Day 72 Goals: what you will make by the end of the day

**Analyse the Popularity of Different Programming Languages over Time**

he oldest programming language still in use today is FORTRAN, which was developed in 1957. Since then many other programming languages have been developed. But which programming language is the most popular? Which programming language is the Kim Kardashian of programming languages; the one people just can't stop talking about?

StackOverflow will help us answer this burning question. Each post on Stack OverFlow comes with a Tag. And this Tag can be the name of a programming language.

To figure out which language is the most popular, all we need to do is count the number of posts on Stack Overflow that are tagged with each language. The language with the most posts wins!

Today you will learn:

* How to visualise your data and create charts with Matplotlib
* How to pivot, group and manipulate your data with Pandas to get it into the format you want
* How to work with timestamps and time-series data
* How to style and customise a line chart to your liking

Download and Open the Starter Notebook

**Open the Template**

To help you with your Data Science journey I'll provide you with starter notebook which already has some of the sections and challenges laid out. Download the .zip file from this lesson's resources, unzip it, and upload it to Google Drive. There you can open the file as a Colab Notebook.

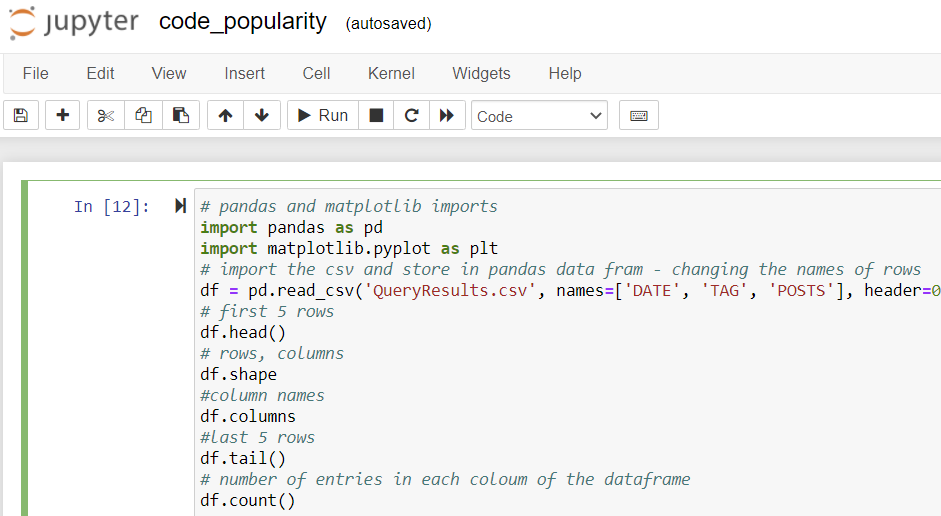
**Import the Data**

In this lesson, I've also included a QueryResults.csv file with the Stack Overflow data that we'll be using. Download this .csv and add it to your notebook.

Challenge

For the next steps, let's review the data exploration that we've done yesterday:

* Read the .csv file and store it in a Pandas DataFrame called df. Have a look at the [read\_csv() documentation](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html" \t "_blank) and try to provide these column names: ['DATE', 'TAG', 'POSTS']
* Look at the first and last 5 rows of the DataFrame.
* How many rows and how many columns does it have?
* Count the number of entries in each column.



[100 Days of Code - The Complete Python Pro Bootcamp for 2021](https://www.udemy.com/course/100-days-of-code/)

Leave a rating

Your progress

Share

Solution: Analysis by Programming Language

In order to look at the number of entries and the number of posts by programming language, we need to make use of the .groupby() method. The key is combining .groupby() with the TAG column, which holds as our categories (the names of the programming languages).

If we .sum() the number of posts then we can see how many posts each programming language had since the creation of Stack Overflow.

If we .count() the entries in each column, we can see how many months of entries exist per programming language.

# number of entries in each coloum of the dataframe   
df.count()  
#number of entrys by programming language   
df.groupby("TAG").sum()  
# how many months of entries exist in each coloumn   
df.groupby("TAG").count()

Data Cleaning: Working with Time Stamps

**Selecting an Individual Cell**

Let's take a closer look at the 'DATE' column in our DataFrame. We can use the double square bracket notation to look at the second entry in the column:

1. df['DATE'][1]

Alternatively, for column names no spaces, we can also use the dot-notation:

1. df.DATE[1]

I prefer the square bracket notation for column names since it's more flexible, but with the dot notation, you get to use autocomplete, which is also nice.

**Inspecting the Data Type**

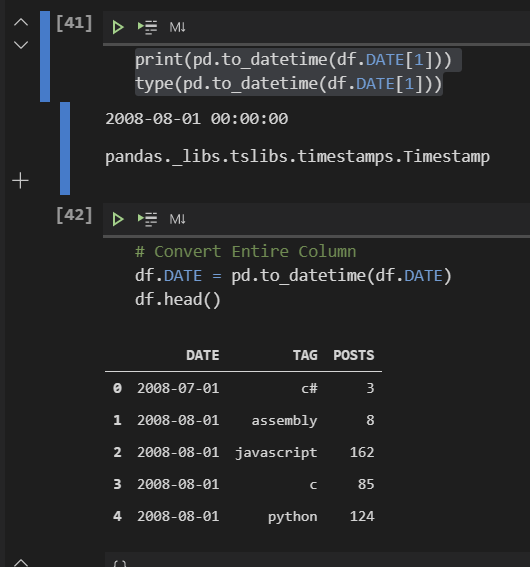
When we type check the contents of this cell, we see that we are not dealing with a date object, but rather with a string.

This is not very handy. Not only will the string format always show the unnecessary 00:00:00, but we also don't get the benefit of working with Datetime objects, which know how to handle dates and times. Pandas can help us convert the string to a timestamp using the [to\_datetime()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html" \t "_blank) method.

Here's how we can convert the entry in our cell and check that it worked:

Let's use Pandas' to\_datetime() to convert the entire df['DATE'] column.

Excellent. Now we can start thinking about how to manipulate our data so that we get a one column per programming language. For all of that and more, I'll see you in the next lesson.



Data Manipulation: Pivoting DataFrames

**The**[**.pivot()**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html)**method**

Sometimes you want to convert your DataFrame so that each category has its own column. For example, suppose you needed to take the table below and create a separate column for each actor, where each row is the Age of the actor:

How would you do this with the DataFrame below?

1. test\_df = pd.DataFrame({'Age': ['Young', 'Young', 'Young', 'Young', 'Old', 'Old', 'Old', 'Old'],
2. 'Actor': ['Jack', 'Arnold', 'Keanu', 'Sylvester', 'Jack', 'Arnold', 'Keanu', 'Sylvester'],
3. 'Power': [100, 80, 25, 50, 99, 75, 5, 30]})
4. test\_df

The easiest way to accomplish this is by using the .pivot() method in Pandas. Try the example for yourself. The thing to understand is how to supply the correct aguments to get the desired outcome. The index are the categories for the rows. The columns are the categories for the columns. And the values are what you want in the new cells.

1. pivoted\_df = test\_df.pivot(index='Age', columns='Actor', values='Power')
2. pivoted\_df

However, there's one very important thing to notice. What happens if a value is missing? In the example above there's no value for old Sylvester. In this case, the .pivot() method will insert a NaN value.

**Mini-Challenge**

* Can you pivot the df DataFrame so that each row is a date and each column is a programming language? Store the result under a variable called reshaped\_df.
* Examine the dimensions of the reshaped DataFrame. How many rows does it have? How many columns?
* Examine the head and the tail of the DataFrame. What does it look like?
* Print out the column names.
* Count the number of entries per column.

You should get something like this:

**Solution**

Here's how you pivot our existing DataFrame to get the outcome above:

1. reshaped\_df = df.pivot(index='DATE', columns='TAG', values='POSTS')

We have 145 rows and 14 columns in the new DataFrame. Each programming language became a column and our date column became the new index (i.e., the label for the rows).

When we count the number of entries per column we see that not all languages are the same. The reason is that the .count() method excludes NaN values. When we pivoted the DataFrame the NaN values were inserted when there were no posts for a language in that month (e.g., Swift in July, 2008).

**Dealing with NaN Values**

In this case, we don't want to drop the rows that have a NaN value. Instead, we want to substitute the number 0 for each NaN value in the DataFrame. We can do this with the .fillna() method.

1. reshaped\_df.fillna(0, inplace=True)

The inplace argument means that we are updating reshaped\_df. Without this argument we would have to write something like this:

1. reshaped\_df = reshaped\_df.fillna(0)

Let's check if we successfully replaced all the NaN values in our DataFrame.

We can also check if there are any NaN values left in the entire DataFrame with this line:

1. reshaped\_df.isna().values.any()

Here we are using the .isna() method that we've used before, but we're chaining two more things: the values attribute and the any() method. This means we don't have to search through the entire DataFrame to spot if .isna() is True.

Now we're all set to create some charts and visualise our data. For all of that and more, I'll see you in the next lesson!

Data Visualisation with Matplotlib

**Matplotlib**

To create our first charts we're going to use a library called [Matplotlib](https://matplotlib.org/). There are many different libraries in Python to help us create charts and graphs. Matplotlib is an incredibly popular one and it works beautifully in combination with Pandas, so let's check it out.

First, we have to import Matplotlib.

1. import matplotlib.pyplot as plt

Let's do this at the top:

**Mini Challenge**

You can actually show a line chart for the popularity of a programming language using only a single line of code. Can you use the [.plot() documentation](https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot) to figure out how to do this? Try and plot the popularity of the Java programming language. Here's what you're aiming for:

**Solution**

All you need to do is supply the values for the horizontal axis (the x-values) and the vertical axis (the y-values) for the chart. The x-values are our dates and the y-values are the number of posts. We can supply these values to the .plot() function by position like so:

1. plt.plot(reshaped\_df.index, reshaped\_df.java)

or like so if you prefer the square bracket notation.

1. plt.plot(reshaped\_df.index, reshaped\_df['java'])

**Styling the Chart**

Let's look at a couple of methods that will help us style our chart:

.figure() - allows us to resize our chart

.xticks() - configures our x-axis

.yticks() - configures our y-axis

.xlabel() - add text to the x-axis

.ylabel() - add text to the y-axis

.ylim() - allows us to set a lower and upper bound

To make our chart larger we can provide a width (16) and a height (10) as the figsize of the figure.

1. plt.figure(figsize=(16,10))
2. plt.plot(reshaped\_df.index, reshaped\_df.java)

This will make our chart easier to see. But when we increase the size of the chart, we should also increase the fontsize of the ticks on our axes so that they remain easy to read:

Now we can add labels. Also, we're never going to get less than 0 posts, so let's set a lower limit of 0 for the y-axis with .ylim().

1. plt.xlabel('Date', fontsize=14)
2. plt.ylabel('Number of Posts', fontsize=14)
3. plt.ylim(0, 35000)

But what if we wanted to plot all the programming languages on the same chart? We don't want to type out .plot() a million times, right? We can actually just use a for-loop:

1. for column in reshaped\_df.columns:
2. plt.plot(reshaped\_df.index, reshaped\_df[column])

This will allow us to iterate over each column in the DataFrame and plot it on our chart. The final result should look like this:

Learning Points & Summary

Congratulations on completing another challenging data science project! Today we've seen how to grab some raw data and create some interesting charts using Pandas and Matplotlib. We've

* used .groupby() to explore the number of posts and entries per programming language
* converted strings to Datetime objects with to\_datetime() for easier plotting
* reshaped our DataFrame by converting categories to columns using .pivot()
* used .count() and isna().values.any() to look for NaN values in our DataFrame, which we then replaced using .fillna()
* created (multiple) line charts using .plot() with a for-loop
* styled our charts by changing the size, the labels, and the upper and lower bounds of our axis.
* added a legend to tell apart which line is which by colour
* smoothed out our time-series observations with .rolling().mean() and plotted them to better identify trends over time.