At the beginning of this course, we learned that there are two types of data visualization:

* Exploratory data visualization: we create graphs for *ourselves* to better understand and explore data.
* Explanatory data visualization: we create graphs for *others* to inform, make a point, or tell a story.

Throughout the course, we focused on explanatory data visualization and learned the following:

* How to use information design principles (familiarity and maximizing the data-ink ratio) to create better graphs for an audience.
* About the elements of a story and how to create storytelling data visualizations using Matplotlib.
* How to create visual patterns using Gestalt principles.
* How to guide the audience's attention with pre-attentive attributes.
* How to use Matplotlib built-in styles — with a case study on the FiveThirtyEight style.

To make learning more efficient, we learned about each of these topics one at a time. In this guided project, we'll go one step further and combine these skills.

The dataset we'll use describes Euro daily exchange rates between 1999 and 2021. The euro (symbolized with €) is the official currency in most of the countries of the European Union.

If the exchange rate of the euro to the US dollar is 1.5, you get 1.5 US dollars if you pay 1.0 euro (one euro has more value than one US dollar at this exchange rate).

Daria Chemkaeva put together the data set and made it available on [Kaggle](https://www.kaggle.com/lsind18/euro-exchange-daily-rates-19992020) — the data source is the European Central Bank. Note that the dataset gets regular updates — we downloaded it on January 2021.

Let's start by reading in the dataset. While we do this, start thinking about what data visualizations you might want to build using this data. You can find the solution notebook for this project [here](https://github.com/dataquestio/solutions/blob/master/Mission529Solutions.ipynb).

Instructions

1. Read in the euro-daily-hist\_1999\_2020.csv file into a pandas DataFrame — store the file into a variable named exchange\_rates.
2. Inspect the first and the last five rows to understand the structure of the dataset.
3. Use the DataFrame.info() method to learn some basic facts about the dataset:
   * What is the number of rows and columns?
   * Are there null values?
   * What is the data type of each column?

Before we start creating data visualizations, we'll need to do a bit of data cleaning. To leave you more time to practice your data visualization skills, we'll provide you with most of the code necessary for data cleaning.

Our focus in the guided part of the project will be on the exchange rate between the euro and the American dollar. If you want to explore a different currency after finishing the guided part, the code we use for cleaning will come in handy.

Below, we do the following:

* We rename the [US dollar ] and Period\Unit: columns to something easier to type — US\_dollar and Time.
* We change the Time column to a datetime data type.
* We sort the values by Time in ascending order.
* We reset the index (and drop the initial index).

exchange\_rates.rename(columns={'[US dollar ]': 'US\_dollar',

                              'Period\\Unit:': 'Time'},

                     inplace=True)

exchange\_rates['Time'] = pd.to\_datetime(exchange\_rates['Time'])

exchange\_rates.sort\_values('Time', inplace=True)

exchange\_rates.reset\_index(drop=True, inplace=True)

Let's continue the cleaning process in the next exercise.

Instructions

1. Run the code above in your notebook — exchange\_rates is the variable name we used for reading in the euro-daily-hist\_1999\_2020.csv file.
2. Isolate the Time and the US\_dollar columns. Assign them to a different variable named euro\_to\_dollar.
3. Run the Series.value\_counts() method on the US\_dollar column, and see if you notice anything wrong.
4. Drop all the rows where the - character appears in the US\_dollar column.
5. Convert the US\_dollar column to a float data type.

Now that we're finished cleaning the data, we'll generate a line plot to visualize the evolution of the euro-dollar exchange rate.

import matplotlib.pyplot as plt

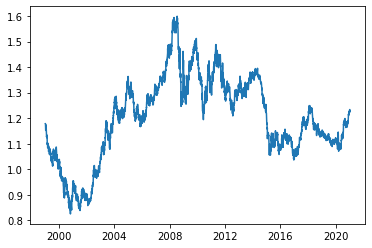
%matplotlib inline # Enables Jupyter to display graphs

​

plt.plot(euro\_to\_dollar['Time'],

        euro\_to\_dollar['US\_dollar'])

plt.show()



If we look at the line's shape, we see many small wiggles — rather than seeing a smooth line. The wiggles, however, have meaning: : they are the visual representation of the daily variation in the exchange rate. The rate goes up and down, up and down again, day to day. The rate only shows clear upward or downward trends in the longer run (months or years).

Depending on our goals, we may not want to show that daily variation on our graph. If we want to hide it and show only the long-term trends, we can use the **rolling mean** (also known as the moving average).

To understand how a rolling mean works, let's say we have ten values for ten consecutive days.

values = pd.DataFrame()

values['daily\_values'] = pd.Series(range(1,20,2))

values

|  | **daily\_values** |
| --- | --- |
| **0** | 1 |
| **1** | 3 |
| **2** | 5 |
| **3** | 7 |
| **4** | 9 |
| **5** | 11 |
| **6** | 13 |
| **7** | 15 |
| **8** | 17 |
| **9** | 19 |

The value of the second day is 3. Let's say we rather want that value to be the mean between the value of the first day and the value of the second day. The value of the first day is 1, the value of the second day is 3, and their arithmetical mean is 2.

Let's now say we want the same for each day: the value is the mean between the value of that day and the value of the previous day. For the third day, the value is the mean between the third and the second day. For the seventh day, the value is the mean between the seventh and the sixth day, and so on.

We can calculate the mean for each day using the [pandas.Series.rolling().mean() method](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.rolling.html) — the only exception is the first day, which doesn't have a previous day.

values['rolling\_mean\_2'] = values['daily\_values'].rolling(2).mean()

values

|  | **daily\_values** | **rolling\_mean\_2** |
| --- | --- | --- |
| **0** | 1 | NaN |
| **1** | 3 | 2.0 |
| **2** | 5 | 4.0 |
| **3** | 7 | 6.0 |
| **4** | 9 | 8.0 |
| **5** | 11 | 10.0 |
| **6** | 13 | 12.0 |
| **7** | 15 | 14.0 |
| **8** | 17 | 16.0 |
| **9** | 19 | 18.0 |

We calculated the rolling mean using values from two days — the current day and the previous day. We call the number of days used in the calculation the **rolling window** (or moving window). If the rolling window is three, then we calculate the mean between the value of the current day and the values of the previous two days.

values['rolling\_mean\_3'] = values['daily\_values'].rolling(3).mean()

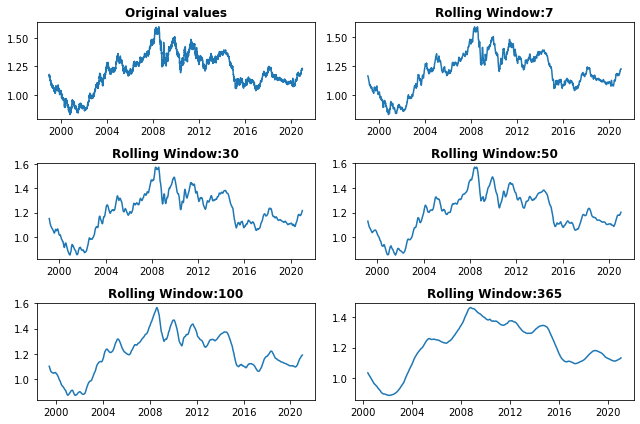
values['rolling\_mean\_5'] = values['daily\_values'].rolling(5).mean()

values

|  | **daily\_values** | **rolling\_mean\_2** | **rolling\_mean\_3** | **rolling\_mean\_5** |
| --- | --- | --- | --- | --- |
| **0** | 1 | NaN | NaN | NaN |
| **1** | 3 | 2.0 | NaN | NaN |
| **2** | 5 | 4.0 | 3.0 | NaN |
| **3** | 7 | 6.0 | 5.0 | NaN |
| **4** | 9 | 8.0 | 7.0 | 5.0 |
| **5** | 11 | 10.0 | 9.0 | 7.0 |
| **6** | 13 | 12.0 | 11.0 | 9.0 |
| **7** | 15 | 14.0 | 13.0 | 11.0 |
| **8** | 17 | 16.0 | 15.0 | 13.0 |
| **9** | 19 | 18.0 | 17.0 | 15.0 |

The rolling window can take various values — some common values include seven (one week), 30 (one month), 50 or 100 days, or 365 days (one year).

Below, we see how our line plot changes visually as we increase the rolling window. The window gets larger, and the line becomes smoother. This increases the data-ink ratio, and it can be useful if we want the audience to focus only on long-term trends.



Instructions

1. Calculate the rolling means for the US\_dollar column using a moving window of 30 days. Add the rolling means to a new column named rolling\_mean.

On this screen, we're going to spend some time coming up with an idea for the graph we want to build. Our goal is to create a storytelling data visualization using the data we have.

To create a story, remember that we need to arrange our data into a series of events that show change.

Here are a few story ideas for our data:

* We show how the euro-dollar rate has changed during the coronavirus pandemic. We can show the 2020 data and the 2016-2019 data as a baseline. We can use a line plot.
* We show how the euro-dollar rate changed during the 2007-2008 financial crisis. We can also show the data for 2006 and 2009 for comparison. We can use a line plot.
* We show comparatively how the euro-dollar rate changed under the last three US presidents (George W. Bush (2001-2009), Barack Obama (2009-2017), and Donald Trump (2017-2021)). We can use a line plot.

These are just a few ideas — you should think of a story angle that you find interesting.

Don't become frustrated if you don't have an idea — creativity is unpredictable. Sometimes you can have ten ideas in a minute, and sometimes you won't have any ideas in a week.

If you don't have any ideas, you can choose one from the examples above. What is important here is to spend at least a few minutes brainstorming graph ideas.

Instructions

1. Note one idea regarding the storytelling data visualization you want to build.

Now that we know what story we want to tell, we're going to think more about how the graph is going to look.

First, we need to choose the right graph type. The graph should give our story the visual form it needs — we don't want to choose a histogram to visually represent a time series.

Equally important, we need to choose something that our audience will be familiar with. To simplify your work, you can choose any audience you want for your graph.

Next, we're going to sketch the graph — we do this before writing the code. In sketching the graph, it's important to use some of the principles we've learned so far:

* Maximize the data-ink ratio.
* Create visual patterns using Gestalt principles.
* Guide the audience's attention using pre-attentive attributes.

These design principles can help you in two ways:

* They generate design options.
* They help you choose among those options.

We advise using pen and paper to sketch the graph because it gives you more flexibility. Alternatively, you can make the sketch using a graphics editor.

This is another step that relies on creativity, so we're going to say it again: creativity is unpredictable, so don't become frustrated. If you don't come up with an idea, we have a few suggestions on the next screen.

Instructions

1. Using pen and paper, sketch the graph you want to create.

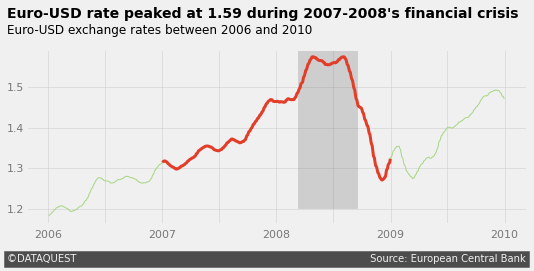
Now that we have a good idea of the story we want to tell and how it's going to look, we'll start coding.

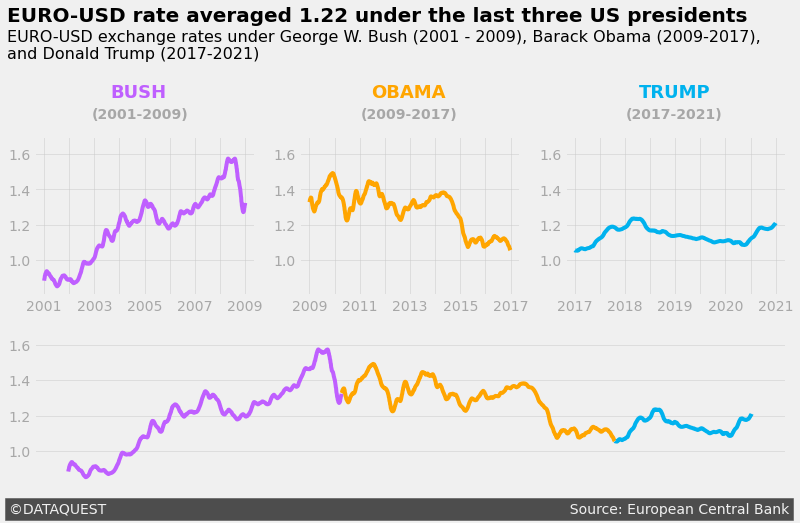
Writing the code yourself may feel intimidating. However, you don't have to create the best graph in the world. The point here is to get better at building something yourself.

Here are a few tips on how to approach this:

* If you don't remember something you've already learned (a common issue when we're learning), read that part of the lesson again.
* Google any Matplotlib issue you may have. For better results, start your queries with the word "matplotlib":
  + "matplotlib how to remove tick labels"
  + "matplotlib how to remove only the right and the top axes"
* Don't hesitate to read Matplotlib documentation or look through their [examples gallery](https://matplotlib.org/gallery/index.html).
* Matplotlib doesn't have a function for everything — try to improvise by separately creating the constituent parts of what you want to build.
* Where there is code, there will be errors — google any errors you can't fix.
* Reach out to the Dataquest community if you get stuck.

To give you an example (and hopefully some inspiration!), we've built the two storytelling data visualizations below — you'll find the code in the [solution notebook](https://github.com/mircealex/solutions/blob/master/Mission529Solutions.ipynb). If you don't feel like building something yourself, you can try to reproduce these two graphs instead.





If you're going to run the code in the solution notebook, and the lines on your plots disappear, or you get a ValueError: Image size of AxB pixels is too large, see this [Community thread](https://community.dataquest.io/t/getting-an-image-size-error-in-guided-project-storytelling-data-visualization-on-exchange-rates/552213/2?u=alex).

Instructions

1. Code the graph you sketched in the previous step. Take your time, and don't become frustrated. Good luck!

In this project, we created a storytelling data visualization using the following workflow:

* We started by exploring and cleaning the data.
* We brainstormed ideas for storytelling data visualizations and chose one.
* We sketched our data visualization using pen and paper.
* We coded the sketched data visualization.

Next steps include the following:

* Creating a graph for a different currency.
* Creating a graph for multiple currencies — perhaps you can compare their evolution.
* Making your project portfolio-ready by following this [style guide](https://www.dataquest.io/blog/data-science-project-style-guide/).

Curious to see what other students have done on this project? [Head over to our Community to check them out](https://community.dataquest.io/tags/c/social/share/49/350). While you are there, please share your own feedback!

And of course, we welcome you to share your own project and show off your hard work. Head over to our Community to [share your finished Guided Project](https://community.dataquest.io/tags/c/social/share/49/529)!

This is the end of the course — congratulations!