**Analyzing Amazon Order History**

**About**

We are going to analyze our Amazon data using a little Python programming. But if you have never coded anything before, don’t worry! We’re going to walk through it all together. By the end of this article, you will have written fewer than thirty lines of code, and you will have done some cool things like.

* Figured out the total amount of money you have spent on Amazon, ever.
* Found your most and least expensive orders, and your average and median order totals.
* Figured out how much you have paid in taxes, and your effective sales tax rate on Amazon.
* Visualized your spending habits day-by-day in a bar graph.
* Customized some elements of the bar graph, like size and colors.

|  |  |  |  |
| --- | --- | --- | --- |
| **Contributor Name** | **Role,Company Name** | Date | Description |
| Ajay kumar | Cloud data Engineer, Truist Finacial | 01/15/2021 | Initial document |

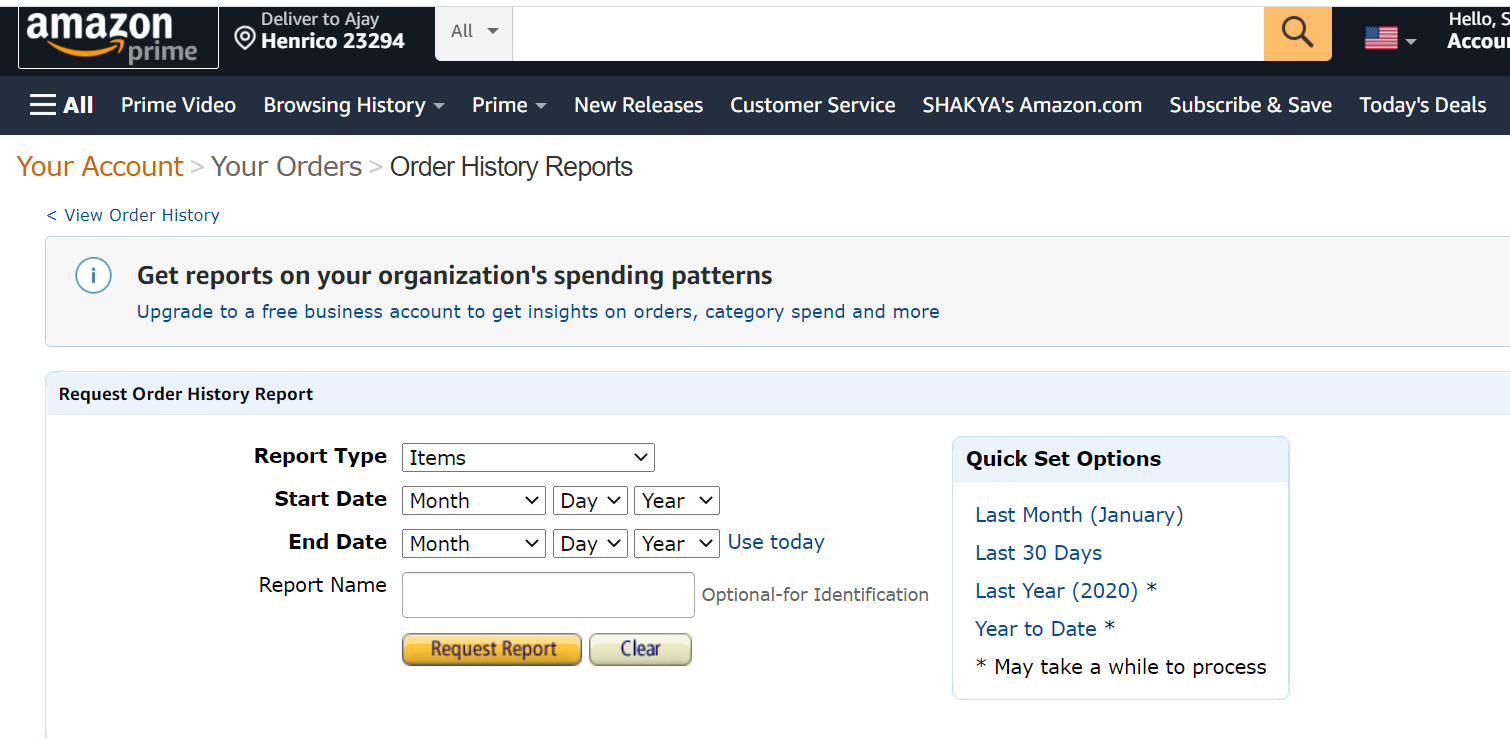
**Step1: Required Items**

1. Install Anaconda from below link.

* https://www.anaconda.com/products/individual

1. Download amazon order history file(To Download follow below steps)

* Log into Amazon.com
* Go to accounts🡪Ordering and shopping Preference🡪 select “Download order Report” then below screen will appear.



* Select “Start Date” and “End Date” and press “Request Report” , then you report will download once it ready(the process will take based on date range and order history)

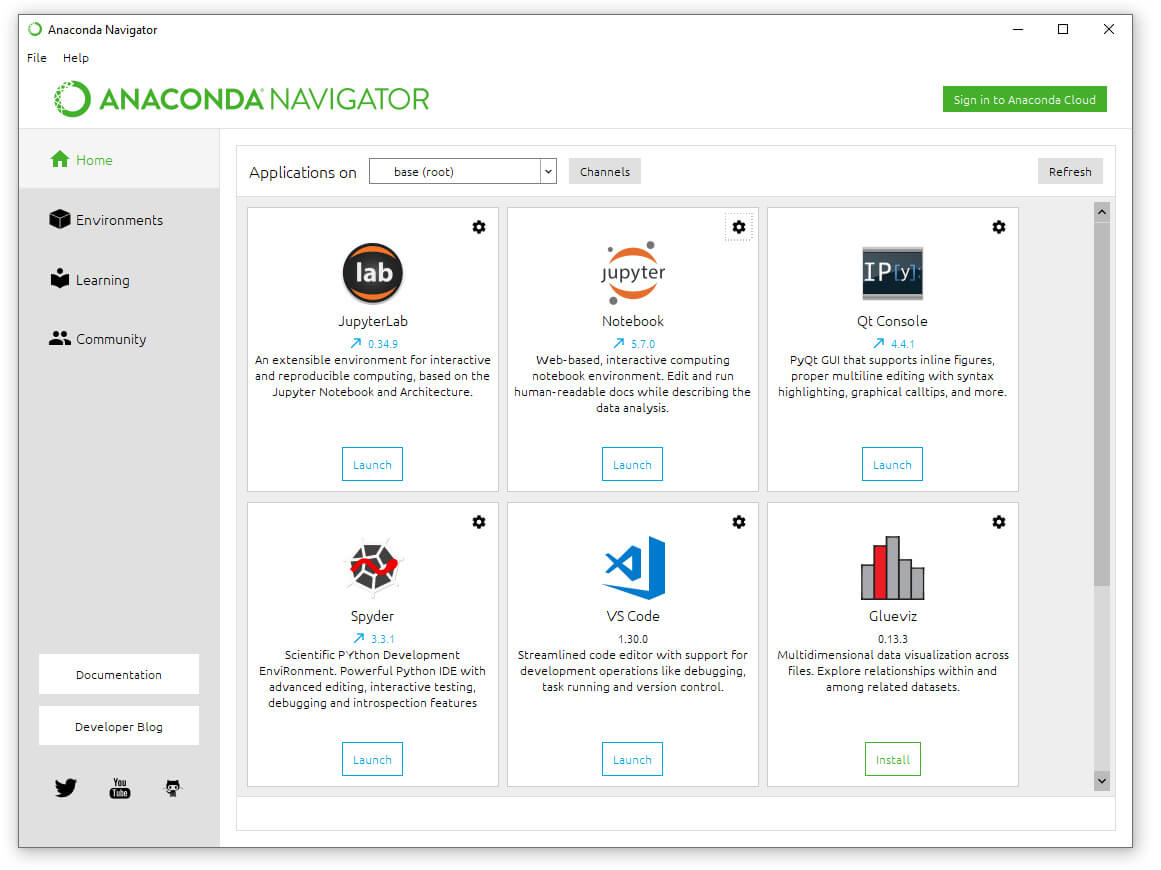
#### Step 2: Renaming and Moving the Amazon Order Data

We are almost ready to dive into a little programming and do our analysis. But first, let’s make things a little easier by renaming and moving that file we just downloaded. By default, this will probably be called something like **01\_jan\_2019\_to\_feb\_2021.csv**, but let’s rename it something simpler: **Amazon-order-History.csv.**

Next, we will move the **Amazon-order-History.csv** file to the same folder we installed Anaconda in so that we can access it easily from Jupyter Notebooks. If you did not change the default folder during the installation process, this will be **C:/Users/yourUsername/** on Windows.

Once above process is completed, we will see our report in Jupytor notebook after we launch Jupytor(how to launch Jupytor notebook will explained in step 3)

#### Step2: To Open a Jupyter Notebook

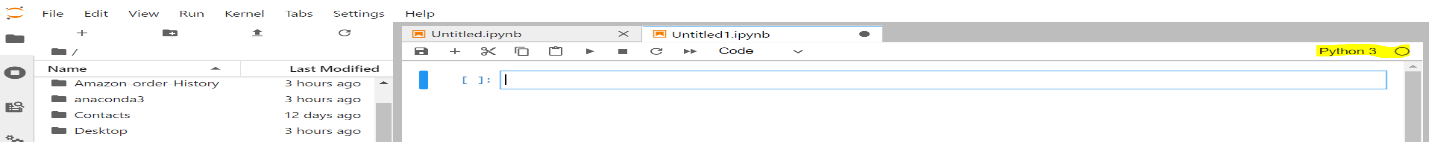
Now we are finally getting to the fun part! Open the Anaconda Navigator app. You can find this by searching for “Anaconda Navigator” on your computer. On Windows, you can also find it in the Start Menu, and on Mac it should be in your Applications folder. Once it is open, you’ll see a screen that looks like this:

Click the “Launch” button under Jupyter Notebook. That will open a new screen in your browser. On the top right of that screen click “New” and then under “Notebook” in the drop-down menu, click “Python 3”.

## How Much Have I Spent on Amazon?

Jupyter Notebooks allows us to write and run little snippets of code in cells. On the screen, below the menu and buttons, you will see a little block that says in [] with a blinking cursor. This is the first cell, and it is where we can write our code.

For this analysis, we will be using Python, a very versatile programming language that is popular for data analysis. We will also be using Python packages called pandas and matplotlib. You can think of packages as sort of like browser plug-ins — they are tools that help extend the capabilities of regular Python. Pandas and matplotlib are already installed with Anaconda, so we do not need to download or install them ourselves.



#### Step3: Getting the Data into Pandas

Our first step will be to write the code to open and look at our Amazon data, so that we can see what we are working with. We will be using pandas for most of our analysis, so our first step will be to import the panda’s package and give it the nickname **PD** so that we can refer to it more easily in our code.

Next, we will create a new variable called df (Data Frame), and then tell pandas to store our Amazon data in in that variable. A Data Frame is basically just the pandas name for a table. What we are really doing here is storing our data in a table format.

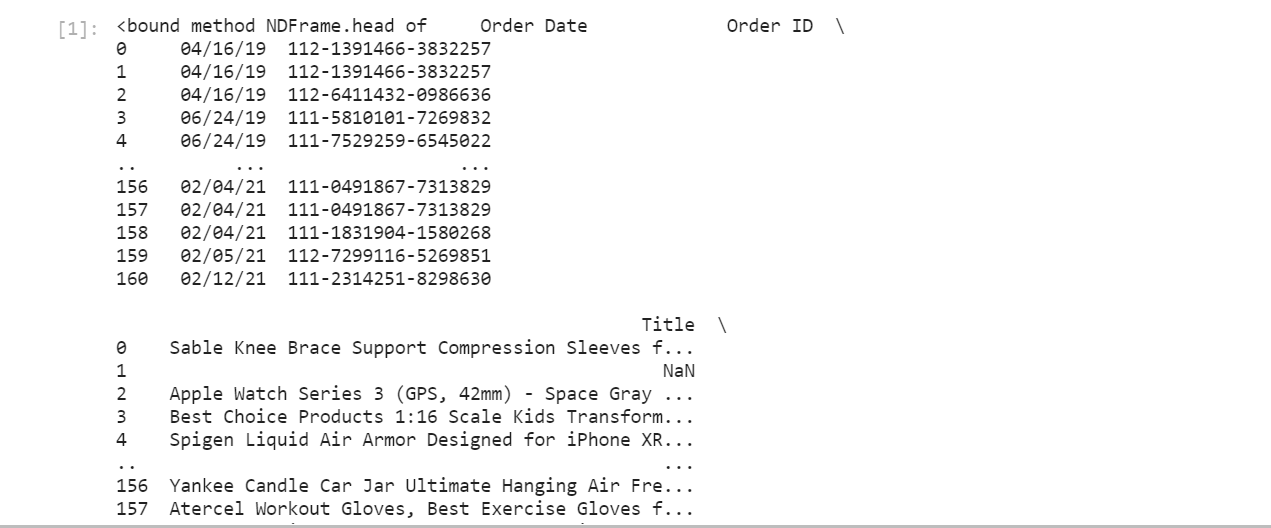
To do that, we need to tell pandas to read the .csv file as a Data frame. We will use pd. read\_csv(‘file\_name.csv’) for this. Let’s break that down piece by piece:

* PD: tells Python to use pandas to do this operation.
* .read\_csv(): tells pandas to read a csv file and store it as a Data Frame
* The 'file\_name.csv' part inside the parentheses tells pandas to read a csv file with that filename, in the same folder where Anaconda is installed.

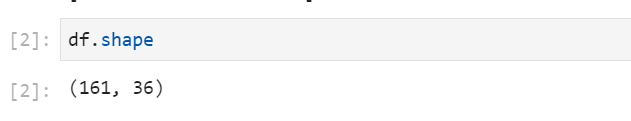
Finally, we’ll use a panda’s function called. Head () to look at the first five rows our our data, which will give us a taste of what we’re working with, and what kinds of data Amazon’s order history includes. For this, we use the syntax DataFrame.Head(), so since our Data Frame is stored in the variable df we’ll use df.head().

Here is how it looks all put together. Let’s input this code into our first Jupyter Notebook cell, and then hit the “Run” button, or use Shift + Enter (Cmd + Enter on Mac) on the keyboard to run the selected cell.





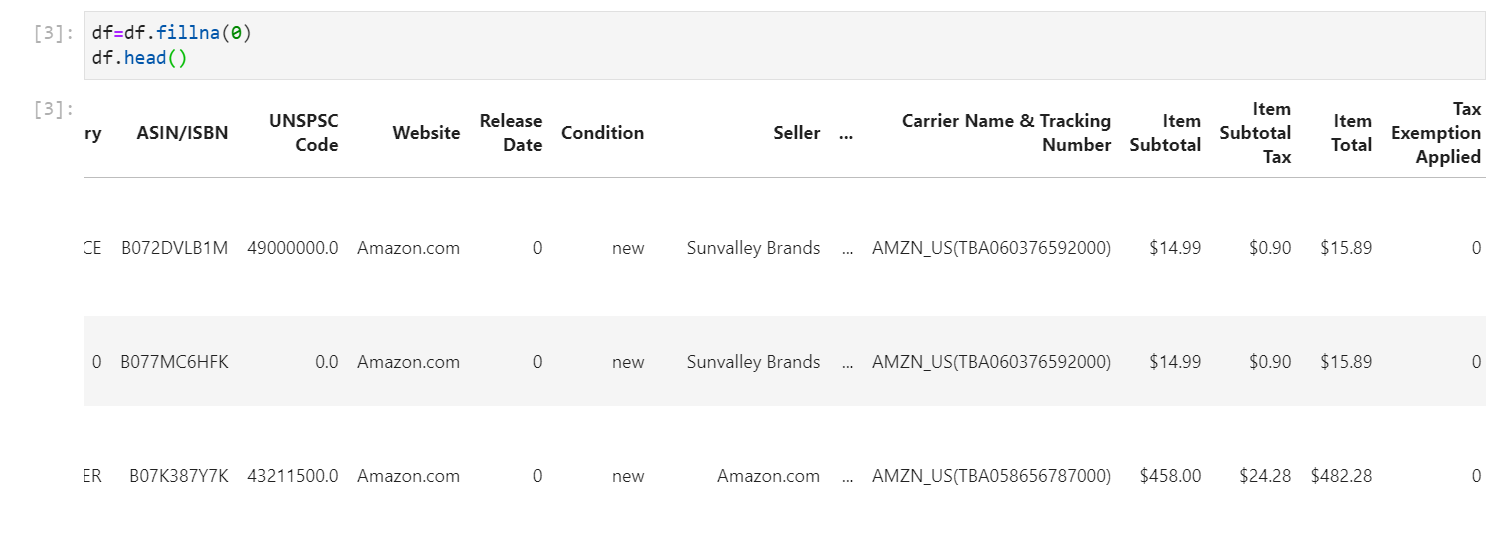
If we want to get a readout of the full size of our data set, we can use df.shape(). This gives us the dimensions of our DataFrame.



#### Step 4: Cleaning the Data

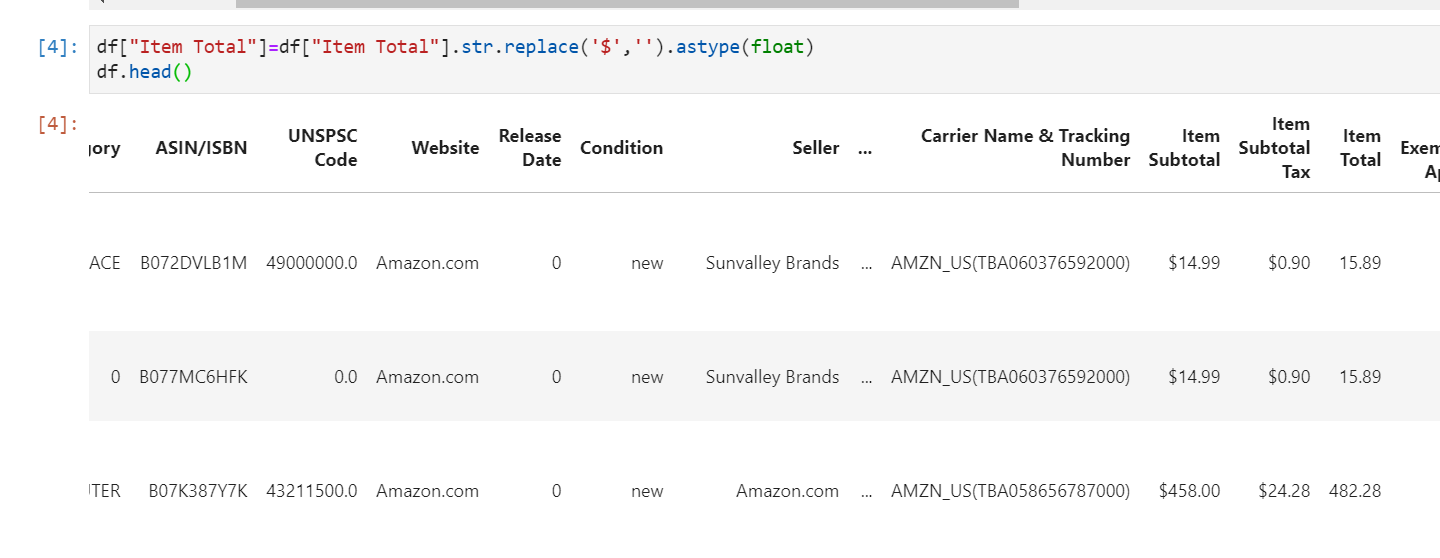
Cleaning the data is an important part of most data analysis projects, and we will need to do a little bit of data cleaning here before we can do our analysis and find out the total sum we’ve spent on Amazon.

We will use df-df. fillna(0) to replace the values and re-define our df variable to point to that new data frame. Then, we’ll use the .head() function again to make sure our changes worked.



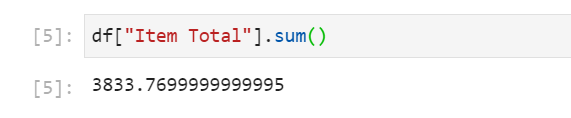
Once again, we don’t want to make these changes temporarily, we want to modify our DataFrame and then point our df variable to that new version of the DataFrame. However, since this time we’re only modifying one column of the data, we’ll use df[“Item Total”]to specify that we only want to make these changes in that column.

Now, let’s put it all together, and then once again use head () to check our work on the next line.

* 

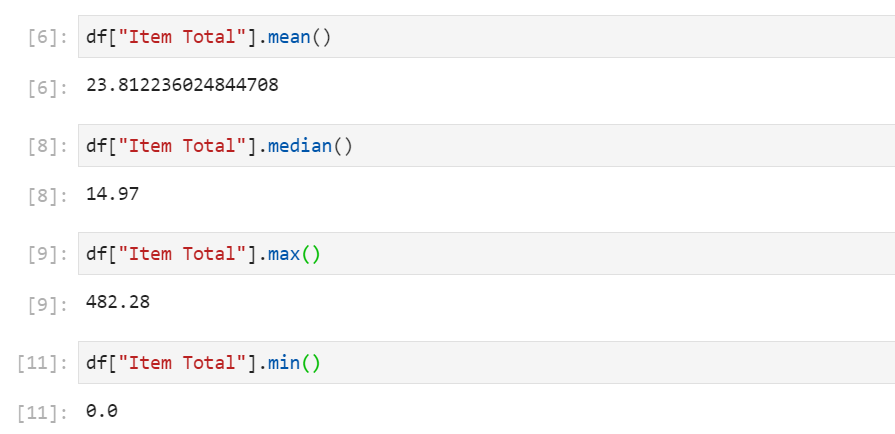
#### Calculating the Total Amount of Money Spent on Amazon

Pandas includes some basic math functions that we can use on this column. Let’s start with **.sum(),** which will give us the total of every number in this column — in other words, the total we’ve spent on Amazon.



There it is: $3833.7699. That’s how much money I have spent on Amazon from Jan 2019 to Jan 2021.

Similarly we can perform rest of mathematical operations like .mean(), .median(), .Max(), .min()

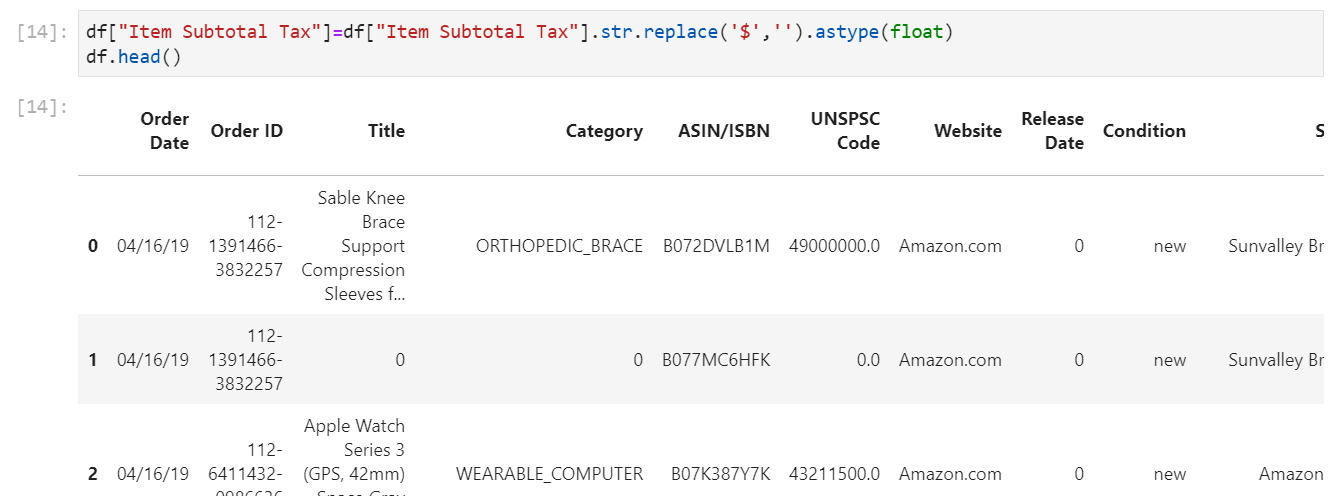


## How Much Have I Paid in Sales Tax?

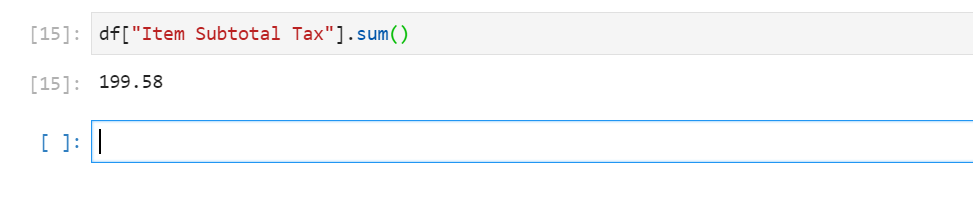
That is a lot of interesting analysis of the total price I paid for each order, but what if I wanted to look into a different column? For example, what if I wanted to see how much I have paid in sales tax?

That data’s stored in the ***Item Subtotal Tax*** column. This column has not been cleaned yet, so it has the same problems that the ***Item Total*** column did when we started.

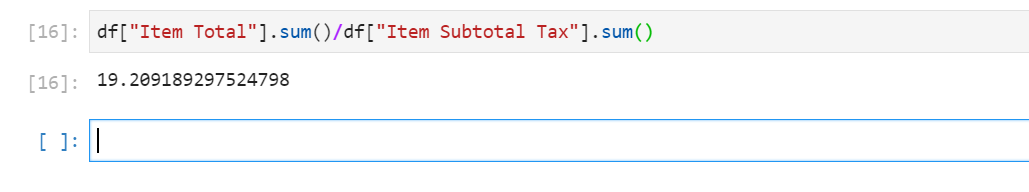
But here’s the beauty of programming: we’ve already written the code that can fix that. All we need to do is copy-paste our column-cleaning code from above and replace ***Item Total*** with ***Item Subtotal Tax***  to tell Python to perform those same operations on the ***Item Subtotal Tax*** column.



what’s the total amount of money we’ve paid in tax, according to this data set? We’ll use **.sum()** again to find out:



Apparently, about $52.61 of the total money I spent on Amazon went towards taxes. The tax rate on different items varies — we can see in the DataFrame preview above that some items did not charge any tax at all. But if we wanted to, we could figure out the overall tax rate by simply dividing our “**Item Total**” sum from our “**Item Subtotal Tax**” sum, like so:



In similar we can able to max and min and median tax we paid for amazon orders using below commands .

To know max tax amount: df[“Item Subtotal Tax”].max()

To know min tax amount: df[“Item Subtotal Tax”].min()

To know median tax amount : df[“Item Subtotal Tax”].median()

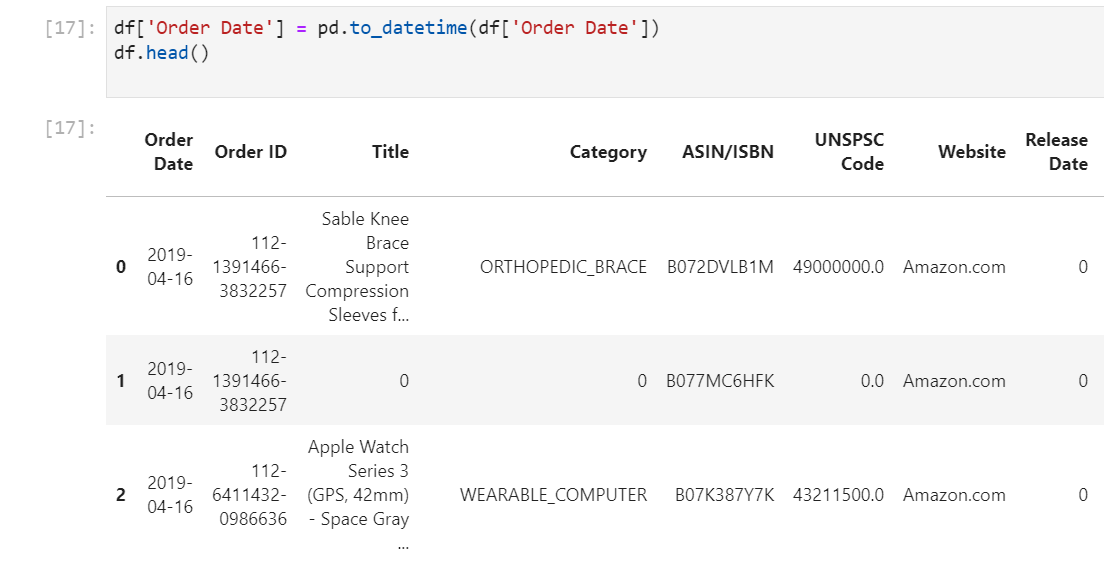
To know max tax amount : df[“Item Subtotal Tax”].mean()

## Analyzing Amazon Spending Over Time

Before we can work with dates in Python, we need to convert them to a datetime data type, so that the computer can recognize them as dates. Luckily, with pandas this is quite straightforward! We can use .**pd.to\_datetime()** and specify the column we want to modify within those brackets to tell pandas to read that column as dates.

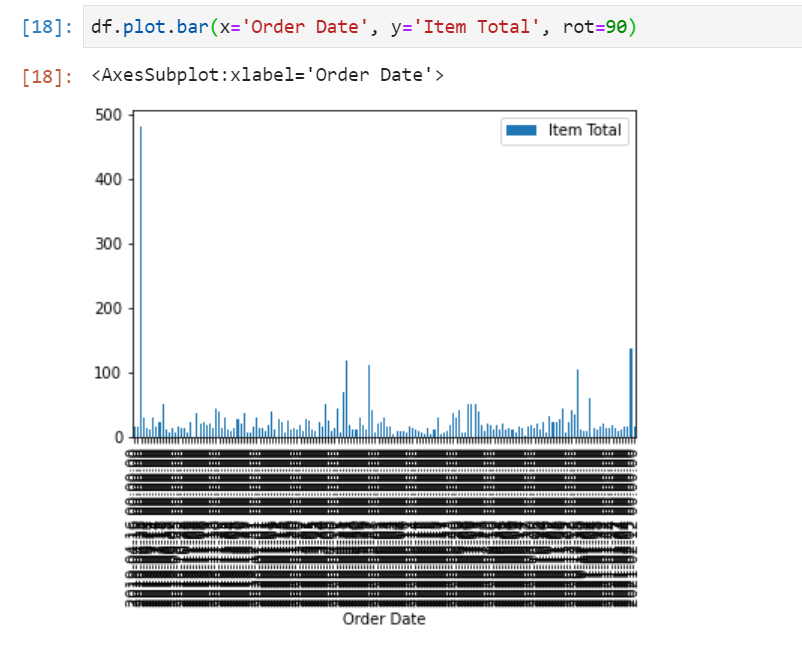
As before, we want to make sure we’re storing this change in our df dataframe, so we’ll specify that we’re modifying the **df[‘Order Date’]** column, and then assign that to the “Order Date” column in our df variable so that the old “**Order Date**” column is replaced with our changed version.

We really only need to see one row to make sure our dates have changed here, so let’s feed a custom **argument** to our old friend **df.head().** We can customize the number of rows it displays. If we put the number 1 inside those parentheses, it will display one row of the data set.



From that one row, we can see that the date formatting has changed from 2019/04/16 to 2019-04-16. That’s a good sign that our change to **datetime** has worked, and we can proceed to the next step of our analysis.

Great! We can make a basic bar graph in a pretty straightforward way using df.plot.bar(). We can pass the desired X and Y axis columns to this function as arguments, and we can add other arguments as well to do things like rotate the x-axis labels. but take a look at what happens when we chart our dataframe as-is:

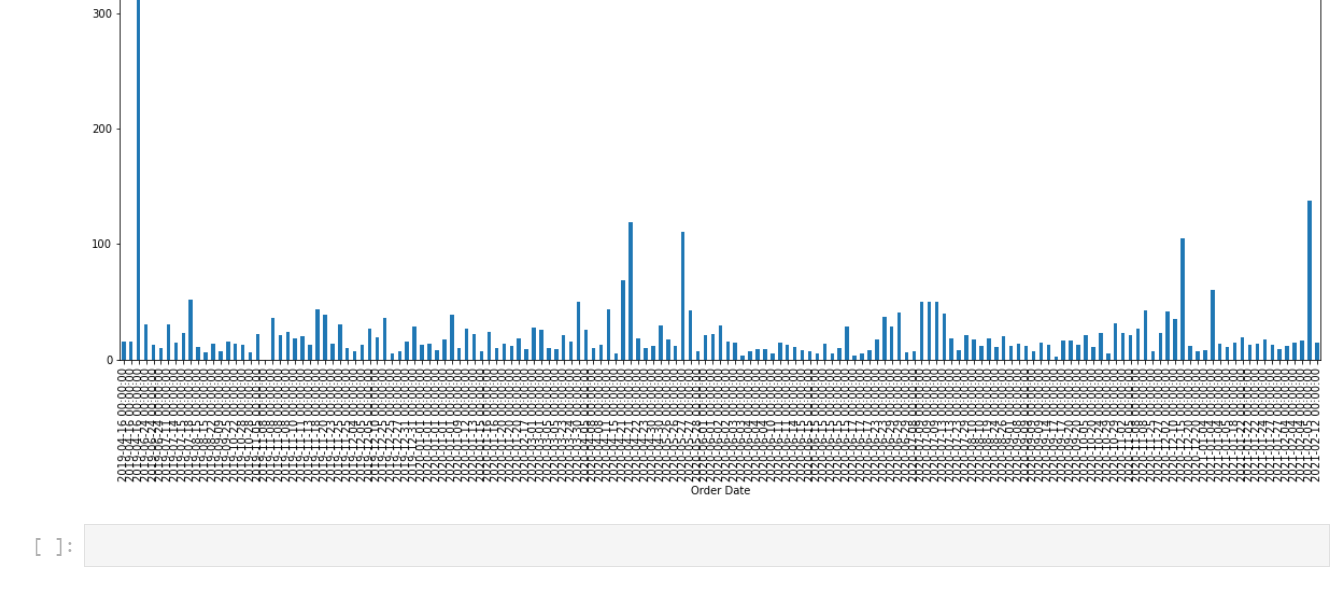


That chart is a little small, and it is hard to see whether we’ve really gotten what we want. Let’s make it bigger so that we can read the data more easily.

We can do that by adding an argument to our plotting code called figsize. This allows us to define the size of the graph (fig is short for figure) by width and height. The units it uses aren’t particularly intuitive, but (20, 10) is a good default to start from for most charts, and then we can adjust further if needed.

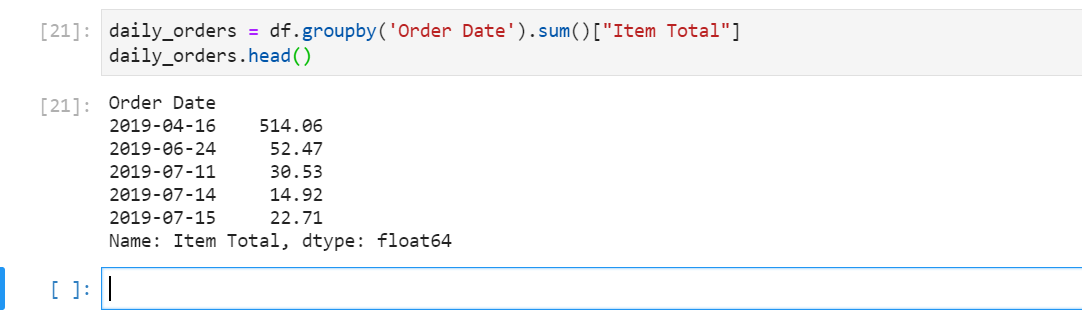
All we need to do is add figsize=(20,10) to the arguments we already have in our code. Note that each separate argument is separated with a comma.





We want to assign the result of that, which will be a pandas Series, to a new variable. Let’s call it daily\_orders.

Here’s what it looks like all together:



**Explore few more arguments to take this project further more.**

Below are few arguments try this once:

* daily\_orders.plot.bar(figsize=(20,10))
* daily\_orders.plot.bar(figsize=(20, 10), color='#61D199')