TECHACADEMY

Data Science Project Guide: Covid-19

TechAcademy e.V.

Summer Term 2020

Contents

1 Welcome!

In the first few chapters you will be introduced to the basics of the `R` and `Python` tracks respectively and you will find helpful explanations to questions you might have in the beginning of your coding journey. There will be a quick introduction to the Data Science track so that you can get started with the project quickly. So let's get started with the basics!

In all tracks you will work on your project in small groups of fellow students. This not only helps you get the project done faster, it also helps make your results even better. Our experience shows: The different backgrounds of the members and discussing different opinions and ideas will produce the best results. Besides, it is of course more fun to work on a project together than to code alone!
The groups can consist of a maximum of three members. You can choose your two teammates independently, we won't interfere with your arrangements. It is important that all group members complete the same level of difficulty (beginner or advanced), since the tasks are different in scope for each level. We explicitly encourage you to collaborate with students from different departments. This not only allows you to get to know people from other departments, but may even give you a whole new perspective on the project and tasks. When submitting it is important to note: for a certificate, each person must submit the project individually. However, this can be identical within your group. You can get more information at our first Coding Meetup on **May 20, 2020**.

Every semester, we at TechAcademy think hard about a new project that is suited for you to learn data science. This semester, the topic choice was rather straightforward. Covid-19 is the dominating issue for our society at the time of writing this project in March and April 2020. We developed this case study as the crisis was unfolding in real time. So the recency of this topic comes at a small cost to you – the data is changing rapidly and so are the results and conclusions. Not every textbook method works perfectly well on this data set – but that's how it is in real life. We believe that you will learn a great deal about real life Data Science with this very timely topic. Maybe even more than you would with a polished textbook case study and data set.

This case study and the associated project guide was developed and written entirely from scratch by TechAcademy's Data Science team. Lukas Jürgensmeier and Lara Zaremba developed the project in `R`, while Felix Schneider and Manuel Mair am Tinkhof developed it in `Python`. We thank Jonathan Ratschat for providing valuable feedback on earlier versions of the project.

2 What's Data Science and How Do I Do It?

Data science is a multi-layered field in which the use of the latest machine learning methods is only a sub-area. To get there, you'll need many steps before – from collecting to manipulating to exploring the data. And eventually, you will need to somehow communicate your findings.

But first things first. To analyze data, it must first be obtained. You need to know where to obtain it and how to integrate it in your respective tools. The data is rarely available as it would

be needed for further processing. Familiarizing yourself with the information available, cleaning it up and processing it into the desired formats that can be read by humans and machines are important steps that often make up a large part of the work.

Before the obtained data can be analyzed, the right tool must be selected and mastered: the programming language. The most often used languages for Data Science are `R`, which was explicitly developed for statistics, and `Python`, which is characterized by its additional versatility. The data scientist does not have to be a perfect software developer who masters every detail and paradigm, but the competent handling of syntax and idiosyncrasies is essential. There are some well-developed method collections, so-called packages or libraries, which provide a lot of functionality. The use of these collections also has to be learned and mastered. Once all of this is achieved, the data can finally be analyzed. Here too, it is important to know and understand the multitude of statistical approaches in order to be able to choose the right method for the problem at hand. The newest, best, most beautiful neural network is not always the solution for everything.
One step is still missing in the data science process: understanding and communicating the results. The results are often not spontaneously intuitive or sometimes even surprising. Here, the specific expertise and creativity can be played out, especially in the visualization.

2.1 What's R?

`R` is a programming language that was developed by statisticians in the early 1990s for use in the calculation and visualization of statistical applications. A lot has happened since then and by now, `R` is one of the most widely used programming languages in the field of data science. Code in `R` does not have to be compiled, but can be used interactively and dynamically. This makes it possible to quickly gain basic knowledge about existing data and to display it graphically.
`R` offers much more than just programming, but also a complete system for solving statistical problems. A large number of packages and interfaces are available, with which the functionality can be expanded and integration into other applications is made possible.

2.1.1 RStudio Cloud

Before you can use `R`, you usually have to install some separate programs locally on your computer. Typically, you first install a "raw" version of `R`. In theory, you can then already start programming. However, it is very difficult to carry out an entire project with it. That's why there is RStudio, an Integrated Development Environment (IDE) for `R`. This includes many essential features that simplify programming with `R`. Among other things, an auto-completion of your code, a nicely structured user interface and many expansion options.

Experience has shown that installing `R` and RStudio locally takes some effort. Fortunately, RStudio also has a cloud solution that eliminates these steps: RStudio Cloud. There it is possible to edit your project in exactly the same IDE in the browser without any prior installations. You can also easily switch your project from private to public and give your team an insight into your code via a link or by giving them access to the workspace directly. In this way you are able to easily exchange ideas within your team.

We will introduce RStudio Cloud and unlock access to our workspace on our first Coding Meetup. Until then, focus on learning the hard skills of programming with your courses on DataCamp. This brings us to your curriculum in the next section.

2.1.2 Curriculum

The following list shows the required DataCamp courses for the Data Science with `R` Track. As a beginner, please stick to the courses of the "beginner" program, ambitious beginners can of course also take the advanced courses afterwards. However, the courses should be worked through in the order in which they are listed.
The same applies to the advanced courses. Here, too, the specified courses should be processed in the given order. Since it can of course happen that you have already mastered the topics of an advanced course, individual courses can be replaced. The topics of the advanced courses are given in key points. If these key points seem familiar to you, then take a look at the table of contents of the corresponding DataCamp course. If you are convinced that this course does not provide any added value for you, it can be replaced by one of the courses in the "Exchange Pool" (see list). However, this exchange course should not be processed until all other courses in the advanced course have been completed.

Both beginners and advanced learners must have completed at least two thirds of the curriculum in order to receive the certificate. For the beginners this means at least up to the course "Data Visualization with ggplot2 (Part 1)" and for the advanced at least up to "Supervised Learning in `R`: Classification". In addition, at least two thirds of the project tasks must have been completed.

R Fundamentals (Beginner)

- Introduction to R (4h)
- Intermediate R (6h)
- Introduction to Importing Data in R (3h)
- Cleaning Data in R (4h)
- Data Manipulation with dplyr (4h)
- Data Visualization with ggplot2 (Part1) (5h)
- Exploratory Data Analysis in R (4h)
- Correlation and Regression in R (4h)
- Multiple and Logistic Regression in R (4h)

Machine Learning Fundamentals in R (Advanced)

- Intermediate R (6h): conditionals, loops, functions, apply
- Introduction to Importing Data in R (3h): utils, readr, data.table, XLConnect
- Cleaning Data in R (4h): raw data, tidying & preparing data
- Importing & Cleaning Data in R: Case Studies (4h): case studies
- Data Visualization with ggplot2 (Part1) (5h): aesthetics, geometries, qplot
- Supervised Learning in R: Classification (4h): kNN, naive bayes, logistic regression, classification trees
- Supervised learning in R: Regression (4h): linear & non-linear regression, tree-based methods
- Unsupervised Learning in R (4h): k-means, clustering, dimensionality reduction
- Machine Learning with caret in R (4h): train()-function, cross-validation, auc

Data Science R (Advanced) – Exchange Pool

- Data Visualization with ggplot2 (Part 2) (5h)
- Interactive Maps with leaflet in R (4h)
- Machine Learning in Tidyverse (5h)
- Writing Efficient R Code (4h)
- Support Vector Machines in R (4h)
- Supervised Learning in R: Case Studies (4h)
- Optimizing R Code with Rcpp (4h)

2.1.3 Links

- RStudio Cheat Sheets: https://rstudio.cloud/learn/cheat-sheets
- RMarkdown Explanation (to document your analyses): https://rmarkdown.rstudio.com/lesson-1.html
- StackOverflow (forum for all kinds of coding questions): https://stackoverflow.com/
- CrossValidated (Statistics and Data Science forum): https://stats.stackexchange.com/

2.2 What's Python?

`Python` is a dynamic programming language. The code is executed in the interpreter, which means that the code does not first have to be compiled. This makes `Python` very easy and quick to use. The good usability, easy readability and simple structuring were and still are core ideas in the development of this programming language. Basically, you can use `Python` to program according to any paradigm, whereby structured and object-oriented programming is easiest due to the structure of the language, but functional or aspect-oriented programming is

also possible. These options give users great freedom to design projects the way they want, but also great freedom to write code that is difficult to understand and confusing. For this reason, certain standards that are based on the so-called `Python` Enhancement Proposals (PEP) have developed over the decades.

2.2.1 Anaconda and Jupyter

Before you can use `Python`, it must be installed on the computer. `Python` is already installed on Linux and Unix systems (such as macOS), but often it is an older version. Since there are differences in the handling of `Python` version 2 – which is not longer supported anymore – and version 3, we decided to work with version 3.6 or higher.

One of the easiest ways to get both `Python` and most of the best known programming libraries is to install Anaconda. There are detailed explanations for the installation on all operating systems on the website of the provider.

With Anaconda installed, all you have to do is open the Anaconda Navigator and you're ready to go. There are two ways to get started: Spyder or Jupyter. Spyder is the integrated development environment (IDE) for `Python` and offers all possibilities from syntax highlighting to debugging (links to tutorials below). The other option is to use Jupyter or Jupyter notebooks. It is an internet technology based interface for executing commands. The big advantage of this is that you can quickly write short code pieces and try them out interactively without writing an entire executable program. Now you can get started! If you have not worked with Jupyter before, we recommend that you complete the course on DataCamp (https://www.datacamp.com/projects/33) first. There you will get to know many tips and tricks that will make your workflow with Jupyter much easier.
In order to make your work and, above all, the collaboration easier, we are providing you with a platform that contains a Jupyter environment with the necessary libraries, as well as all data necessary for the project. There you will also find a brief explanation in the form of a running Jupyter notebook and introductions to the work with the data sets we have selected. Using a regular folder structure, you can quickly navigate to the folder of your group and your personal folder, in which the necessary files are stored for a smooth start.

We will introduce this environment and unlock access to it during our first Coding Meetup. Until then, focus on learning the hard skills of programming with your courses on DataCamp. This brings us to your curriculum in the next section.

2.2.2 Curriculum

The following list shows the DataCamp courses for the `Python` data science track. As a beginner, please follow the courses for the beginner level. These should be processed in the order in which they are listed.
The same applies to the advanced courses. Here, too, the specified courses should be processed in the given order. Since it can of course happen that you have already mastered the topics of an advanced course, individual courses can be replaced. The topics of the advanced courses are

given in brief. If these key points seem familiar to you, then take a look at the table of contents of the corresponding DataCamp course. If you are convinced that this course does not provide any added value for you, it can be replaced by one of the courses in the "Exchange Pool" (see list). However, this course should not be processed until all other courses in the intermediate `Python` course have been completed.

Both beginners and advanced learners must have completed at least two thirds of the curriculum in order to receive the certificate. For beginners this means at least up to the course "Manipulating DataFrames with Pandas" and for advanced learners at least up to the "Project: Bitcoin Cryptocurrency Market". In addition, at least two thirds of the project tasks have to be completed.

Python Fundamentals (Beginner)

- Introduction to Data Science in Python (4h)
- Intermediate Python (4h)
- Python for Data Science Toolbox (Part 1) (3h)
- Introduction to Data Visualization with Matplotlib (4h)
- Manipulating DataFrames with pandas (4h)
- Merging DataFrames with pandas (4h)
- Exploratory Data Analysis in Phyton (4h)
- Introduction to DataCamp Projects (2h)
- Introduction to Linear Modeling in Python (4h)

Data Science with Python (Advanced)

- Intermediate Python (4h): Matplotlib, Dict, Pandas, Loops
- Python Data Science Toolbox (Part 1) (3h): Default arguments, Lambdas, Error handling
- Python Data Science Toolbox (Part 2) (4h): Iterators, generators, List comprehension
- Cleaning Data in Python (4h): Using pandas for Data cleaning
- Exploring the Bitcoin Cryptocurrency Market (3h): Small project
- Exploratory Data Analysis in Phyton (4h): How to start a data analysis
- Introduction to Linear Modeling in Python (4h): Linear Regression, sklearn
- Supervised Learning with Scikit-Learn (4h): Classification, Regression, Tuning
- Linear Classifiers in Python (4h): Logistic regression, SVM, Loss functions

Data Science with Python (Advanced) - Exchange Pool

- TV, Halftime Shows and the Big Game (4h)
- Interactive Data Visualization with Bokeh (4h)
- Time Series Analysis (4h)
- Machine Learning for Time Series Data in Python (4h)
- Advanced Deep Learning with Keras (4h)
- Data Visualization with Seaborn (4h)
- Web Scraping in Python (4h)
- Writing Efficient Python Code (4h)
- Unsupervised Learning in Python (4h)
- Writing Efficient Code with pandas (4h)
- Introduction to Deep Learning in Python (4h)
- ARIMA Models in Python (4h)

2.2.3 Links

Official Tutorials/Documentation:

- https://docs.python.org/3/tutorial/index.html
- https://jupyter.org/documentation

Further Explanations:

- https://pythonprogramming.net/
- https://automatetheboringstuff.com/
- https://www.reddit.com/r/learnpython
- https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook

2.3 Your Data Science Project

2.3.1 Coding Meetups and Requirements

Now that you have learned the theoretical foundation in the DataCamp courses, you can put your skills into practice. We have put together a project for you based on real data sets. You can read about the details in the following chapters of this project guide.
Of course, we will also go into more detail about the project and the tools that go with it. We will discuss everything you need to know during the first Coding Meetup, which will take place on **May 20, 2020**. After that, the work on the project will officially begin. You can find the exact project tasks together with further explanations and hints in the following chapters.

To receive the certificate, it is essential that you have solved at least two thirds of the "Exploratory Data Analysis" part of the project. For the advanced participants, the entire "forecast using statistical models" part is added. In addition, two thirds of the respective curriculum on DataCamp must be completed. You can find more detailed information on this in the "Curriculum" section of the respective programming language above.

2.3.2 Data Sources

For this project, we use several publicly available data sources. The Covid-19 infection data is provided by the Johns Hopkins University Center for Systems Science and Engineering (JHU CSSE). They provide an accessible, daily updated GitHub repository with worldwide case data.

For Google search data, we use Google Trends, a very interesting free service. If you're interested in diving further into this data source, you can easily download data from this website or directly via `R` or `Python` packages.

Our stock market data comes from Yahoo! Finance, which is also easily downloadable directly into your programming environment with convenient packages in both languages.

3 Introduction to Your Project

3.1 Purpose of the Project Guide

Welcome to the project guide for your TechAcademy data science project! This document will guide you through the different steps of your project and will provide you with useful hints along the way. However, it is not a detailed step by step manual, because we felt like it was important that you develop the skills of coming up with your own way of solving different tasks. This is a great way to apply the knowledge and tools you have acquired in Data Camp.

It might happen that you don't know how to solve a task. This is a normal part of the coding process, so don't worry. It is part of the learning experience and we provided you with helpful tips throughout this guide. We also included pictures of what your results could look like. They are meant to be a useful guidance so that you know what you are working towards. Your plots won't need to look exactly the same way. The date we compiled the plots and included them in this guide was the April 12, 2020, so with the new data your plots will look different anyway. Furthermore, you can find helpful links in the introductory chapters, where your questions might already have been answered. If not, and in the unlikely case that even Google can't help you, the TechAcademy mentors will help you via Slack or directly during the coding meetups.

At the end of the project guide you will find an overview of all tasks that have to be completed, depending on your track (beginner/advanced). You can use this list to check which tasks still need to be completed and which tasks are relevant for your track.

3.2 What is this Project About?

The coronavirus, or COVID-19, is the topic you have probably heard most of in these past few months. What was especially striking to see is the fact that people talked a lot about coronavirus statistics and instead of bringing clarity to the discussion, we all got a little more confused. Some people said it's not more deadly than the flu, others said it was 10 times as deadly. Sometimes politicians confidently stated that the situation was under control and the next day schools were closed and people started buying toilet paper like their life depended on it. In times like these it is especially useful to be able to evaluate data by yourself and to see for yourself what is happening. Fortunately, there is a lot of data on the coronavirus available, so all that is left to do is to learn how to handle it and how to interpret it in a meaningful way.

In analogy to the typical data science workflow, we split this project into two parts. First you are going to learn how to perform an Exploratory Data Analysis (EDA). You will have a closer look at the data, transform it and then get to know the different variables and what they look like in different types of visualizations. Beginners will have completed the project after this, but it will be beneficial to also try and work on the next part: In the second part of the project you will come up with a model that will predict the coronavirus development as accurately as possible. You are going to start with a linear regression model, which you modify as you please and then you can explore all the other possibilities of modeling and predicting data.

But first things first: What exactly is EDA and what can you achieve with it?

3.3 Exploratory Data Analysis – getting to know the data set

As a first step you will get to know the data set. This means you will describe the data. A crucial part of data science is to familiarize yourself with the data set. What variables are contained in the data set and how are they related? You can answer these questions easily by creating plots with the data.

This first part of the project is structured in a way that lets you get to know the data thoroughly by completing the given tasks one after the other. As a beginner, you can stop after this part, because you will have fulfilled the necessary requirements. However, if this first part inspires you to learn more, we encourage you to also work on the second part.

This project guide is structured in the following format. Since the concept of Data Science is independent of specific programming languages, we will describe the general approach in this part of the text. After you got the overall concept and understood the task we are asking you to do, you will find language-specific tips and tricks in visually separated boxes. If you decided to participate in our program in `R`, you only need to look at those boxes. Conversely, you only need to look at the `Python` boxes if you are coding in that language. From time to time it might be interesting to check out the other language – though you can do the same in both, they sometimes have a different approach to the identical problem. It makes sense that you complete the first few beginner chapters mentioned in the introductory chapter. We recommend that you finish the courses at least until and including "Exploratory Data Analysis" for both tracks.

3.4 Prediction – Apply Statistical Methods

This part is mainly for the advanced TechAcademy participants. If you are a beginner and you were able to complete the first part without too many difficulties, we highly recommend trying to do the second part as well. Statistical models are a major part of data science and this is your chance of developing skills in this area.

You got to know the data in the first part and you should be familiar with it so that it is now possible to use it to make predictions about the development of the coronavirus in the future. After having completed the second part, you will send us your predictions and we will then check how accurate your model was. The best model will win!

For this part of the project we recommend the advanced courses mentioned in the introductory chapter. Please note that there are more courses available so if you want to extend your skills even further, feel free and complete more courses on the topics that interest you. We recommend that you finish the courses at least until and including *Unsupervised Learning in Python* for the `Python` track and *Machine Learning Toolbox* for the `R` track.

Ready? After getting a first impression of what this project is all about, let's get started!

4 Exploratory Data Analysis

Before you can dive into the data, set up your programming environment. This will be the place where the magic happens – all your coding will take place there.

In your workspace on rstudio.cloud we have already uploaded an "assignment" for you (Template Coronavirus). When you create a new project within the workspace "Class of '20 | TechAcademy | Data Science with R", your own workspace will open up. We've already made some arrangements for you: The data sets that you will be working with throughout the project are already available in your working directory. We also created an RMarkdown file, with which you will be able to create a detailed report of your project. You can convert this file into a PDF document when you have completed the documentation. Open the file "Markdown_Corona.Rmd" and see for yourself!

On JupyterHub you will find the folder "coronavirus". There, you will find a folder with your name – please create and store your Jupyter notebooks inside there. All the necessary data is stored in the folder "data" which is also located in the "coronavirus" folder.

4.1 Import, Clean and Transform the Data for Your First Plot

In this section you will get to apply the basics of working with data: You will import, clean and transform the data so that you will then be able to visualize it in the next step.

Visualization is one of the most helpful methods to get a feeling for the relation of the various variables of the data set. As already mentioned, this process is called Exploratory Data Analysis (EDA).

Did you know that Google provides quite extensive public access to the relative number of search queries over time? You can simply type in a keyword and Google returns the public's relative interest in that topic over time. First have a look at the Google Trends website for yourself. An interesting way to ease into the whole topic of the coronavirus outbreak is to look at a plot of the Google Trends development with respect to the virus. To visualize the number of people searching for "coronavirus" online, first import the Google Trends data set into your workspace and name it accordingly. We have already downloaded the data for you. It is always helpful to give meaningful names to your data sets and variables so that you don't get confused later on when you have several different data sets and variables in your workspace.

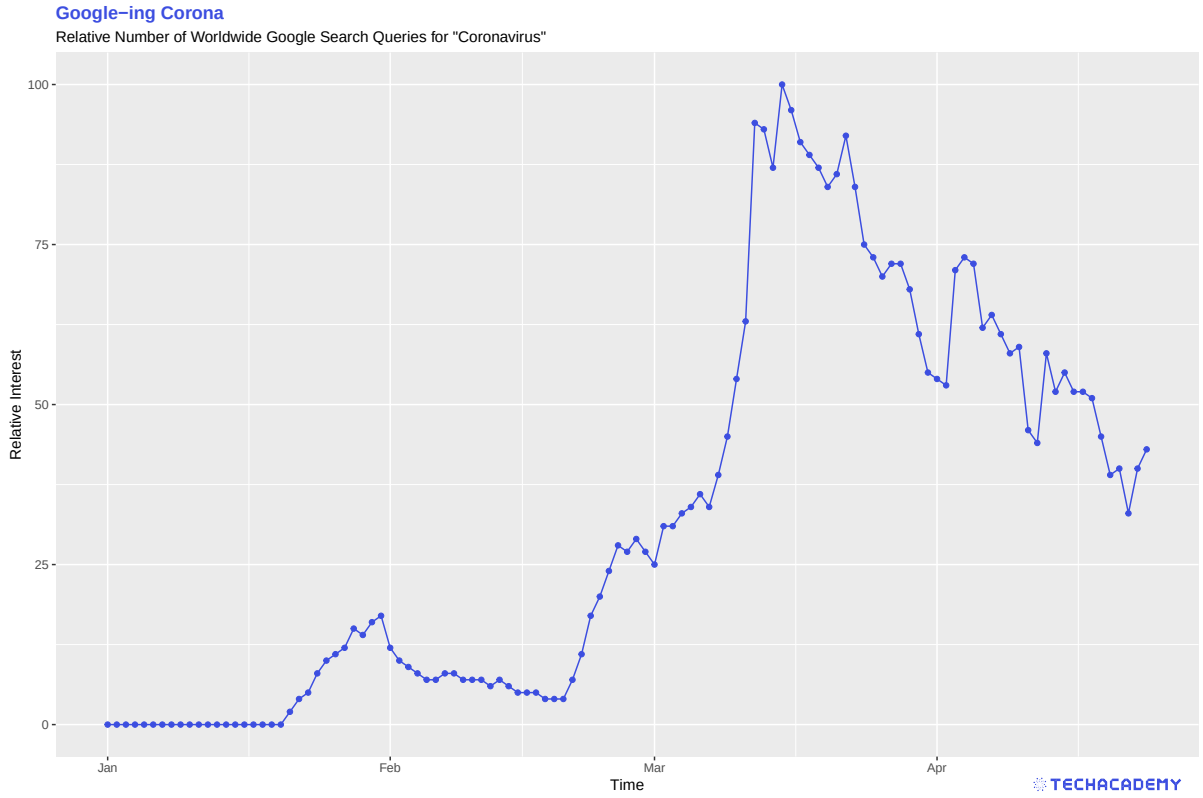
Now it is time to get a quick overview of the data set. What is the structure of the data set, which variables does it contain, what is the type of the variables? What is noticeable?

Before we can get started with the actual visualization, we have to tweak the data set a little so that the different commands will work later on. Sometimes this is easier said than done and very often the format of the data is not usable at first – you have to figure out why and how to manipulate the data make it useful for your next steps.

To create a time series plot in the next step, it is necessary for the date variable to have the right format. Do you already know how to do that from your DataCamp courses?

After having changed the data to a coherent and meaningful format, it is now time to create our first line plot. Before you actually plot the data, it is helpful to envision what it could look like. Which variable has to be on the x- and y-axis, respectively?

This is what a visualization could look like, but don't worry, your result does not have to look exactly like this as long as the structure of the plot is similar.



Looking at the line plot, you can see a clear pattern with respect to different dates. Do you know what happened on the dates the spikes in searches occurred? If you have time you can play with the visualization and refine it, add labels and change colors, but that is not too important, so you can first have a look at the following steps and come back to this when you have completed the rest.

`R` provides many different ways to import data. For starters, use the built-in `base` function `read.csv()` and define the path to the Google Trends file in the argument of the function. We have stored all data for your project in the folder `data`. Don't forget to use quotation marks `"..."` for the path – you always need to use them when you specify `string`-variables in `R`. Also, you will need to adapt to the `R`-way of specifying paths. Defining a file path can either be done with `\\` or `/`, but not via a single backslash `\` like in your explorer.
Use the `base` functions `str()` and `summary()` to print useful information on the data type and summary statistics for your `gtrends` data set.
Converting data types is relatively straightforward. Just use the useful functions in the format `as.<<insert data type here>>`. If you don't convert the `date` variable from a factor to a more useful data type, you will have problems plotting it.
For your first plot, use the simple `base` function `plot()` and define the two relevant input vectors for the time and the relative number of searches. While this function is great for very fast initial plots, you will need a more advanced graphics package later on. Do the same plot again with `ggplot2`. That gives you way more control and flexibility over plotting. Remember from your DataCamp courses that you first need to install a package once and then load it with the `library()` command at the beginning of your `R` session.

To import csv-files into your Jupyter notebook and to convert them into a pandas data frame use pandas' `.read_csv()` method. To read the data, you have to tell pandas the path to the files in the data folder from your current directory. To traverse upwards in the directory structure, you have to use two dots, so something like `../data/<here the rest of the path>` should work. For the first plot you need the Google trends data. Afterwards use pandas' `.head()`, `.describe()`, and the `.info()` methods to explore the data: Are the dates actually stored as datetime objects? If not, you can change objects to datetime with `pd.to_datetime(df.date, format='%Y-%m-%d')`, where `df` stands for the data frame. It would probably make sense to have the date-column set as index: You can do so by applying the `.set_index('date')` method to your data frame. Now the only thing left to do is to plot your first graph!

```
plt.plot(df['hits'])
```

Make the plot visually more appealing by using additional pyplot methods and by passing additional arguments to those methods. For example:

- For the `.plot()` method you can use additional keyword arguments, like `color`, `marker`, `linewidth`, `linestyle`, and `markersize`.
- Use the `.legend()` method to add a legend.
- You can also add labels to your x- and y-axis by passing a string argument to the `.xlabel()` and `.ylabel()` methods.
- You can rotate the ticks on the x-axis with `.plt.xticks(rotation=45)`.

Find a full list of methods in the official matplotlib documentation.

4.2 Visualize Stock Data

The coronavirus had an impact on many areas of our lives and our economy, the stock market being one of them. When visualizing the stock data, the impact becomes evident. Many firms suffered from the outbreak, others profited. Let's have a look!

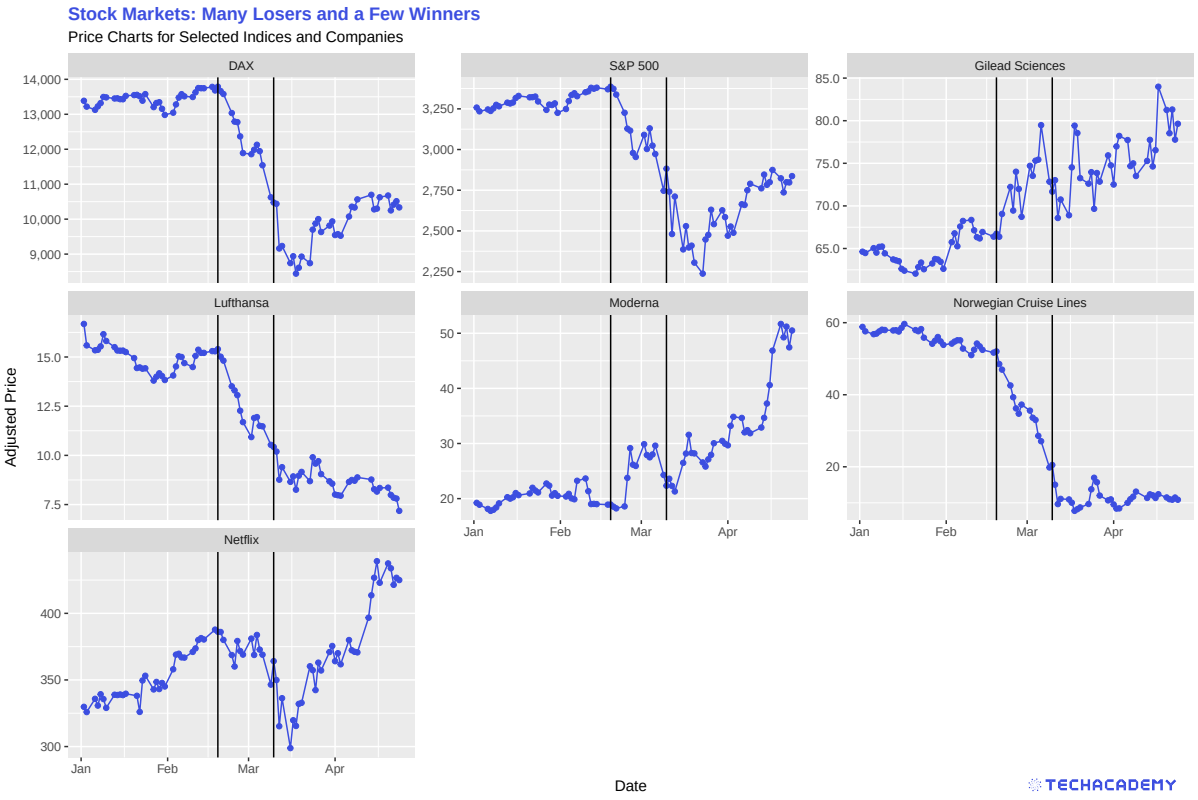
As before, you first need to import the stock market dataset and name it accordingly. Once more, it makes sense to get an overview of the data to see what you are working with. Can you

already see which variables need to be transformed?

Before visualizing the stock data, try to change the names of the stock symbols, so that people who are not familiar with them know which firm's stocks are being analyzed.

This time, we will not create a single line plot but several next to each other, which allows for a better comparison of the stock price development of different firms. The following plots are supposed to give you an orientation of what your result could look like:

You can often see vertical lines in the plots to highlight when an event happened that had a severe impact on a variable. Try to add that to the plots. Here is an idea of what it could look like:



After you visualized the data, think about what the data tells you and which key statements you can derive and communicate with your visualization. Take those ideas and put them into the title and subtitle of the graph. This practice leads the consumers of your visualizations directly to your key message. People looking at your graph for the first time will then more easily and faster spot the important takeaways from your plot.

Use the package `ggplot2` and add `facet_wrap()` after you specified the essential geom(s) to automatically create individual subplots.
Then look at how much more complex this task is in `Python` and be happy that you chose `R`.
To draw vertical lines to your plots, use another `geom` and simply add it to your `ggplot`.
You can label individual entries of a factor-type vector. That doesn't change the specific entries in the vector, but changes how the entries are displayed for example in plots. Use the function `levels()` to define the levels of the stock symbols.

Import the stock data as a data frame and explore the data, just as you've done with the Google Trends data. Notice that by using only the `.head()` method, one could think that the csv-file contains only data about one single stock – this is not the case.
It would probably be easier to work with a data frame which only contains the columns we actually need, i.e. `symbol`, `date`, and `close`. Also, it would probably be nicer to work with the data frame if the `date` column would be the index column, the values of the `symbol` column would represent the column names, and the `close` prices would be the data frame's values: Let's use pandas' `.pivot()` method to achieve both at once: `df.pivot(index=..., columns=..., values=...)`.
Check if the new index is formatted as a datetime object. If not, apply the same method as you did on the Google Trends data frame. Now take a look at the whole data frame: Do you notice the `NaN`s? Try to find out why this is the case by searching it up online. However, if we would plot this data frame in a plot, we might end up with broken lines. One workaround to avoid this, could be to use the `.interpolate()` method on the data frame, which fills the `NaN` values using a linear method by default.
To create multiple plots in one figure you can use the example code below as a template. The code starts by creating a figure object, which is a container for plot elements. The function `plt.subplot(2,2,i)` adds a plot in the i-th window of the 2x2 grid inside the figure. Then, the data frame is filtered by the Netflix stock and the `plt.plot()` method is used to create the first plot. Continue filling the next windows of the 2x2 grid with stock data from other companies.

```
fig = plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
data = stocks[stocks.symbol== symbols['NFLX']]['close']
plt.plot(data)
```

You can add a vertical line to point out a date where stock prices started to become very volatile with `plt.axvline(x=pd.Timestamp('YYYY-MM-DD'), ymin=0, ymax=1)`, whereby x is the coordinate for the x axis, ymin is the bottom and ymax the top of the line. You used pretty much the same code four times in a row, in such a case it is often good to use a for-loop of the form:

```
for i in range(4)
    # your code here
```

Can you find a way to reduce your code using a for-loop? Try to address the changing parts of your code with the index i in the for-loop.
Afterwards you can try to make your plot look nicer. Use `fig.suptitle()` to add a title. If you feel like needing to adjust spaces between the plots, you can use the `plt.subplots_adjust()` to do so. Try using `fig.autofmt_xdate()` to format the date tick labels. If you need help with these functions, try to find out what they do and how to use them in the matplotlib documentation.

4.3 Visualize Covid-19 Infection Data

4.3.1 Worldwide Development of Confirmed Cases

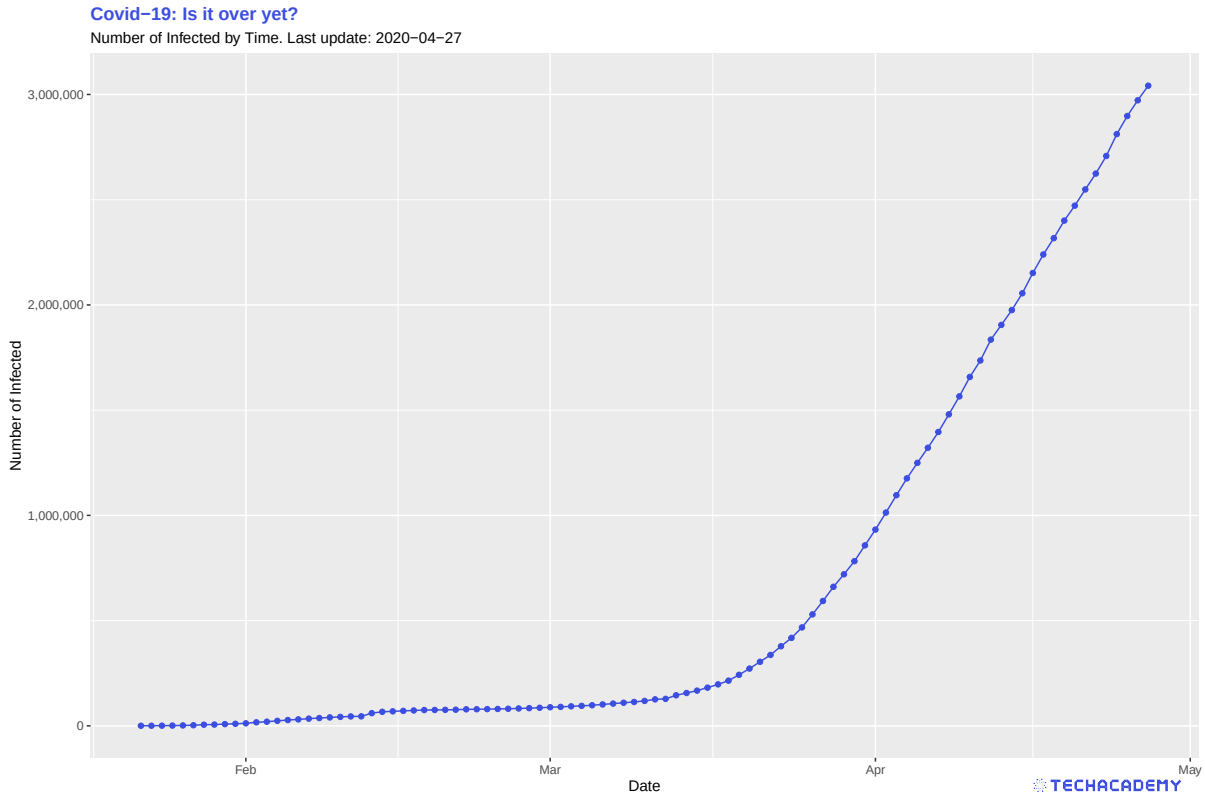
Over the last couple of months you heard new numbers about the coronavirus every day, so it is time to bring some clarity into the development of the virus in different countries over time. So now that we had a first look at certain data sets with a connection to the coronavirus, it is time to look at the coronavirus data itself.

The first data set that you are going to import contains data about the number of confirmed cases all over the world. Name it accordingly and once again, have a look at its structure.

As you can see, we have missing values in the confirmed variable. Your task will be to replace those with the value 0. We can do this because we can assume that if there is a missing value it is most likely that there has not been a confirmed case yet.

Once again it makes sense to create a simple line plot to take a look at the number of confirmed cases over time. In the first step, we will have a look at the worldwide development. For that you will have to transform the data. At this point, our data set has a separate number of confirmed cases for each country per day. In the end, our transformed data set should summarize this so that we only have one number per day that represents the aggregate number of confirmed cases worldwide.

This is what your plot could look like: You can see that the line plot has a small kink around February 10th. Restict the y-axis if you don't see it on first glance to the interval $y = [0, 100.000]$. What happened at that point in time that had such a great impact on the number of confirmed cases?

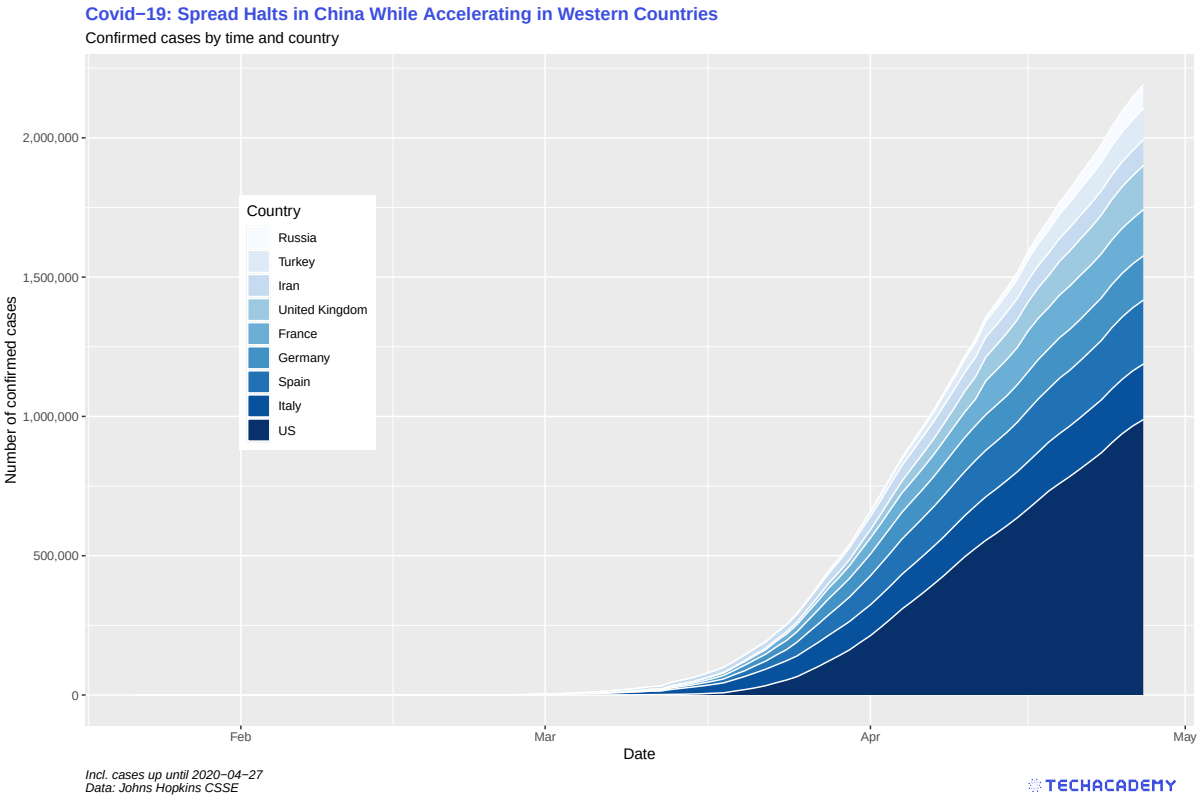


Import the new panel data set `confirmed.csv` and analyze its structure like you did in the previous exercise. Are you noticing the `NA` entries in the variable `confirmed`? Replace those missing values with `0`. There are several ways to do that, one is using `ifelse()`. After you replaced the missing values, check if you were successful by summing up all entries in the logical vector generated by `is.na()`. If this sum is zero, you have replaced all `NA` values. This works because `R` treats the logical values `TRUE` and `FALSE` as `1` and `0`.
Also, `read.csv()` by default imports dates as factors. Transform the column `time` to the appropriate `date` variable type.
You've probably already heard about the `tidyverse` – a highly influential and very useful set of packages that all follow a common philosophy. Even if it might be tough to get used to the tidy way of thinking, you will love it once you have understood it.
You can use the `dplyr` package to summarize `confirmed` by `time`. Remember the pipe-operator `%>%` from the DataCamp course Data Manipulation with dplyr in R? This operator will come in handy to build longer data transformation pipelines. Use a combination of the `dplyr` functions that group and summarize data. In case you don't remember their names and arguments, familiarize yourself with the `dplyr` documentation by typing `?dplyr` into your console or by googling the problem. In general, `?` leads you to the documentation of packages or functions in `R`.
Use `ggplot2` again to visualize your newly generated time series. Don't forget to provide meaningful labels for your title, subtitle and axes.

Import the `confirmed.csv` data as a new data frame and fill the empty spaces (`NaN`) using `.fillna(0)`. Afterwards you can use `df.isna()` to check if there are no missing values in the data frame. This will return True (corresponds to 1) and False (corresponds to 0) if there is a missing value or not. If you `sum()` over this data frame, you get the amount of missing values which should be zero now.
Since we are not interested in every single country's number of confirmed cases yet, the idea is to group all dates together by summing up the numbers of confirmed cases for each day. You can use pandas' `groupby()` function, which itself returns no result – you also have to apply an aggregation function like `groupby(by='time').max()`, `groupby(by='time').mean()` or `groupby(by='time').sum()`. For this task you can use the latter. The time column should now be set as index – make sure that its values are formatted as datetime objects. Finally, you can create the graph by passing the right column of the data frame to the `plot()` method of the matplotlib library.

4.3.2 Stacked Area Plot by Country

You have gained some experience in creating simple line plots so let's take it a step further and create a stacked area plot for the countries with the highest number of confirmed cases. The idea behind this is that you add up individual time series and color the area in between by country. This is a little tricky but with patience you will get there. The first step will be to select the countries with the highest number of confirmed cases on the most recent date of the data set. The result should look similar to this:



First, extract the names of the nine countries that are most affected on the most recent date. Use `filter()` to only keep the most recent date. You should use `arrange()` to order the data frame in descending order by `confirmed`. Then extract the top nine entries and select the column `country`. Save this vector as `top_countries` for later use.

Now prepare the data set for plotting. Filter it such that only data for `top_countries` remains. You might want to use the `%in%` operator for that task.

To draw the plot, use `ggplot2` and add the geom that generates an area plot.

The countries in the plot are not ordered in an ideal way for this visualization – countries are sorted alphabetically. Remember `levels()` that define the ordering of `factor` variables? Order those by `confirmed` cases, such that the most affected country comes last.

First you need to know which are the 9 countries with the most confirmed cases. Use the `groupby()` function with the appropriate aggregation function.
Afterwards you can sort the returned data frame with the `df.sort_values()` function and use index slicing to get only the first 9 entries of the data frame. Then store it under the variable name `top_countries`.
Now we know which countries should be visualized in the plot. For each of these countries we need the amount of confirmed cases over time. To get the confirmed cases for one country, we filter the whole data frame by this country and afterwards group the filtered data frame by time. Because we have to repeat the same procedure with each of the 9 countries, it is a good idea to use a for-loop and iterate over the list of countries stored in `top_countries`.

```
# store data in list
confirmed_list = []

# for-loop over the countries
for country in top_countries:
    confirmed_list.append(#your code to filter confirmed cases by country)
```

In each iteration of the for-loop, append only the values of confirmed cases to a list which you instantiate before the for-loop. At the end of the for-loop you should have a list of pandas series (data frame with just one column) , each containing only the amount of confirmed cases per country over time. To get only values of a pandas series, you can use `df[<column_name>].values`. This list contains the amount of confirmed cases, but not the dates. The dates are the same for each country, so we just need to get a pandas series of one of the filtered countries, select the `time` column and store it under the variable `dates`.
To get the stacked area chart, you can use the matplotlib function `plt.stackplot(x,y,labels)`. For the y values you can just pass the list of data frames which we created before and the x values should be the list of dates. The labels are the corresponding names of the countries.

4.3.3 All Curves in One Plot

Having had a closer look at the confirmed cases, we can go ahead and get more into detail by adding data of dead and recovered patients. But we don't want to keep the three data sets separate, so your next task is to merge the three data sets before we analyze the numbers more thoroughly. You can have a first glimpse at the data and look at the structure. You can see that all three data sets share the same structure, and identical observations share the same ID. This makes it easy because it allows you to merge them by ID.

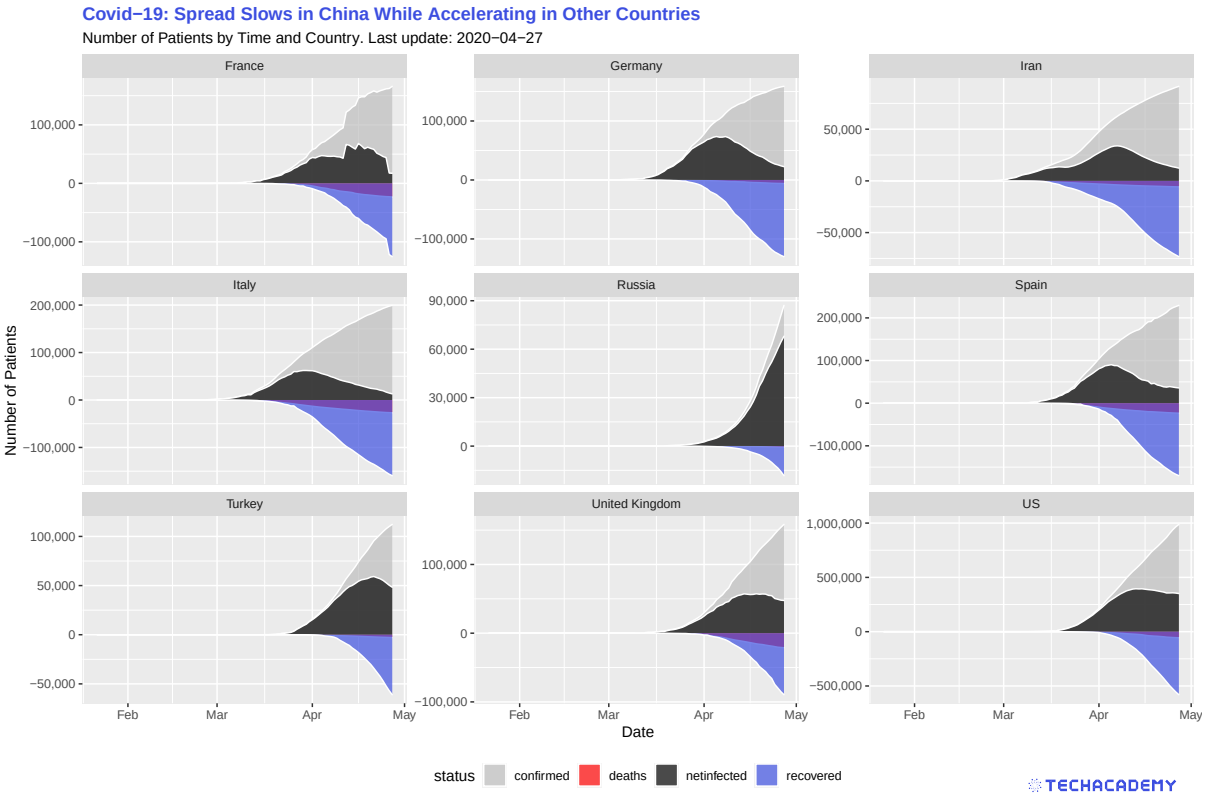
Similar to your first data set, the merged data frame has missing values. Replace them once again with the value 0.

When talking about the coronavirus many people are only mentioning the total number of confirmed cases. But it makes sense to keep track of the currently infected patients and not only the total infected. This way it is easier to see whether the virus is actually receding or still on the rise. This means we want to have a look at the number of confirmed cases that are still infected and not dead or healed already. For this you can subtract the number of deaths and recovered cases from the number of confirmed cases and create a new variable that you can call `netinfected`. Visualize the development of confirmed cases, deaths, netinfected, and recovered

in an area plot for the top 9 countries.

A quick remark on the underlying data for `recovered` and thereby also `netfinfected`: At first, Johns Hopkins University (JHU) provided the variable `recovered`. Over time, gathering reliable data on the number of healed patients grew increasingly difficult. German hospitals for example have no legal obligation to report when a patient recovered. That's why JHU stopped providing this data. As a quick work-around, we assume that two weeks after patients were infected they count as recovered if they did not die. This is a rough estimate and will not always be true, but sometimes you have to make some reasonable assumptions to work with real-world data.

If everything went well your result will look similar to this:



Import the data sets `deaths.csv` and `recovered.csv` and check out their format and structure. Notice that similarly to `confirmed.csv`, they also include the column `id`, which uniquely identifies a single date for one country across those data sets.
Now merge the three data frames by `id` into `df_merged`, which should include all three quantities `confirmed`, `deaths`, and `recovered`. You can either use the `base` function `merge()` or the more powerful `..._join()` from the `dplyr` package. `...` stands for either `left`, `right`, `inner`, or `full`. Those adjectives define which columns to keep after merging. The `dplyr` cheat sheet provides a more detailed explanation on those tools.
After you merged the data frames, again replace the `NA` values with 0. Now create the new variable `netinfected`. Use `$` to specify the new variable and simply define it as the appropriate subtraction involving the three variables `confirmed`, `deaths`, and `recovered`.
After once again filtering the `top_countries` from the full data set, we want to transform the data frame from a *wide* format into a *long* format. You have probably heard of `gather()` and `spread()` from the `tidyr` package in your courses. Those functions work well for this task, but there is an enhanced set of functions from the same package that just got released and work even more intuitively. Load `library(tidyr)` and familiarize yourself with the new functions `pivot_longer()` and `pivot_wider()`. Use one of them to accomplish the task. Use the structure below.

```
df_merged %>%
# filter here
# mutate here (optional)
  pivot_...(<<first var to pivot>>:<<last var to pivot>>,
            names_to = "...",
            values_to = "...") %>%
ggplot() +
...
```

Your first attempts might look totally weird. Play around with `ggplot2` by varying the type of plot (e.g. line or area). You can visually separate the individual columns by assigning the variable `status` to the arguments `color =` and `linetype =` within the aesthetics of `geom_line(aes(...))`. Another enhancement of your visualization is to plot `recovered` and `deaths` on the negative part of the y-axis. This avoids overplotting and enables a better communication of the data. In the previous code chunk above is a predefined space to do just that transformation. Use `mutate()` to replace the values of the two variables with its negative ones.

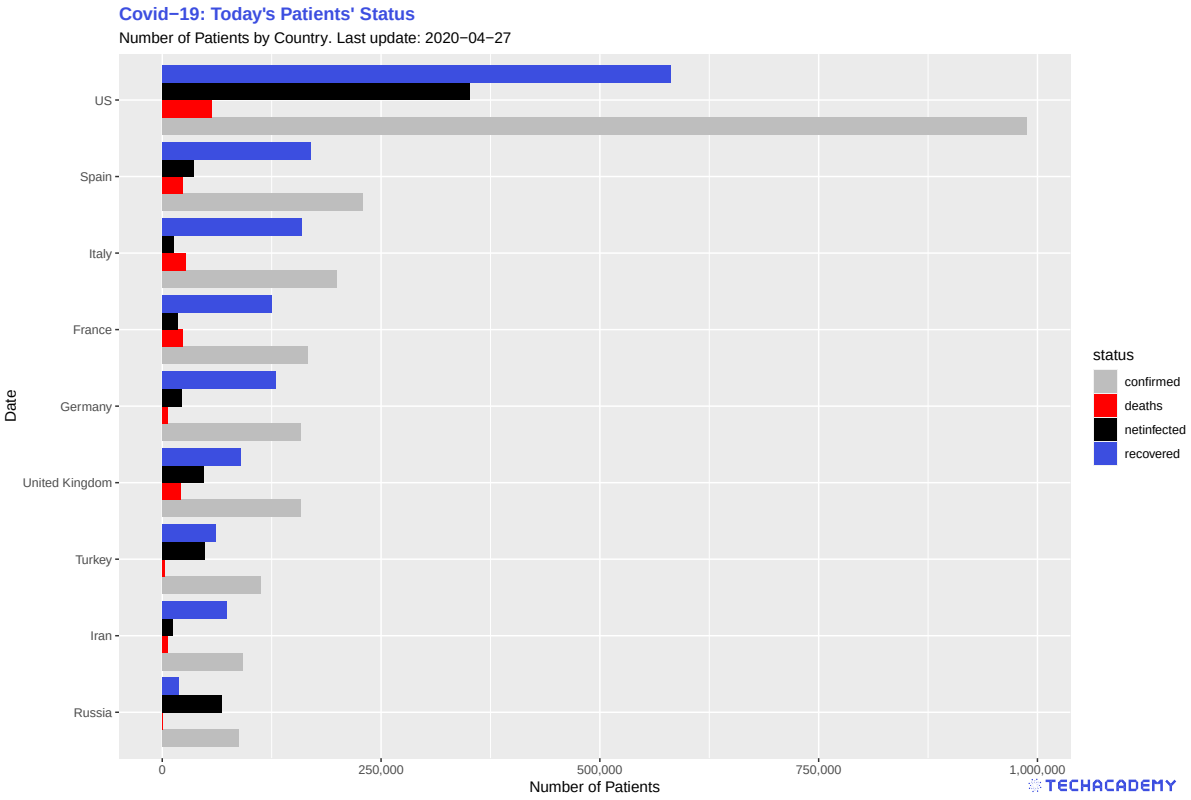
First, we need to merge the data frames `confirmed`, `deaths` and `recovered` – which means we want to form one big data frame containing the information of the three individual data frames. They share the same column `id` and can be merged over this value. You can merge two data frames with the pandas merge function: `merged_df = pd.merge(df1, df2, on='id')`. Now you still need to merge the third data frame.
After the three data frames are merged, we have to fill up `NaNs` (missing values) with the pandas `df.fillna(0)` function.
Now we want to compute the amount of `netinfected` people, which can be computed by `netinfected = confirmed - recovered - deaths`. Create a new column in the data frame and compute this value for each row.
We want to plot four time series per country. Therefore, we filter the complete data frame after the corresponding countries and use, similar to the exercise before, the `df.groupby()` function and group now by `time`. Again we have to use the right aggregation function on the `groupby()` function. But this time we need – besides the `confirmed` column – also the `deaths`, `recovered` and `netinfected` columns. You could use a simple line plot with `plt.plot()` or the function `plt.fill_between(x,y)` to get a plot which looks similar to the given example.
It is a bit tricky to get several plots into one figure, therefore we provide again an example code.

```
for i in [1,2,3,4,5,6]:
    plt.subplot(2,3,i)
    plt.fill_between(x_data, y_data)
```

Fill the `plt.fill_between()` or `plt.plot()` function in each iteration of the for-loop with data of another country and the graphs will be plotted in the corresponding windows of the plot-grid. When you run your code and you can see a grid of plots, each showing four time series of different countries, you can be proud of you. You mastered an advanced concept of the matplotlib library – which can be frustrating from time to time.
Now you can beautify the plot a bit. Add the name of the country as title to each plot (`plt.title('Taka-Tuka-Land')`). The plots for `deaths` and `recovered` contribute negatively to `netinfected` and should be plotted as negative values as shown in the example plot. Furthermore, the x-ticks should show the date, but they are not readable at the moment.
Try to rotate the ticks and plot only every 10^{th} date. To plot only every n^{th} tick is a bit challenging and we need help! We google something like `python matplotlib only nth tick`. Most of the time this will lead to the website Stackoverflow where somebody had the same problem as you and got help from other programmers. Look into the answers of this Stackoverflow post. Several users proposed possible solutions. Can you find one which works for you? And don't be afraid if you don't understand some parts of their code, try-and-error is a good way to learn and find solutions.

4.3.4 Barplot by Country

Although this is a great way of visualizing the development of the virus outbreak, some characteristics are hard to make out in this kind of visualization. Hence, it makes sense to take the same measures and create a bar plot with them. This should look like this:



Use the same data frame and transformations from wide to long as in the last plot. Additionally, filter only the most recent snapshot by using `filter(time == max(...))`. One of the great features of `ggplot2` is that you can easily switch from one type of plot to another one, just by changing the geom. For a barplot, use `geom_bar` and specify its aesthetics correctly within that geom to create four different bars for each country.
You might want to flip both axes to have your barplot better structured. There is a very intuitive command to achieve that in `ggplot2`. Google yourself to the solution.

Use the data frame from before, i.e. the data frame should contain at least the columns representing countries, numbers of deaths, time, net infected, and number of confirmed cases. If you happen to have converted any column to its negative, then convert it back to positive. Since we are not interested in the whole time series, but only in the latest data available, we can filter the data frame by the latest available date. You can use `.max()` to get the `highest` date in the appropriate column and use `df[df['time'] == latest_date]` to filter by the columns of that date.
We only want to plot nine countries with the highest number of confirmed cases: Sort by the appropriate column doing `df.sort_values(by=['...'], ascending=False)`, and then slice the data frame such that it only consists of 9 rows.
This time we are going to use pandas for plotting (which uses matplotlib's pyplot - so it is actually still creating matplotlib objects). We do so because it may be more intuitive in this case.
Our data frame should only contain the relevant columns, namely `country`, `confirmed`, `deaths`, `recovered`, and `netinfected`. You can select these columns by passing a list of column names in an indexing operator: `df[['col1', 'col2', ...]]`. Afterwards, set the `country` column as the data frame's index.
Finally, create a figure using `plt.figure()` and plot the horizontal bars:

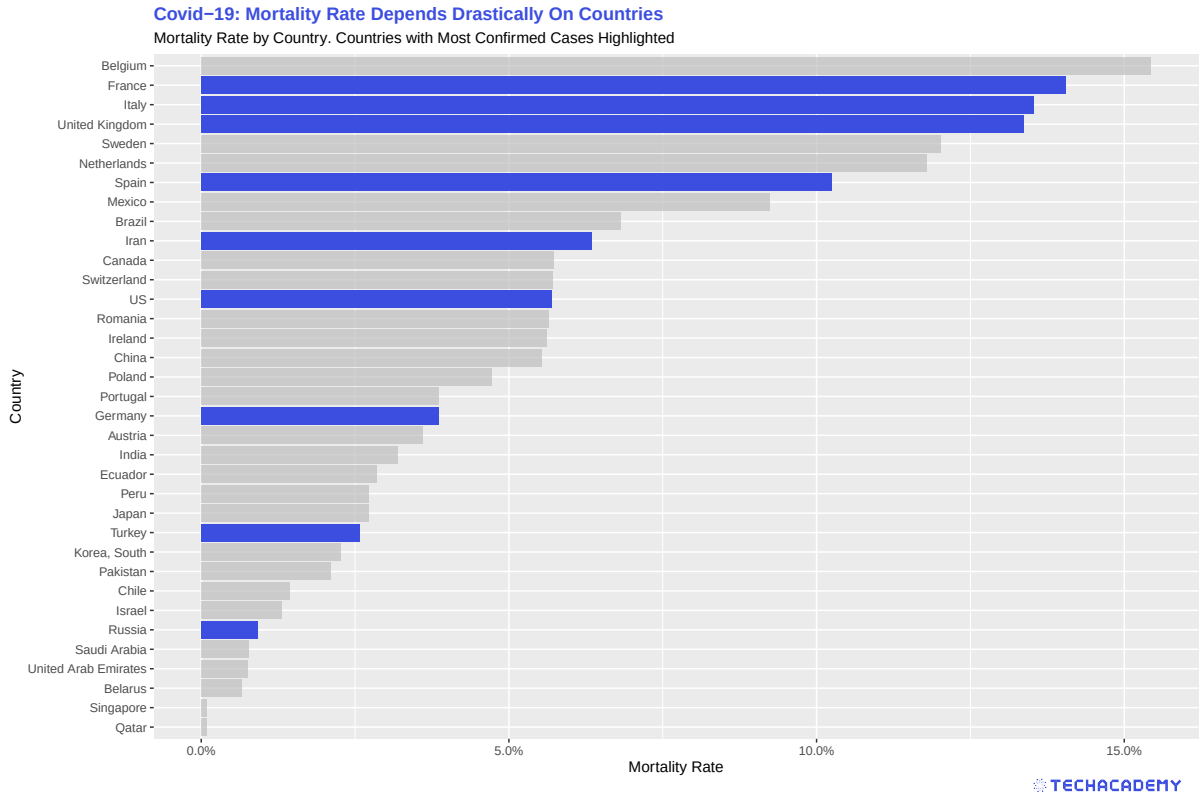
```
df.plot.barh(ax=plt.gca(), width=0.5, title='Best Plot Ever', fontsize=10)
```

Notice that, as already mentioned, we are using pandas' `df.plot.barh()` indestad of matplotlib's `plt.barh()` method (which we are going to use in the next exercise). `gca()` stands for get current axes and returns the current axes instance on the current figure. If the current axes don't exist, the appropriate axes will be created. You can try to create the plot without the `ax` keyword argument and see what happens.
Don't forget to adjust/add title, labels, etc.

4.3.5 Mortality Rate

Apart from the number of net infected, the mortality rate is another measure that is important for evaluating the risk of the coronavirus pandemic. Which variables would you choose to calculate the mortality rate? Would you pick deaths/confirmed cases, deaths/recovered or deaths/net infected? Choose a measure and give a reason for your decision. Then calculate a new variable for the mortality rate. In the next step, create a bar plot of the 35 countries with the highest mortality rate. Are you surprised by some countries that show up in this ranking? Is there a country with an extraordinarily high mortality rate that does not seem to make sense? Could you come up with a reason for this? Is there a way to filter the countries so that your barplot becomes more representative of the actual mortality rates?

In the next step, highlight the countries with the highest number of total confirmed cases. The finished bar plot that has the nine countries that have the most confirmed cases highlighted should look similar to this:



Even after filtering out outliers, you can still see that there is a big difference between the countries with regard to the mortality rate. Can you explain this difference? Is it the actual mortality rate that varies between countries or is another factor at play here? What about testing? How does it affect the mortality rate?

As a Data Scientist, it is especially crucial to think about what the data tells you. Do not always take your fancy results at face value. The newly calculated mortality rate might be such a case, where you shouldn't jump to conclusions. Why do you think it is problematic to calculate the mortality rate like we did during an ongoing pandemic? If you had all the resources that you needed, how would you collect data and calculate the mortality rate? And when would you do it?

Again, filter only the most recent snapshot by using `filter(time == max(...))`. Now use `mutate()` to calculate the mortality rate and add it to the data frame as a new column. You already know how to do a barplot from the previous hint. You can once more use `arrange()` to order the rows in descending order – this time by `mortality`.
To highlight certain columns of the plot, use the powerful package `gghighlight` and simply add its function for this task to your plot. Specify the countries you want to highlight as an argument. Maybe `top_countries` from the previous sections comes in handy here?

The tricky part here is to highlight countries with a high number of confirmed cases. The basic idea is to create two data frames: One with the top X countries by number of confirmed cases and another one with the top Y countries by death rate. Then iterate through each country in the second data frame (mortality rate), and see which countries can also be found in the first data frame (confirmed cases). For countries where this is the case, we know we have to color them in a different color.
Let's start by creating those two new data frames: You can reuse the data frame that you used just before the slicing step in the last exercise and do the following:

```
highest_confirmed = df_old
highest_mortality = df_old
```

The first data frame should contain the top 20 (or so) countries with the highest number of confirmed cases (you need to sort and slice). We need this data frame only to determine the colors of the bars. For the second one you need to add a new column displaying the mortality rate for each country: `df.assign('mortality_rate'=….)`. In place of the dots, you will need to insert your formula of choice (for example you could divide the `deaths` column by the `confirmed` column). Afterwards, filter the data frame by the 35 countries with the highest mortality rate. This is the data frame that will actually be plotted.
Now we can take care of the coloring: When plotting with `plt.barh()`, matplotlib allows us to pass a list of colors as an argument, whereby the first color in that list determines the color of the first bar, the second color determines the color of the second bar, and so on. Note that the `first` bar actually corresponds to the first row in the data frame. For example ['red', 'red', 'yellow'] would color the first and second bar red, but the third bar (i.e. third row in data frame) yellow. For us, this means that we can iterate through each country in the `highest_mortality` data frame and check if that country is also in the `highest_confirmed` data frame. If this is the case, we know we have to color that bar in a different color:

```
cols = list()  # initiate empty list

for country in highest_mortality['country']:
    # if country also in highest_confirmed, then highlight in blue
    if country in highest_confirmed['country'].values:
        cols.append('blue')
    else:
        cols.append('grey')
```

Again, understand that the first color in the `cols` list determines the color of the first bar (i.e. the first row, or country, in our data frame). Can you create the color list in one line making use of python's list comprehension?
Now, everything is ready to be plotted with `plt.barh(highest_mortality['country'], highest_mortality['mortality_rate'], color=cols)`.

4.4 Visualization with Maps

This part will be the final and most advanced part of the EDA – but also the most rewarding. If you have ever checked the news during the coronavirus outbreak you must have come across a world map with information of how the virus is spreading and how badly each country is affected. You will now build your own map where you can show which countries are affected and how high the number of confirmed cases is. Instead of having to rely on others' decisions

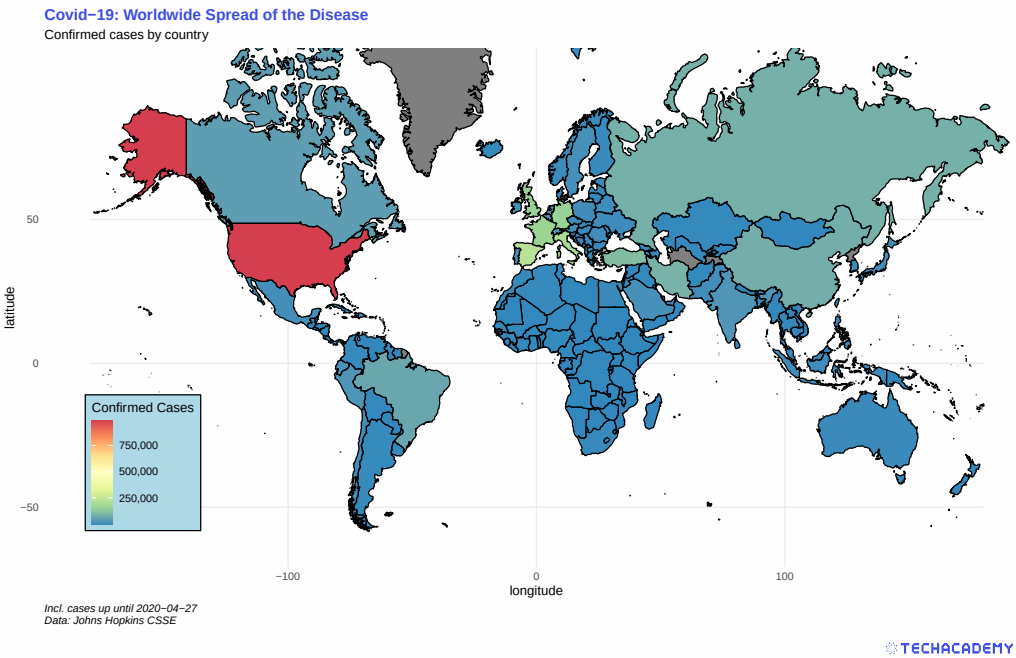
regarding the visualization of data on the virus, by creating your own map you can decide freely which type of maps visualization you prefer and which information you want to highlight.

You already have coordinates included in your data set but some data transformation is still necessary. It can get a little tricky, but as before we will give you hints along the way so that you will have an idea of what to do.

Up until this point the basic ideas of the tasks were the same for the Python and R track so the introduction to the exercises was the same, but for the maps tasks, the output will vary significantly between the tracks. That is why you will find every information necessary as well as the output you are working towards in the respective track boxes.

4.4.1 Color Countries by Confirmed Cases

For this section, we've already prepared a new data set. Load `world_map.csv` into your workspace and have a look at the included variables. It contains all data to draw all countries' borders on a map. Also, we've merged the Covid-19 data on a single day to the spatial data.
We will start with a basic map that shows all countries on a world map and colors them according to the number of confirmed cases. How convenient that we can also use our favorite graphics package `ggplot2` for that task. Supply the plot's aesthetics with the longitude and latitude as x and y-coordinates, respectively. Additionally, every country is identified by a unique `group`. We need to supply this to the aesthetics as well by `aes(..., group = group)` such that the function knows which data points belong to the same country. As geom, we will use `geom_polygon()` to draw the borders. Use the argument `fill = ...` to determine the color of a country based on `confirmed`.
A problem you will run into with this technique of map generation is that the plot will be distorted if you change its size. Countries will become wider or longer than you might be used to when looking at a typical map. This issue is connected to the broader challenge of projecting areas on a ball (the earth) from a 3D element to a simple 2D map. Check online how you can handle the proper projection in `ggplot2` and implement that in your map.



As a first step, we want to visualize the outbreak in each country with circles, where each country's circle size is is dependent on the amount of confirmed cases.
Let's start by importing the familiar `confirmed.csv` file. The goal of our map is to inform the viewer about the number of confirmed cases per country for the latest date available: Find out the latest date and filter the data frame by that date.
Filter the data frame by the columns of interest to facilitate the iteration over each row. The columns are: `country`, `latitude`, `longitude`, and `confirmed`. Now that our data frame is in the right format, it is time to check out the documentation of the ipyleaflet library. This is the library we will use to create our maps.
For our first map, we will use the module "Circle Marker". Please be sure to take a look at the example in the documentation before proceeding, especially for knowing which modules you need to import!
If you copy-paste-run the example in the docs, you can see that it plots only one single circle on the map – but we want multiple circles!
Let's start by making sure that when creating the instance `m` of the `Map` class you pass the positional argument `center=(48, -2)` (which might differ from the example in the documentation):

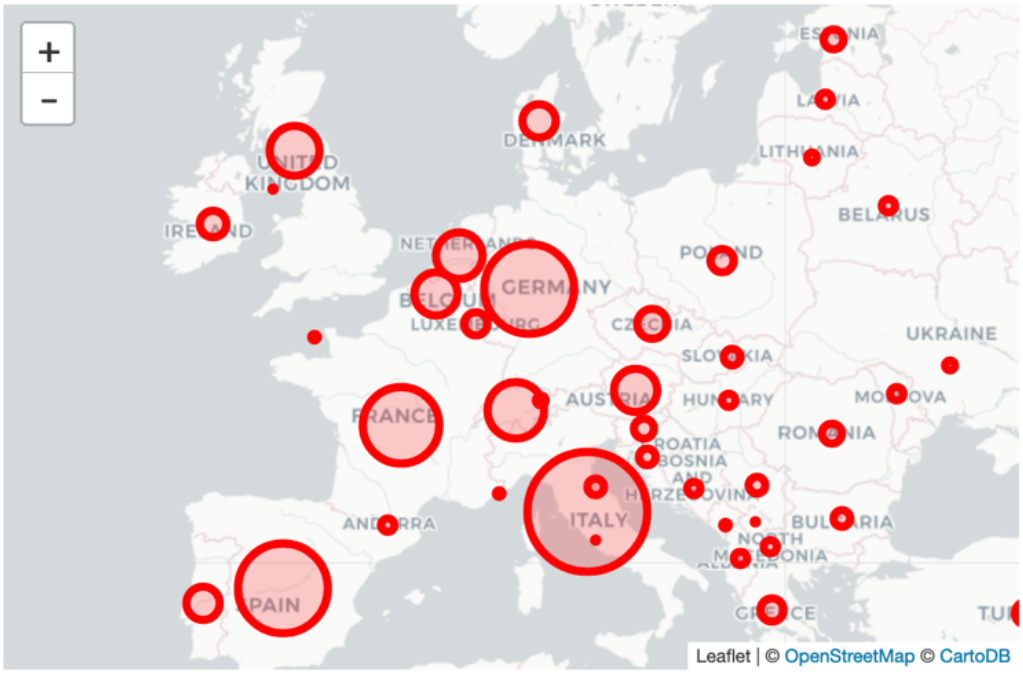
```
m = Map(center=(48, -2), zoom=9, basemap=basemaps.CartonDB.Positron)
# Feel free to change the zoom and layout of the basemap.
```

In order to plot multiple circles, you may want to iterate through every row in your data frame. Doing so, you will find it helpful to use the following technique:

```
for i, country, long, lat, nr in confirmed_df.itertuples(index=True):
    pass  # your for-loop body here
```

In each iteration you would need to create an instance of the `CircleMarker` class. Additionally, call the methods `.location()`, `.radius()`, `.color()`, and `.fill_color()` and pass on the right arguments. In the same iteration call the `.add_layer()` method with the above instance of the `CircleMarker` as its argument.
Note: Think about a way to roughly size the circles, such that the viewer has an idea about the number of cases in each country listed in our data. Obviously, if you just set the number of confirmed cases as the radius, the circles are going to look way too big. Maybe use something like the logarithm of that number?
It probably won't look great on your first try – just give it another shot!

This is what your map could look like:



4.4.2 Enhance Your Map

The next map will be the most advanced one – plot all European countries, color them according to the number of deaths and draw a label that states the number of deaths in that country. First, extract the `countrycode` of all countries within Europe. We provide you with a country lookup table that defines the region (e.g. Europe) for every individual country. Import this data set and extract the desired country codes into a vector. Use this structure:

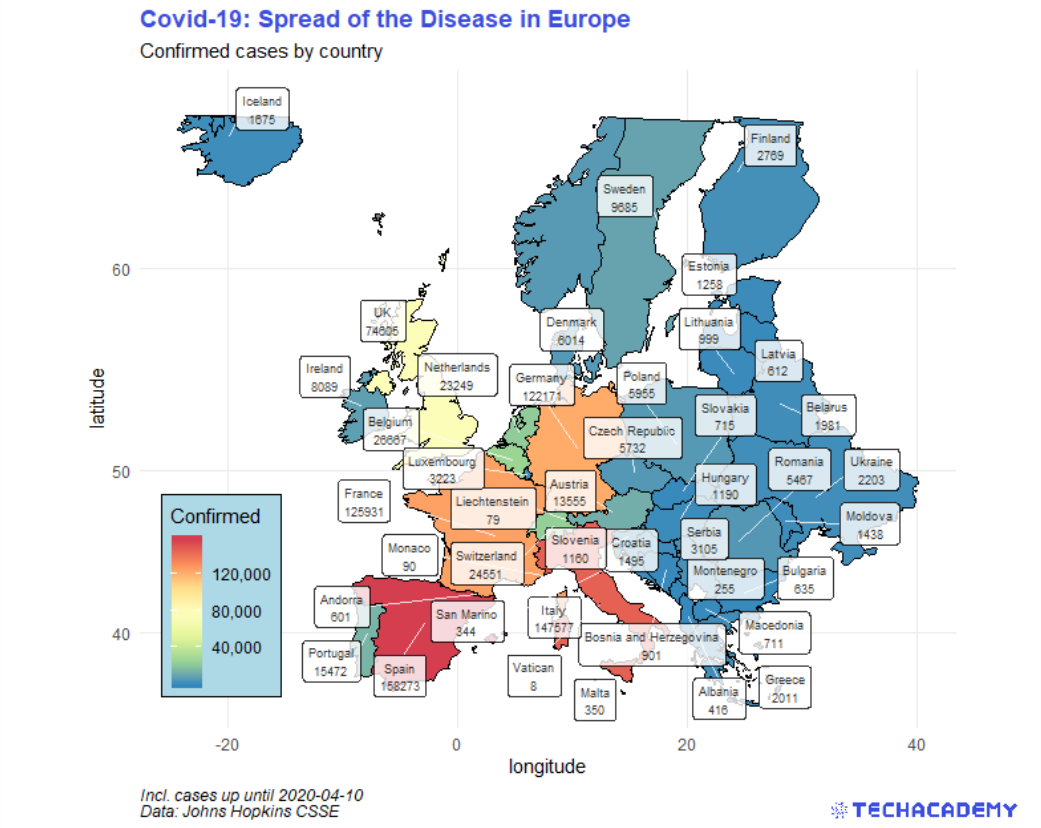
```
country_lookup <- read.csv("your/path/country_lookup.csv", stringsAsFactors = T)
country_lookup %>%
  filter(region == "<<specify region here>>") %>%
  # in case you want to exclude more countries, filter them here
  pull(alpha.3) -> countrycode_europe
```

Now you want to calculate the geographic center of a country. This will help you to place the label on the country later on. Generate that data set with the following hints:

```
world_map %>%
# group and summarize the variables of interest here
# filter your data frame to only have the relevant countries
  filter(countrycode ...) %>%
# create column with labels:
# Use "\n" within a string as a line break between
# the country name and the number of confirmed cases
# generate a new variable that contains the label in a string format
# then assign everything to a new data frame in the last line
... -> label_centroids
```

Now you can generate a map in the same way as in the first maps exercise. Filter all European countries from `world_map` using `countrycode_europe`. After adding the polygons, use the package `ggrepel` and its function `geom_label_repel()` to draw the labels. Provide the required argument(s), and define aes(…) differently than in geom_polygon(). Hint: use your newly generated data frame `label_centroids`. It is important to set `inherit.aes = FALSE`, otherwise `ggplot2` inherits the aesthetics from the geom before.
Now enhance the appearance of your plot by defining another color scale for the countries. Just having different shades of blue is not that helpful for distinguishing the countries.

This is what your map could look like:



For our last map, we are going to create a world map and color each country by its number of confirmed cases. You should be able to use the data frame for confirmed cases from before. But before continuing, you may want to have a look at the ipyleaflet documentation for the layer `GeoJson`, which we are going to use here.

In contrast to before, when we only needed latitude and longitude for a country to place our circles, we now need all the coordinates of each country's border to be able to draw it. Thus, we've prepared a `.json` file for you with all border coordinates (i.e. country polygons) for each country, which is stored as `geo.json`. You can load `.json` files into your jupyter notebook doing:

```
with open('/PATH/TO/geo.json', 'r') as f:
    world_map_dict = json.load(f)
```

This file contains a ton of data. Just notice that each list element in the dictionary key `features` contains the name of the country, as well as its coordinates (for example print `world_map_dict['features'][0]`).

In order to use the confirmed cases data frame for this map, we need to convert it into a dictionary with country names as keys and the respective number of confirmed cases as its values. We can achieve this in one line with dict comprehension and python's built in zip function: `dict(zip(df.country, df.confirmed))`.

Now that we have our `confirmed cases` and the `borders` dictionaries, we may now take care of the coloring. Again, we already prepared a json file called `colors.json`. Looking at its content in your Jupyter notebook, you will see that it has numbers stored as keys, and rgb-color-codes stored as the respective values. The numbers are actually the number of confirmed cases to which the colors "belong" to. `ipyleaflet` requires us to make two lists out of this dictionary, because we need to pass both as separate arguments when instantiating the map's `layer`. You can create both lists in two lines using python's list comprehension:

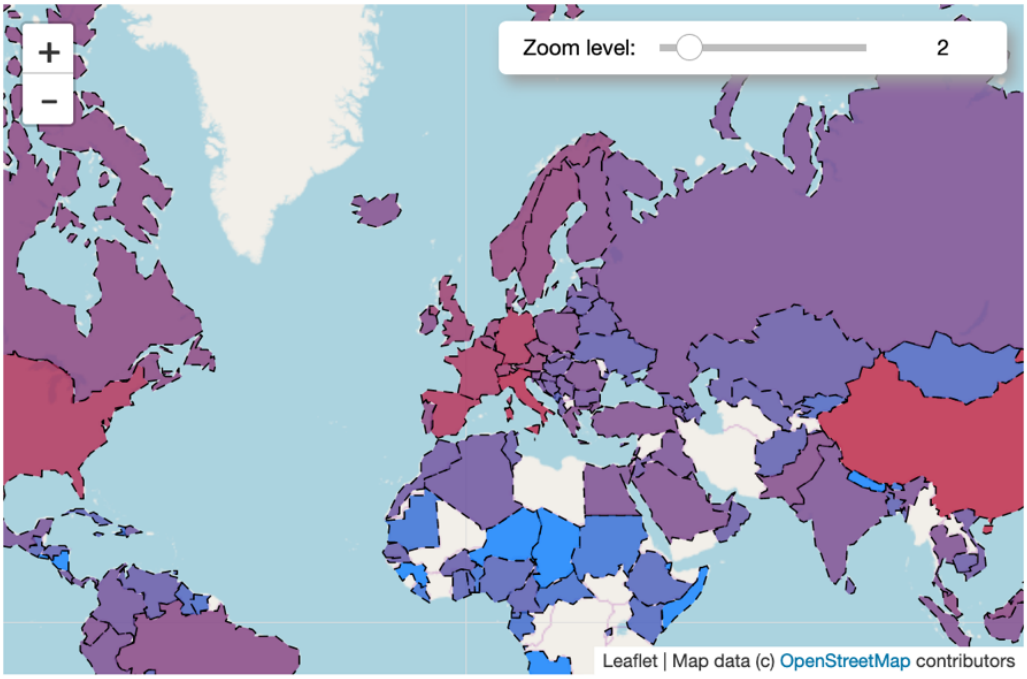
```
color_list = [v for k,v in color_codes.items()]
nr_list = [float(k) for k,v in color_codes.items()]
```

Notice how we had to use python's float method to convert the numbers to float, as they were stored as strings in our colors.json file.
That's pretty much it! We can now create the `layer`-instance of the Choropleth class:

```
layer = ipyleaflet.Choropleth(
    geo_data=world_map_dict,
    choro_data=confirmed_dict,
    colormap=LinearColormap(colors=color_list, index=nr_list,
                            vmin=0, vmax=nr_list[-1]),
    border_color='black',
    style={'fillOpacity': 0.8, 'dashArray': '5, 5'})

m = Map(center=(43, 0), zoom=4)
m.add_layer(layer)
m
```

Feel free to play around with the maps you created. Maybe try to insert some controls (see under `controls` in the ipyleaflet documentation), like the zoom-bar you can see in the picture below, or a full screen button.
This is how your map could look like:



Congratulations! Because you put in the effort and determination you managed to perform data transformations as well as visualization techniques that are highly relevant for modern Data Science. You can proudly say that you now have acquired basic skills in the field of Data Science by successfully completing the first part of the project. If you are part of the beginner track you have fulfilled the necessary requirements. However, we do recommend that you have a look at the second part of the project and try and solve it as well. This is the part where it really gets interesting because we will have a look at methods of predicting the future. Sounds exciting? Then get ready!

5 Covid-19 Prediction

Until now you have looked deeply into the given data and visualized different aspects of it. But wouldn't it be nice to predict the number of new infections with other variables contained in the data set? In this part, we'll do just that and go a step further by developing simple (or more advanced) statistical models which will be able to predict the amount of new infections on a given day.

These predictions will be based on features which contain information about this specific day. The goal of our analysis will be to identify dependencies (correlations) between these features and the number of new infections. Your model can then exploit those correlations to make predictions.

You already looked at the number of Google search queries for "corona"-related keywords in the first part. Do you think that those might have something in common with the daily new infections? Also, you've analyzed stock market data for the relevant period. Is the stock market moving according to new infections? You might think that those two variables have some predictive power. When a person is showing symptoms and fears that they might have contracted the disease, they will google it and try to get more information. Also, the more new infections are announced, the worse for the economy and hence – as a proxy for general economic expectations – the stock market. In this part, we'll check those claims and see how well models based on those and other variables perform.

The procedure to get to a statistical model ready for prediction is divided into two parts. First, you train your model on some data which is called the **train** data set. Afterwards, you test your model on data which the model has never come into contact with during training. This is called the **test** data set. This procedure gives us a good estimation of how well your model would perform in real life with completely new data. If you train your model in a way that it only remembers the data from the training data set, it would perform very well on this set. However, it would not be able to generalize the structure of the data and transfer it to new unseen data – this problem is called *overfitting*. We can detect it by evaluating the predictions of our model on a test data set. On the other hand, you could also *underfit* your model. This means it doesn't learn the relationships between the dependent and independent variable(s) well enough to perform reasonably well on the unseen test data set. Hence, your top priority must be to find the optimal balance between those two problems and fit your model well to both training and test data.

But how can you check if your model under- or overfitted or whether it is just fine? As already explained above, after the training process we evaluate our model by seeing how well it performs when being used on the test data set. We do this by letting our model use the previously unseen test observations to make predictions for the dependent variable. In our case, we would use the Google trends variable of the test data set to predict the number of new infections. We then compare our predictions with the actual number of cases that is contained in the test data set and see how close we came.

For this performance evaluation it is useful to define a metric which compares the prediction

of your model with the true values. One such measure is the *mean absolute percentage error (MAPE)* which computes the mean percentage deviation of the prediction from the true values.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{(y_t - \hat{y}_t)}{y_t}\right|$$

The larger the MAPE is, the worse your model performs. You can interpret that metric as the average percentage deviation of your model's prediction to the true values. $MAPE = 0.5$ for example means that on average, your prediction is 50% off the true values. If you look closer at other prediction tasks for continuous variables, you'll also come across some other metrics such as the *root mean squared error (RMSE)* or *mean absolute error (MAE)*. Think about why those are not well suited for this kind of prediction evaluation.

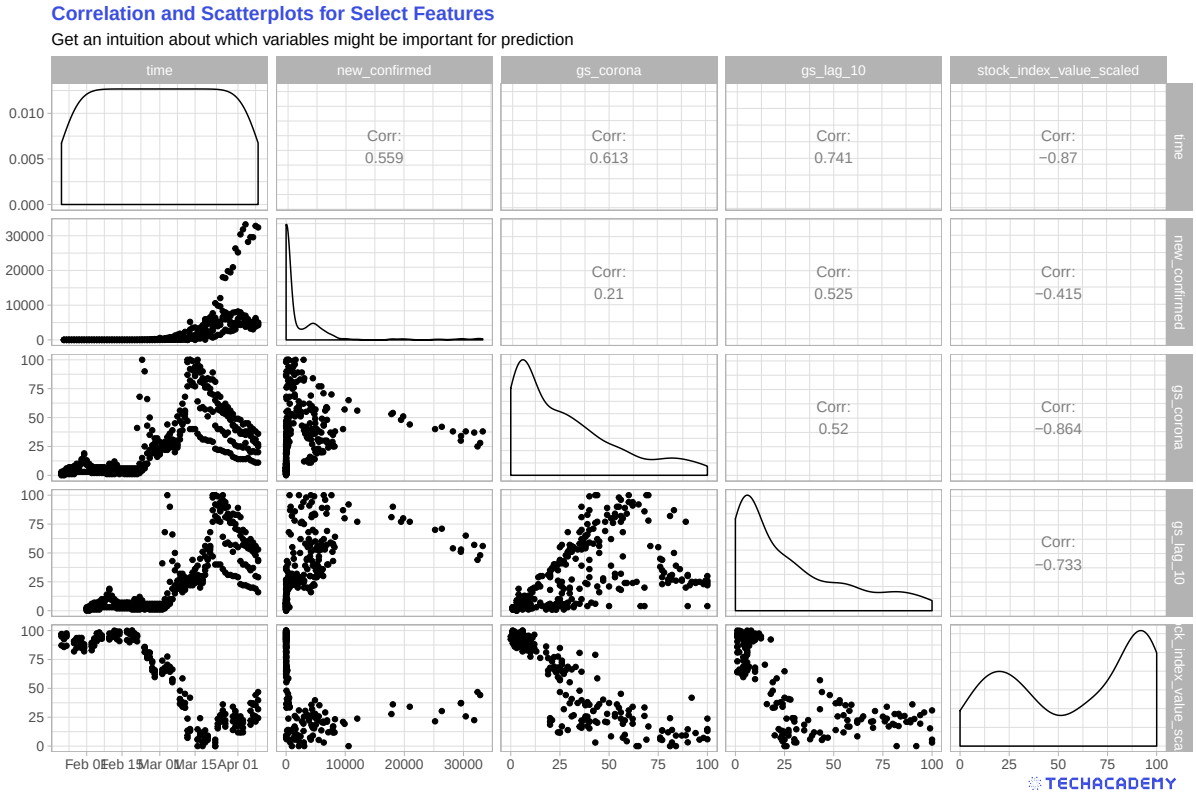
So let's get started with the prediction. Once again, we have to lay some groundwork before we can let our models do the magic. You'll start with some feature engineering and correlation analysis. After that, you can set up your first very simple univariate linear regression model (i.e. one dependent and one independent variable) for a limited set of observations – in this case German data only. When we've got that running, we'll continuously improve our model by adding or modifying features. We then take our best model and check out how well it performs on the entire data set. You'll then see how well the German model runs on four other countries.

5.1 Feature Engineering and Correlation Analysis

Before we start, you'll need to load a new data set. We've already prepared a panel data set with five countries and many important characteristics. Import `predict.csv`, on which this entire part will be based. Many variables are similar to the data set before, but there are also some important differences. The goal of our model is to predict the number of new infections per day. This value corresponds to the difference between the cumulative number of infections of the current day and the day before in each country. Assign the computed values to a new variable `new_confirmed` – this will be the column your model is trying to predict. In a later model, we also want to include `lags` for the Google Search trend data. This means that each row of our data frame should not only contain the search trend for the current day but also from the day before (lag=1), two days before (lag=2) and so on. Compute the lags from 1 to 10 for each row and assign it to new columns in your data frame. We will explain later why we need this and what the idea behind this feature is, but it is easier to compute it right away.

The next step is to analyse the dependency between the different columns or features in your train data set. Select several columns of the data frame, compute the correlations between them and visualize them in a scatterplot. It is important to include the column `new_confirmed` here because we are especially interested in the dependency between the other features and this column.

Your output could look similar to this one:



Compute the daily number of new infections for each observation in the data set. You can select previous day's values with the `dplyr` function `lag()`. Take care of accurately computing the lags over several countries in our data set by grouping your data frame appropriately.

You can either compute ten different lags by just copying and pasting your code. Or you can write a function that takes care of this process automatically. The first option is very error prone and not flexible at all. What if you want to do 15 instead of ten lags? So google how you can automatically create several different lag-variables at once.

Use the function `cor()` to analyze the correlation between your variables in a correlation matrix. Which features display the highest correlation?

Use the library `GGally` to generate a nice table that contains both scatterplots and correlations – but restrict yourself to promising variables. This plot is rather complex and takes increasingly more time to compute and becomes less readable the more variables you include. Maybe stick to the most promising 5 to 7 variables.

Table 1: Train/Test Split

DataSet	From	To
Train	01 March 2020	25 March 2020
Test	26 March 2020	09 April 2020

You have to compute the `new_confirmed` column first, which can easily be done with the pandas function `df['column_name'].diff()`, which computes the difference between the current value of the column and the value of the row before. But keep in mind that you first have to group your data by the different countries in the data frame. Quite similarly you can compute the lags using the pandas function `df['colum_name'].shift(1)`.
To compute the correlation matrix you can use a function from the seaborn library, which builds on top of matplotlib and is often a useful extension. Import seaborn using `import seaborn as sns` and use than the seaborn function `sns.heatmap(df.corr())`. This function takes as input a pandas data frame on which the pandas function `corr()` is applied – which computes the correlations between the different columns. If you compute the correlation heatmap for all columns in the data frame, it can be a bit confusing, therefore select just a subset of all columns. You can then visualize the correlations in a scatter plot using the seaborn function `sns.pairplot(df)`, which again takes a data frame as input.

5.2 Build a Simple Model Prototype

If you have a look at the correlation matrix from the last exercise, you notice a correlation between `new_confirmed` and `gs_corona` of around 0.2. We want to use this data now in our model and evaluate how good this prediction is. The idea is, that if a lot of people search on the internet for symptoms and the keyword `corona`, a lot of new infections are to be expected. Conversely, if few people search for the disease, we might expect less new infections. We start with a linear regression which is a simple yet powerful tool. The model will look like this:

$$new\_confirmed_t = \beta_0 + \beta_1 gs\_corona_t + \epsilon_t$$

$new\_confirmed$ is your dependent variable that you want to explain by the independent variable(s), in this case gs_corona only. To optimally fit a linear line to the data, the ordinary least squares method estimates an intercept β_0 and the coefficient(s) β_k for every single independent variable. In our univariate case, this is only β_1.

Before we can start with modelling, we have to define our train and test data sets. We will begin with a prediction for Germany, so filter the data set in a way that it only contains the data from Germany. Afterwards, divide your data set into a train and test data set based on the time periods shown in Table 1.

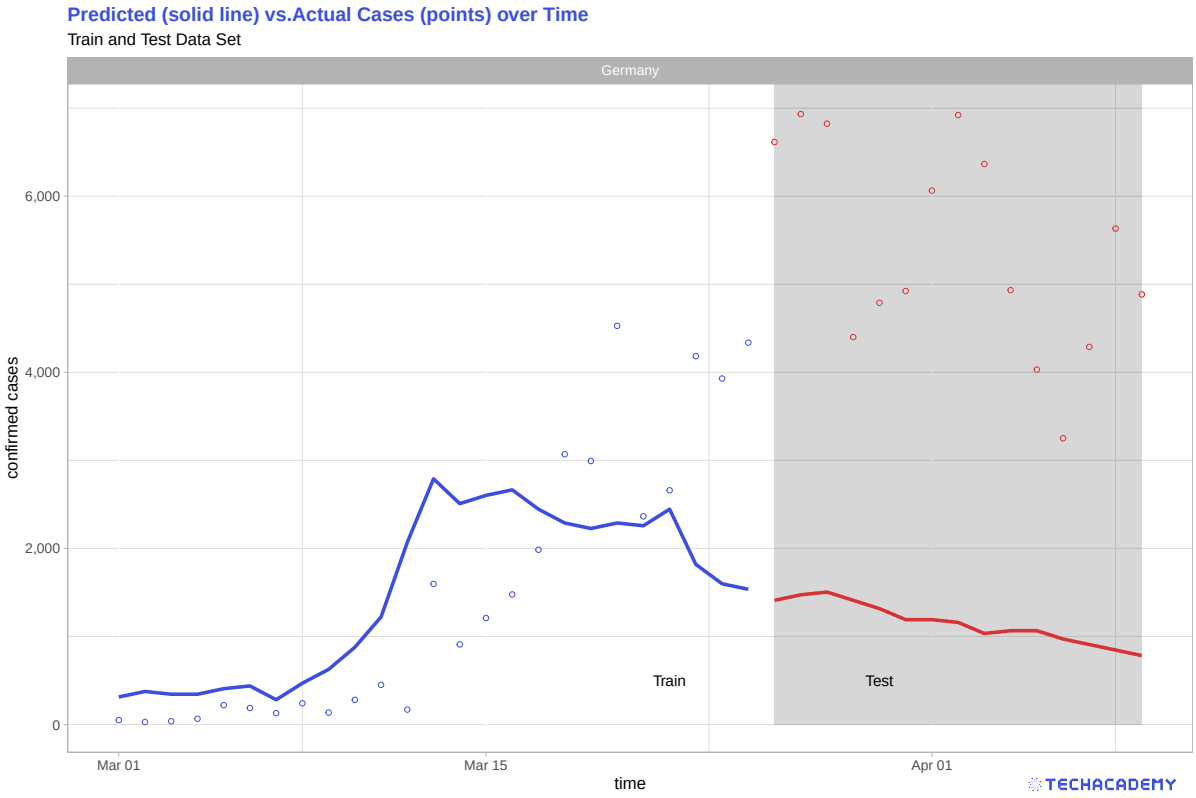
Now we are well prepared to start modelling. Define a linear regression model of the form shown above and train it on your train data set with `new_confirmed` as the value we want to predict and `gs_corona` as the only feature of the model. After our model is trained, we want to try a

prediction with this model.

Now your model has learned the relationship between the number of new infections and google searches in the training period. You can now take this model to the test time period and check how well it performs on previously unseen data.

Use your test data set to feed it into your model and predict the number of new infections for the test period. At the moment we don't know how well our model performs. Let's compute the MAPE as described above for our test data set. Compute the MAPE for your model using the predictions of the test set and the corresponding actual values of `new_confirmed`.

Furthermore, we want to visualize the true values for the train and test data set, as well as the predictions from our model data set for both time periods in one plot. This is how the plot should look like:



Your first simple linear regression model result might look something like this. Think about what the coefficient and the model fit metrics mean. How good do you think is this model based on the result?

	Dependent variable:
	new_confirmed
gs_corona	31.367***
	(8.744)
Constant	−345.827
	(572.689)
Observations	25
R^2	0.359
Adjusted R^2	0.331
Residual Std. Error	1,285.638 (df = 23)
F Statistic	12.867*** (df = 1; 23)
Note:	*p<0.1; **p<0.05; ***p<0.01

Generate two separate data sets: `train` and `test`. Split the original data by time as defined above and select only observations in Germany.
Then use the `base` function `lm()` to train the model on your `train` data set. Save and inspect that model with summary().
Use `predict()` to apply the model to your test data set. This function returns the predictions $\hat{y}_t$ as a vector.
It might make sense to name the relevant y-vectors in a structured way. You could think about such a structure:
`y_train_vec`, `yhat_train_vec`, `y_test_vec`, and `yhat_test_vec`, where `y` refers to the actual values and `yhat` denotes the predicted values.
You can easily generate regression tables in nice LaTeX, HTML or text format with the package `stargazer`. Use it to compare different models among each other in your RMarkdown document.

Filter the data correspondingly and you should have for different pandas series: X_train = values of `gs_corona` in the train period X_test = values of `gs_corona` in the test period y_train = values of `new_confirmed` in train period y_test = values of `new_confirmed` in test period
You have to import `LinearRegression()` from `sklear.linear_model` and can then train your model using model = LinearRegression() model.fit(X_train.reshape(-1,1), y_train)
You need to reshape the X values because you only use one feature in this case. If you use more then one feature you don't have to reshape your data. After training, you can use the `predict()` method on your model an pass the X_test values to the model.

5.3 Refine Your Simple Regression Model

If you look at the predictions of your model for the train and test period, you see that it doesn't work well, especially on the test period. A possibility could be that there is a lag between the time people google for symptoms to when they are diagnosed with Covid-19. You probably start googling once you show symptoms which is likely a few days before you visit a doctor. After the visit, it will additionally take a few days before your test results will arrive. Therefore we want to include a lag in our model. Luckily we've already computed the lagged values.

Train a new linear regression model, but in this case, don't use the `gs_corona` column as a feature, but the column with a lag of ten days. Your model will then have this form:

$$new\_confirmed_t = \beta_0 + \beta_1 gs\_corona_{t-10} + \epsilon_t$$

Look at the subscript for gs_corona. That's the only difference to your previous model.

Again compute the MAPE and visualize the prediction as you have done before. You can just reuse most of the code from the previous exercise. Compare the MAPE and your visualization with the model you build before. Did it improve?

Now check out the other features that are available to you in the data set. Include additional features into your model, for example, stock data or other lagged data. Build several different models and each time evaluate them and compare it with the other models. Do you find models that perform significantly better than the benchmark univariate linear regression model? Keep in mind that you can only reasonably compare a model's performance on previously unseen test data sets. Always keep visualizing the results, as this is one of the easiest methods to spot how well a model can predict the true values.

Now take your best-performing model in terms of MAPE and based on your visual inspection. Use it for the next exercise.

5.4 Extend Your Model to Several Countries

Until now you've trained your model only on data for Germany. Train your model now on several countries on the train data set and evaluate how it performs on both the training and test observations. Do this by computing the MAPE and visualizing your predictions.

Since you have different countries in your models, it might make sense to include a country-variable (dummy) in your regression.

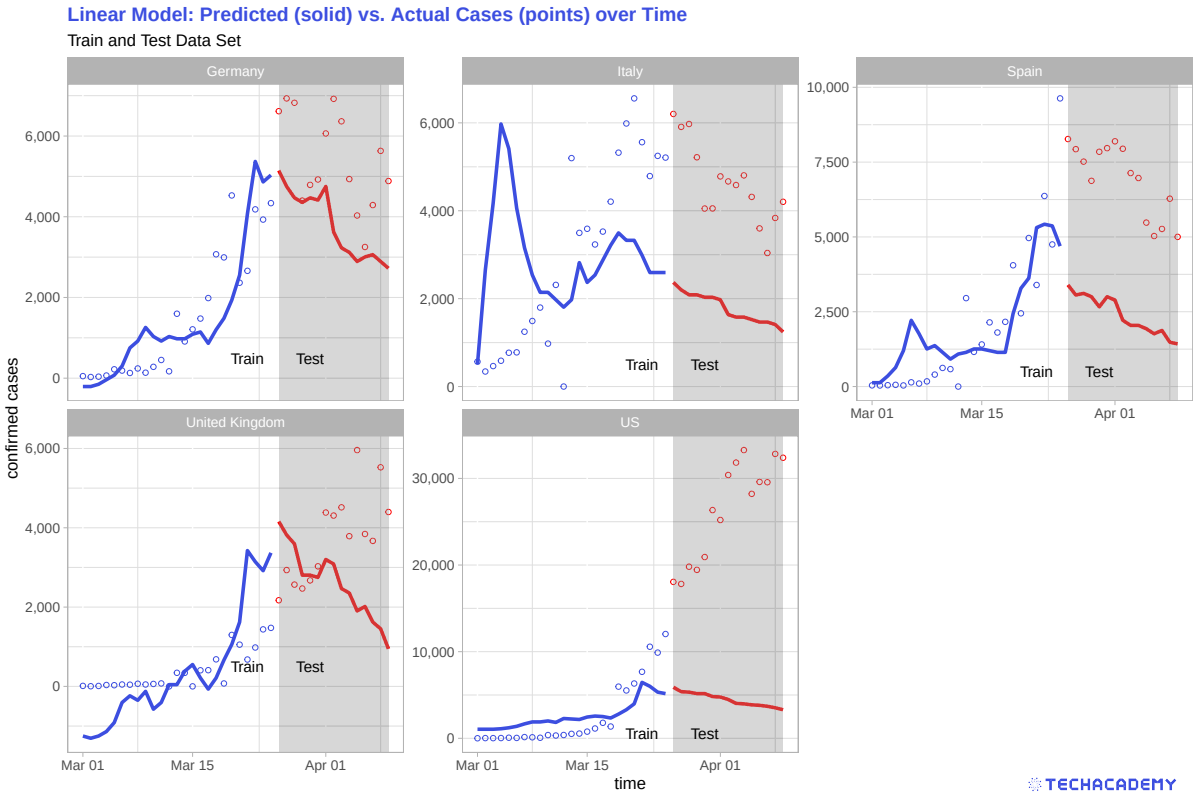
$$new\_confirmed_{it} = \beta_0 + \beta_1 gs\_corona_{it-10} + \beta_3 country_i + \epsilon_{it}$$

A side note, which you can consider in your model, but don't need to: In this model setup, β_3 tells you, by how much the number of new confirmed cases deviates from the baseline country, on average in *absolute terms*. This might not make too much sense. Take the example US and Germany. It is likely that those countries do not have a time-stable difference in the absolute number of new confirmed cases. Rather it might be the case that Germany on average has a constant percentage less daily new infections over time than the US. You can incorporate that behaviour into your model by not using the "level" of $new\_confirmed$ but instead using the natural logarithm $ln(new\_confirmed)$ as the dependent variable. The interpretation for the coefficients now changes. Google how you can interpret the coefficient for the country dummy in this case (hint: "level-level, log-level regression").

If you choose to make those transformations, don't forget to transform your predicted values back from the log values to level by exponentiating them.

Again evaluate your model's predictions visually and in terms of MAPE. How well does your

model work on other countries than Germany? Try to find reasons for country-specific differences and conclude if you would use such a model to predict the number of new daily infections.



Congratulations! You've made it to the end of your TechAcademy Data Science project. After visualizing the data in the first part, you've also set up predictive models in this section. If you stuck around until this part and managed to code the better part of the exercises, you've definitely earned your certificate! We hope you had fun learning Data Science with this data set and you enjoyed it – at least the parts where you didn't get stuck forever because of some unexplainable coding error.

6 Excercise Checklist

This checklist should help you keeping track of your exercises. Remember that you have to hand in satisfactory solutions to at least two thirds of the exercises. If you're part of the beginner track this refers to two thirds of part A (EDA) only. If you're part of the advanced track, you have to hand in at least two thirds of both individual parts A and B. Hence, you cannot hand in 100 percent of the first part and only 50 percent of the second one. You'll need more than 66% in each one for a certificate. After all, you're not really that advanced if you only did half of it, right?

Part A: Exploratory Data Analysis (Beginner and Advanced)

1. Visualize Google Searches
   - Change the format of the dates
   - Plot the time series for "coronavirus" Google searches
   - Find out what happened on the date the spike occurred
2. Visualize the stock market data
   - Change the names of the stock symbols
   - Plot the stocks in different panels and add vertical lines
3. Covid-19 Visualization
   - Create a line plot of the worldwide development of confirmed cases
     - Replace missing values with 0
     - Aggregate Data over all countries by time
     - Plot the worldwide time series of confirmed cases
     - Why is there a kink around February 10th?
   - Stacked line plot by country, top 9 countries
     - Find countries with the highest number of confirmed cases
     - Plot the stacked line plot
   - Area Plot with new variable `netinfected`
     - Merge the data sets by ID
     - Create new variable `netinfected`
     - Plot an area plot for the coronavirus variables for each country
     - Create a bar plot with the same measures for each country
   - Bar plot of the Mortality Rate
     - Create a new variable "mortality rate" with a measure of your choice
     - Select the 35 countries with the highest mortality rate
     - Create a bar plot of the mortality rate of those countries
     - Highlight the countries with the highest number of confirmed cases
     - How can you explain the difference in the mortality rate for the top 5 countries?
     - Do you see an issue with how you calculated the mortality rate? How would you measure the mortality rate if you had all the resources you needed?
   - Visualization with Maps (depends on R/Python: Check the boxes in the exercise for exact tasks)

Part B: COVID-19 Predictions (Advanced)

1. Feature Engineering and Correlation Analysis
   - Generate "New Confirmed" variable
   - Generate lag(GoogleSearch) (0-10)
   - Generate correlation matrix
   - Visualize relationships in e.g. scatterplots
2. Build a Simple Model Prototype
   - Filter Germany only
   - Generate train/test split by date
   - Set up simple univariate linear regression
   - Calculate model performance (MAPE)
   - Plot y_{it} and $\hat{y}_{it}$ vs. time in one plot and visualize train/test split
3. Refine Your Simple Regression Model
   - Again same observations as above, but use lag(GoogleSearch) as independent variable
   - Compare model performance (MAPE and visually) to the current Google Search Variable without lag
   - Try other features and check if they increase performance
4. Extend Your Model to Several Countries
   - Pick your best model and train and test it on the entire data set with several countries
   - Evaluate model performance (MAPE and visually)
   - Find reasons why country-specific differences exist and outline if your approach is viable