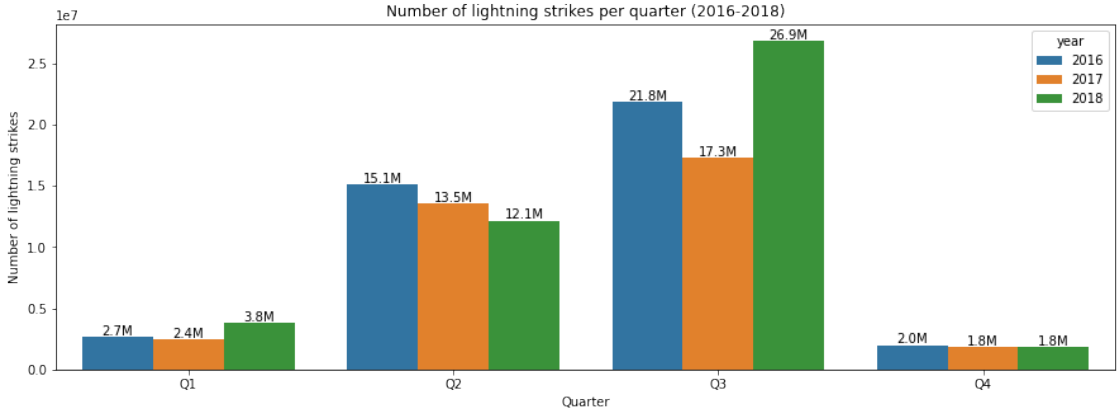
[13]:
```
# Create two new columns
df_by_quarter['quarter_number'] = df_by_quarter['quarter'].str[-2:]
df_by_quarter['year'] = df_by_quarter['quarter'].str[:4]
df_by_quarter.head()
```

[13]:

	quarter	number_of_strikes	number_of_strikes_formatted	quarter_number	year
0	2016-Q1	2683798	2.7M	Q1	2016
1	2016-Q2	15084857	15.1M	Q2	2016
2	2016-Q3	21843820	21.8M	Q3	2016
3	2016-Q4	1969754	2.0M	Q4	2016
4	2017-Q1	2444279	2.4M	Q1	2017

Next, fill in the bar chart parameters.

[14]:
```
plt.figure(figsize = (15, 5))
p = sns.barplot(
    data = df_by_quarter,
    x = 'quarter_number',
    y = 'number_of_strikes',
    hue = 'year')
for b in p.patches:
    p.annotate(str(round(b.get_height()/1000000, 1))+'M',
                    (b.get_x() + b.get_width() / 2., b.get_height() + 1.2e6),
                    ha = 'center', va = 'bottom',
                    xytext = (0, -12),
                    textcoords = 'offset points')
plt.xlabel("Quarter")
plt.ylabel("Number of lightning strikes")
plt.title("Number of lightning strikes per quarter (2016-2018)")
plt.show()
```



If you have successfully completed the material above, congratulations! You now understand how to manipulate date strings in Python and should be able to start doing it on your own datasets.