



## Data Science Project Guide: Spotify

TechAcademy e.V.

Summer Term 2021



# Contents

<b>1</b>	<b>Welcome!</b>	<b>5</b>
<b>2</b>	<b>What's Data Science and How Do I Do It?</b>	<b>7</b>
2.1	What's R? . . . . .	8
2.1.1	RStudio Cloud . . . . .	8
2.1.2	Curriculum . . . . .	8
2.1.3	Links . . . . .	9
2.2	What's Python? . . . . .	10
2.2.1	Anaconda and Jupyter . . . . .	10
2.2.2	Curriculum . . . . .	10
2.2.3	Links . . . . .	12
2.3	Your Data Science Project . . . . .	13
2.3.1	Coding Meetups and Requirements . . . . .	13
<b>3</b>	<b>Introduction to Your Project</b>	<b>15</b>
3.1	Purpose of the Project Guide . . . . .	15
3.2	What is this Project About? . . . . .	15
3.3	Exploratory Data Analysis – getting to know the data set . . . . .	16
3.4	Prediction – Apply Statistical Methods . . . . .	16
<b>4</b>	<b>Exploratory Data Analysis</b>	<b>17</b>
4.1	Getting Started . . . . .	19
4.1.1	Importing Data . . . . .	19
4.1.2	Convert Milliseconds to Minutes . . . . .	20
4.1.3	Song Durations . . . . .	20
4.1.4	Visualizing Song Distributions . . . . .	21
4.2	Time Series Data . . . . .	22
4.2.1	Visualizing Song Durations over Time . . . . .	22
4.2.2	Merge . . . . .	23
4.2.3	Line Plot . . . . .	23
4.2.4	Radar Plot (Advanced) . . . . .	24
4.3	Artists . . . . .	26
4.3.1	Most Productive Artists . . . . .	26
4.3.2	Charts . . . . .	26

4.4	Compare Artists Based on their Songs . . . . .	28
4.4.1	Pairplot of different artists . . . . .	28
4.4.2	Discover New Visualization Methods . . . . .	29
4.5	Analyse Personal Spotify Data . . . . .	30
4.5.1	Generate Bar Plot with Top 10 artists . . . . .	30
4.5.2	Visualize your Streaming Behavior . . . . .	31
<b>5</b>	<b>Popularity Prediction – Application of Statistical Methods</b>	<b>33</b>
5.1	Examine the Correlation Between the Variables (train) . . . . .	34
5.2	First Predictions with Simple Regression Models (train) . . . . .	34
5.3	From Training to Testing – Making Predictions . . . . .	36
5.4	Apply Advanced Machine Learning Algorithms . . . . .	36
<b>6</b>	<b>Excercise Checklist</b>	<b>39</b>
<b>7</b>	<b>What’s Next in Your Data Science Career?</b>	<b>41</b>
7.1	Data Science in General . . . . .	41
7.2	R . . . . .	42
7.3	Python . . . . .	43

# Chapter 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 four members. You can choose your 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 19, 2021**.

This Spotify case study and the associated project guide was developed and written entirely from scratch by TechAcademy's Data Science team. Karlo Lukic and Lara Zaremba developed the project in **R**, while Felix Schneider and Manuel Mair am Tinkhof developed it in **Python**.



## Chapter 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 working with the Google Colab platform that contains a Jupyter environment with the necessary libraries. You can then import all the data necessary for the project with Google Drive. We will introduce this environment 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 [Merging DataFrames with pandas \(4h\)](#) (including this course) and for advanced learners at least up to the [Exploratory Data Analysis in Python \(4h\)](#) (including this course). 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 Python \(4h\)](#)
- [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 Python \(4h\)](#): 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 19, 2021**. 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 “Price Prediction – The Application of 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.



## Chapter 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 valuable hints along the way. However, it is not a detailed step-by-step manual because we felt like you needed to develop the skills of coming up with your way of solving different tasks. This method is a great way to apply the knowledge and tools you have acquired through DataCamp.

Questions might come up, or you might not know how to solve a task right away—but don't worry—this is just part of coding. In those cases, you can also 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 you still need to complete or which assignments are relevant for your track.

### 3.2 What is this Project About?

Last semester, we dealt with the business model of Airbnb. This semester, we will have a look at Spotify data! More precisely, we are first analyzing a very detailed Spotify data set, after which you will get the chance to look at your own Spotify data. You will find all kinds of information in the data sets - valuable and useless ones.

Are you already curious to see for yourself? 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 get to know the different variables and what they look like in various visualizations. Beginners will have completed the project after this task. Still, it will be beneficial to try and work on the next part: In the second part of the project, you will develop a model that predicts a song's popularity

as accurately as possible. You will start with a linear regression model, which you can modify as you please, and then you can explore all the other possibilities of a 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. That means you will describe the data and answer questions like “Which variables are contained in the data set? ...and how are they related?”. For this, you can often use graphical tools like box plots or histograms.

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 coding requirements for the certificate. 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 understand the overall picture and the tasks we ask you to do, you will find language-specific tips and tricks in visually separated boxes. If you participate in our R-program, you’ll only need to look at the boxes with the blue border. Conversely, you only need to look at the yellow-bordered boxes if you are coding in **Python**. 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 finishing 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 significant 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 a song’s popularity based on the information you have about the songs in the data set.

For this part of the project, we recommend the advanced courses mentioned in the introductory chapter. Please note that more classes are 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 lessons 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!



## Chapter 4

# Exploratory Data Analysis

At the end of this chapter, you will look into your personal Spotify streaming history. You need to request this data from Spotify, and this can take a few days. Therefore, you should ask for your data as early as possible. Then, start with the first exercises of this chapter which will use some of the data we provide.

### Download your personal Spotify data

If you do not use Spotify personally, you can skip this step and start with chapter 4.1. In chapter 4.5, we will use the requested personal Spotify data, and you can use the data from one of your team members. If nobody on your team uses Spotify, we will provide you with some dummy data.

We will follow eight steps based on [this](#) blog post by Dan Price to download your personal Spotify data:

1. Go to <https://accounts.spotify.com/en/status> and enter your login credentials.
2. In the menu on the screen's left-hand side, click on Privacy settings.
3. Scroll down to the Download your data section. You will see the following:

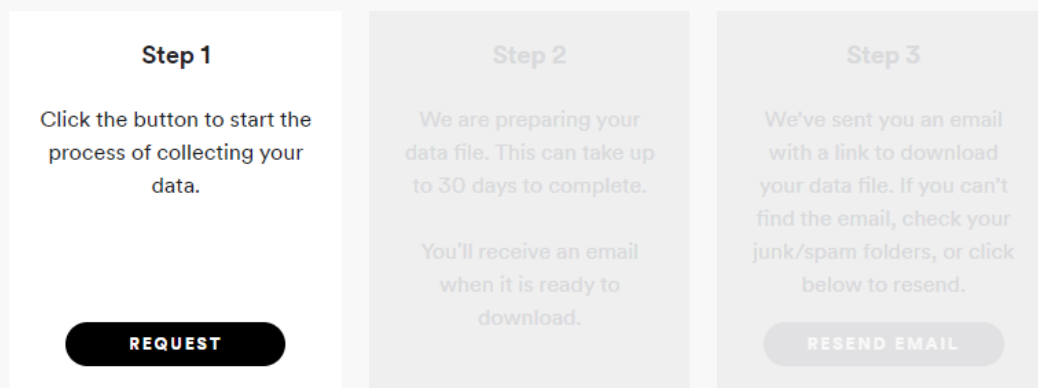
## Download your data

Most of the personal data that Spotify has about you is accessible through the Spotify app (e.g. playlists, search queries, followers, and streaming history). If you would like to get a consolidated copy of this data, you can download it by following the steps below.

The download will include a copy of your playlists, searches, streaming history for the past year, a list of items saved in your library, the number of followers you have, the number and names of the other users and artists you follow, and your payment and subscription data. For more information, see your [data rights and privacy settings](#).

As the downloadable file you will receive will contain your profile information, you should keep it secure and be careful when storing, sending, or uploading it to any other services.

If you have any questions or concerns about the personal data contained in your downloadable file, please [contact us](#).



4. Click on *Request*. You will probably have to complete a captcha and will receive a confirmation email.
5. Click “Confirm” in the email. This confirmation will take you back to Spotify, where you will receive an on-screen notification saying: “We are preparing your data file. This [process] can take up to 30 days to complete. You’ll receive an email when it is ready to download.” - Spotify
6. When the data preparation is complete, return to Privacy settings; you will now have access to Step 3.
7. Click on Download.

In the next couple of days, you will receive an email from Spotify to download your personal Spotify data as a `my_spotify_data.zip` file. Unzip the file and look for the `StreamingHistory0.json` file.

**Hint:** Check what Spotify knows about you by inspecting `Userdata.json`, `SearchQueries.json`, or `Inferences.json` files using Notepad (Windows), TextEdit (MacOS), or VIM (Linux). Surprised? :)

## 4.1 Getting Started

In this chapter, you will apply the knowledge that you gained throughout the semester to real-world data. The exercises will cover typical problems that you usually encounter when analyzing data.

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



In your workspace on [rstudio.cloud](https://rstudio.cloud), we have already uploaded an “assignment” for you (Template Spotify). When you create a new project within the workspace *Class of '21 | TechAcademy | Data Science with R*, your workspace will open up. We’ve already made some arrangements for you: The data sets 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 an HTML document when you have completed the documentation. Open the file “Markdown\_Spotify.Rmd” and see for yourself!



We recommend using [Google Colab](https://colab.research.google.com/) for this project since it requires no particular setup, stores its notebooks to your Google Drive, and makes it easy for you to share them with your team members.

As an alternative to Google Colab, you might want to install Jupyter Notebook locally using the Anaconda distribution. Either way, when importing Spotify data, you can use the links to the separate data files provided in the “Data-Links” document, which you will find in the TechAcademy drive.

We will give you a more detailed step-by-step demo during the first coding meetup.

### 4.1.1 Importing Data

First things first, you will need to load the following two CSV files into your coding environment:

- [spotify\\_songs.csv](#)
- [song\\_features.csv](#)

How is the data structured, and which variables does it contain? To get a peek at the data you’ll be working with, output the first/last few rows. Also, compute some basic descriptive statistics (count, mean, and standard deviation) and determine the column’s data types.



First, the data set has to be loaded from your folder structure into the workspace. Import both the `song_features.csv` and the `spotify_songs.csv` data sets into your workspace (e.g., using `{base}` R's `read.csv()`, `{readr}`'s `read_csv()` or `{data.table}`'s `fread()` function) and name the object(s) accordingly.

Though you can name the object however you wish, we recommend using `original_file_name` + e.g., `_df` (for data.frame) / `_tbl` (for tibble) / `_DT` (for data.table) naming convention, so that you can easily distinguish your R objects (data frames, plots, vectors, etc.) later on.

Now, get an overview of the data set. Are there any variables you could drop while reading the data set? You can use the following functions to print the data frame, for example `head()` or `glimpse()`.

Lastly, get a quick summary of the data using `{base}` R's `summary()` or `{psych}`'s `describe()` function. To find more ways to generate quick data summaries in R, check [this](#) blog post from Adam Medcalf.



Start by importing the pandas' library: `import pandas as pd`. You can use the same panda's method you used to import CSV data from your local machine to import CSV data from a web server: Just replace the file path with the URL of the CSV file.

The URLs are:

- `spotify_songs.csv`: <https://tinyurl.com/TechAcademy-songs>
- `song_features.csv`: <https://tinyurl.com/TechAcademy-song-features>

Finally, get familiar with the data by outputting some of the data frame (e.g. `df.head()`), as well as some basic information such as column data types and some descriptive statistics (`df.info()` `df.describe()`).

### 4.1.2 Convert Milliseconds to Minutes

We will start with the data set `spotify_songs.csv`.

You might have noticed a column with song durations given in milliseconds from the task before. Since (presumably) no one measures song durations in milliseconds, convert them into minutes.



You could use `{dplyr}`'s `mutate` function to compute and add a new column `duration_min` using the existing column `duration_ms`. If you haven't heard of the `{dplyr}` package yet, take the respective [DataCamp course](#) and the DataCamp's course on [Exploratory Data Analysis](#). Use the respective function and add the new variable to your existing data set.

You can use [this](#) online calculator as a point of reference for the needed `mutate()` calculation.



Compute the length of songs in minutes based on the column "duration\_ms" and assign it to a new column with

```
df["duration_minutes"] = ...
```

### 4.1.3 Song Durations

Next, we want to focus on the concept of distribution: In this task, you should focus on computing the mean and standard deviation of the duration in minutes and finding the song with the longest duration. Ideally, the output should be well formatted to something like "The

song with the longest duration is [song] by [artist].”



Simply use {base} R’s `mean()` function to compute a mean of any column in R. Similarly, R comes with `sd()` function to calculate the standard deviation of a column. You can combine these two functions, e.g., in a {dplyr}’s `summarise()` verb.

Next, find the song with the longest duration (in minutes) using the newly created `duration_min` column. The `filter()` verb from {dplyr} in combination with `max()` function could be helpful in this case. Moreover, you can combine {dplyr}’s `select()` verb with the mentioned `filter()` verb to show: song’s duration, its artist(s) and its name.

For example:

duration_min	artist(s)	song name
X.X	YY	ZZ



Calculating mean and standard deviation should be reasonably easy by applying the `mean()` and `std()` function to the corresponding column.

To find the song with the longest duration, you can use pandas `nlargest` function. You can apply it to a data frame `df` with `df.nlargest(n, column)`, where `n` is the number of rows to keep and `column` defines in which column you want to find the largest values.

#### 4.1.4 Visualizing Song Distributions

Visualizing data is essential to facilitate perception and understanding of information: Create a graph to visualize the distribution of song lengths.

We focus on creating a histogram in the tips section since it’s perhaps the most common approach for plotting distributions. You can, however, choose a different chart type; just make sure that the information you want to display is clear and correct. However, as in school, always add axis labels when possible.



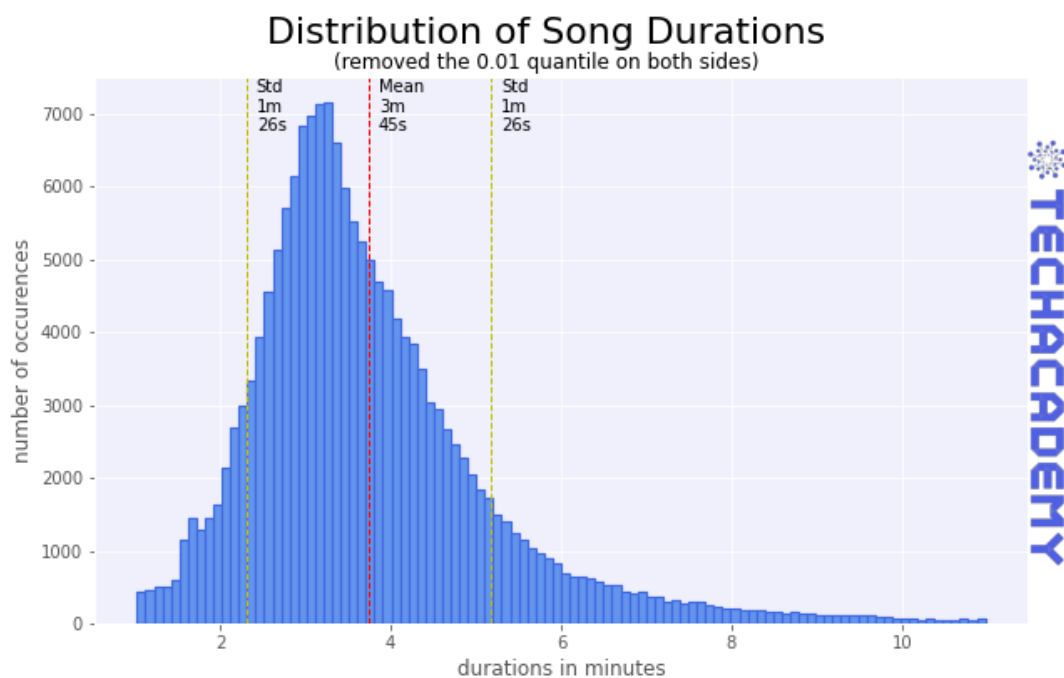
You can use {base} R’s `hist()` function to generate a simple histogram of song duration (in minutes). However, we encourage you to use the more extensive {[ggplot2](#)} package. {[ggplot2](#)}’s syntax is perhaps a bit harder to learn at the start, but it gives you more plotting benefits over the long run. If you haven’t heard of the {[ggplot2](#)} package yet, take the respective [DataCamp course](#).

You could, e.g. install {[esquisse](#)} package to learn the {[ggplot2](#)}’s syntax easier. {[esquisse](#)} will provide you a graphical user interface (GUI) inside RStudio to generate {[ggplot2](#)} figures. You can then copy/paste the syntax that generated the {[ggplot2](#)} figure to your R Markdown file. Awesome, eh?



With matplotlib’s `pyplot.hist(...)`, you can plot a histogram in one line of code.

With some more lines of code, you could (and should) add axis labels. Try adding a title to the histogram, change the color to your preferences, etc. Maybe you could even delete outliers with methods such as `col1.between()` and `col1.quantile()` - but don’t bother with details early on and leave them to the end in case you have enough time.



## 4.2 Time Series Data

### 4.2.1 Visualizing Song Durations over Time

While at the topic of song durations, it might also be interesting to see how the average song length evolved: Choose and plot the chart you think is most appropriate for displaying this type of information.

After plotting, please comment/interpret the graph: Were there any significant increases/decreases in song length over time? If so, what could be the reason?



You first need to prepare the data before visualizing it. Generate the average duration of a song (in minutes) by each year. You already used `summarise()` and `mean()` functions before. Now, you need to generate that same statistic for each year. You can, e.g., use {dplyr}'s `group_by()` verb for this purpose.

Afterward, use {ggplot2} or {base} R's `plot()` to create a chart.

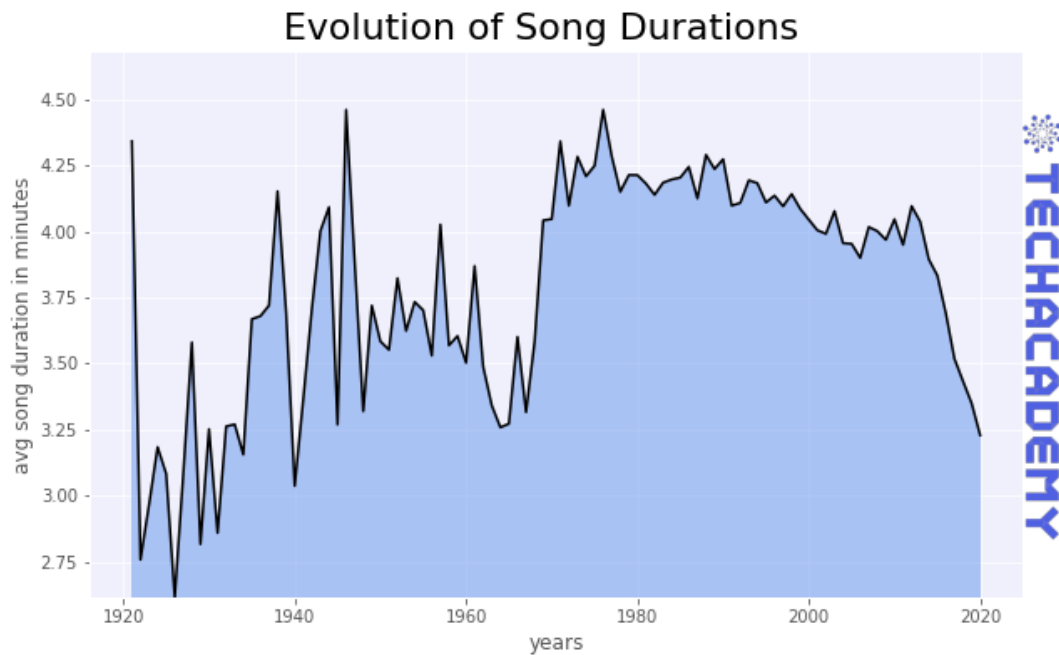
Please answer open questions in RMarkdown outside the code chunk.



There is a pandas method to group your data frame by year. In combination with the `.mean()` method, you can compute the average song length by year in one line of code.

Plot the result with matplotlib.

Please answer open questions in text cells.



### 4.2.2 Merge

As a data scientist, you'll often find yourself working with data sets from different data sources referencing the same object. For example, you might have the song names in one file and the respective song features in a separate file. It would make more sense to just merge the two data sets into one.

Indeed, this is what happened with our data. So, to make life easier for the upcoming tasks, you'll now need to merge both data sets. As you might have noticed, there is one column that both data sets share. Find out which column that is and combine the data sets on that column.



You could, e.g., use {dplyr}'s `left_join()` to combine the two data sets (`song_features.csv` and `songs.csv`) side-by-side.

Take a look at the “[R for Data Science](#)” book for an illustration of join types.



We want to combine the two data sets `spotify_songs.csv` and `song_features.csv`. The official documentation of the Pandas library provides a great guide on how to deal with this stuff: [https://pandas.pydata.org/docs/user\\_guide/merging.html](https://pandas.pydata.org/docs/user_guide/merging.html)

### 4.2.3 Line Plot

Your goal for this task is to plot how `acousticness`, `danceability`, `energy`, `valence`, `speechiness`, and `instrumentalness` changed over time.



This task involves three steps: creating a summarized data set, reshaping that data set from wide to long format if you'd like to use `{ggplot2}` and finally – plotting it.

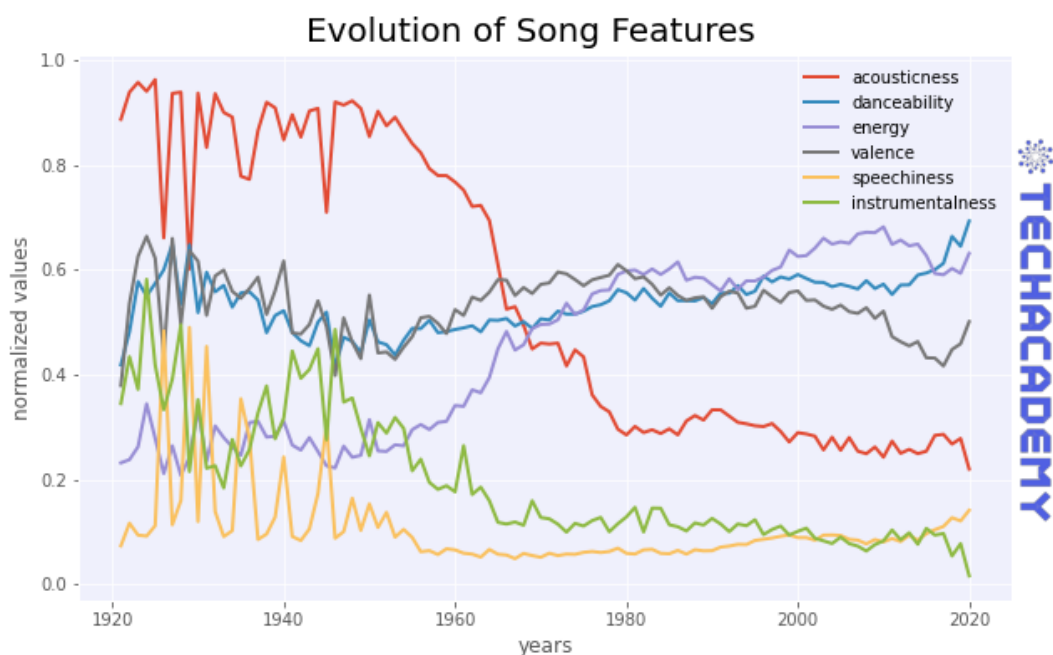
Use the familiar `group_by()` and `summarise()` `{dplyr}` verbs for the first part of the task. Unlike before, include multiple columns in your data summary. Use the familiar `mean()` function on each column to summarize the data. Save the resulting data summary as a new object.

Use the `{tidyr}`'s `pivot_longer()` for the second part of the task. `pivot_longer()` allows you to transform derived summarized data set from wide to long format by decreasing the number of columns and increasing the number of rows (a.k.a., “lengthening” the data). Such transformation is useful for plotting, e.g., in `{ggplot2}`.

Use the `{ggplot2}` to create a line chart of six variables for this task's final step. Alternatively, use the `{base}` R's `plot()`.



Once again, group the entries by year, get the mean values for the respective column for all years, and use matplotlib's `pyplot` for plotting. Don't forget to add a legend and axis labels to the plot.



#### 4.2.4 Radar Plot (Advanced)

**If you are on an advanced track, do this exercise; otherwise, you can skip it.**

Let's get a little fancy by displaying the yearly average song feature values through an interactive radar chart (aka spider plot or polar plot). This chart is very favorable in some cases when you try to display more than two dimensions. Here, each column (or dimension) gets its axis, and the dots on each axis are connected, resulting in a polygon shape.

The features you'll need to plot are: `acousticness`, `danceability`, `energy`, `valence`, `speechiness`, and `instrumentalness`.





Use the `{plotly}` package to create an interactive Radar/Spider chart of six columns over time. Similar to task 2.3 before, this task will involve summarizing, transforming, and plotting the data. You are now already experienced in summarizing the data with, e.g., `group_by()` and `summarise()` `{dplyr}` verbs. Focus on summarizing the six columns.

Afterward, transform the data from wide to long format. You could, e.g., use `{tidyr}`'s `pivot_longer()` again or `{data.table}`'s `melt()`.

Lastly, plot the figure using `{plotly}`. Use [this](#) resource as a point of reference for a static plot. Then, create an interactive animation using [this](#) resource.



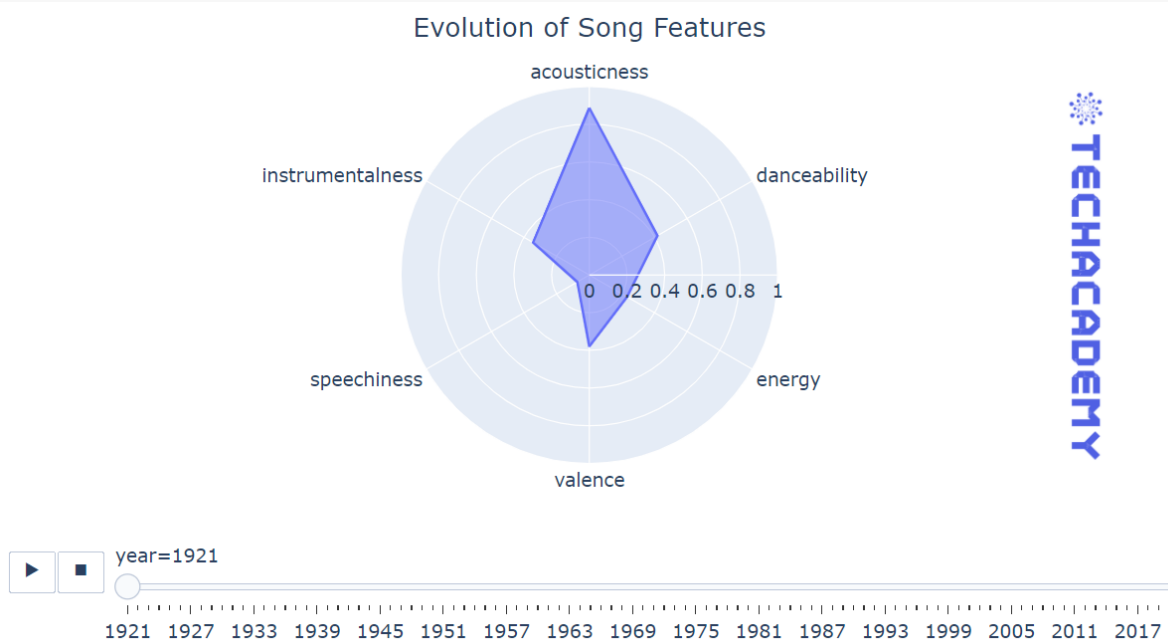
Here we work with the visualization library `plotly`, which is a more sophisticated plotting tool than `matplotlib`: `import plotly.express as px`.

Use the already computed data frame with average feature values for each year. Note that after you grouped the data frame, the `year` column automatically has been set to be the index column – reverse that (i.e., “reset” the index). Then apply the `.melt("year")` method on the data frame and have a look at the result. Before continuing, briefly try to understand what happened.

Done that, you can create the radial plot with the `px.line_polar(...)` method. The documentation (google something like “plotly express radar chart” to find the documentation) tells you which arguments you need (and can) pass to this method: - Of course, you need to pass some value for the `data_frame` argument. For our purpose, this will be the melted data frame. - You'll also need to pass values for the keyword arguments `r`, `theta`, `animation_frame`, `line_close`, and `range_r`. The values for these arguments are the strings `"value"`, `"variable"`, `"year"`, the boolean value `True`, and the tuple `(0,1)` respectively.

Apply `.show()` directly after the `line_polar(...)` method to display the plot.

As always, try updating font size, layout, etc., as you like.



## 4.3 Artists

### 4.3.1 Most Productive Artists

Our subsequent interest lies in analyzing music artists: Start by showing which artists wrote the most songs using a horizontal bar plot (or any other plot you think works best).



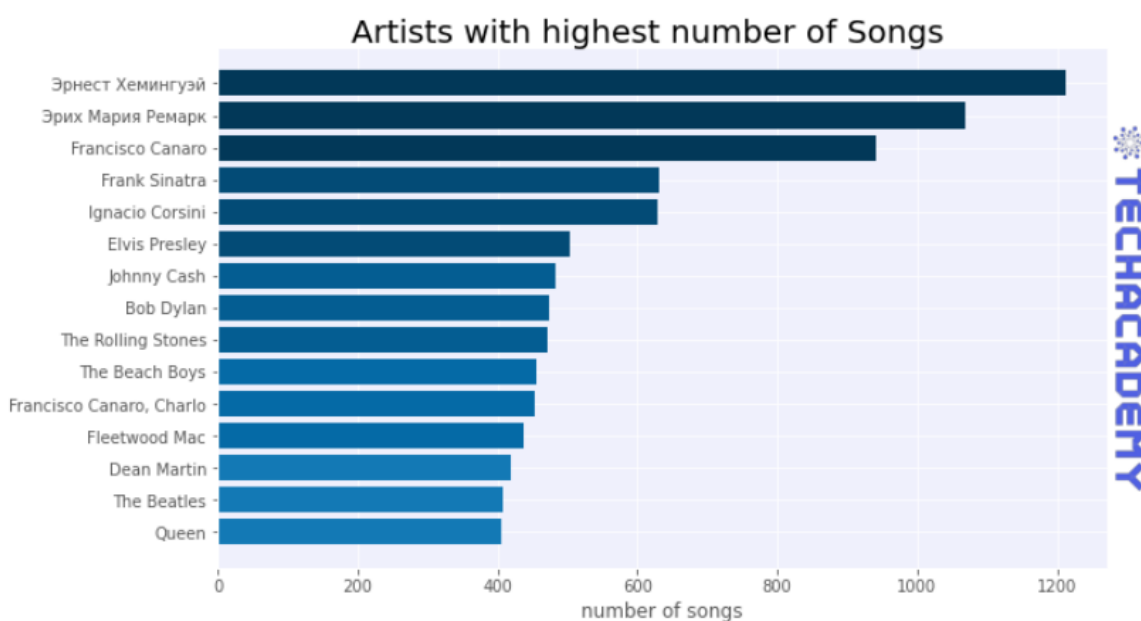
You could use the familiar `group_by()` {dplyr} verb for this task. However, it would help if you counted the songs of each artist. Does that ring a bell for another {dplyr} verb?

After grouping and counting, sort the result by its count. Then, subset the 15 artists with most songs. You can save the derived data summary to a new object.

Lastly, plot the result using {base} R or {ggplot2} by creating a bar chart.



Pandas' `.value_counts(...)` method is all you need to prepare the data. For plotting use matplotlib's `plt.barh(...)`.



### 4.3.2 Charts

Care to discover new songs?

In this section, you'll work with entirely new data (links in the respective tips section). The goal should be to create a map with a hover effect displaying each country's top song (and its composer) when hovering your mouse over that country.



First, load the data set `weekly_charts.csv` and find each country's top song. You could use the familiar `{dplyr}` verbs for this part. You could also save the resulting data set for plotting. Then, use `{highcharter}` to create a simple interactive map. Check [this documentation](#) if you'd like to do that. In the case of `{highcharter}`, you do not need any additional data as the package relies on ISO Alpha 2 (two-letter) country codes supplied in the `country` column of the `weekly_charts.csv` data set. Make use of `{highcharter}`'s `hc_tooltip()` function and its `pointFormat` argument for a mouse hover effect.

Alternatively, use `{plotly}` to create an interactive map. In this case, you have more customization options, but you need to add ISO Alpha 3 (three-letter) country codes to the `weekly_charts.csv` data set. You can load [this CSV file](#) via URL to merge ISO Alpha 3 country codes with countries in the `weekly_charts.csv` data set. Then, create a plot using `{plotly}` as described [here](#).



There exists a library for plotting maps: `import folium`. With folium, you can easily plot a map by running the following two lines:

```
m = folium.Map(location=[20,10], zoom_start=2)
m
```

Folium requires you to specify each country's country border coordinates you like to be drawn (i.e., colored, like in our example map). For this, we prepared a JSON file with all that data. Note that this is not a CSV file; hence you can't use the pandas `.read_csv()` method. You'll need to work with two new libraries for importing the data - one for requesting the data from a web server (i.e., Google Drive) and one to parse (i.e., read) the data. Here's the code:

```
import requests
import JSON

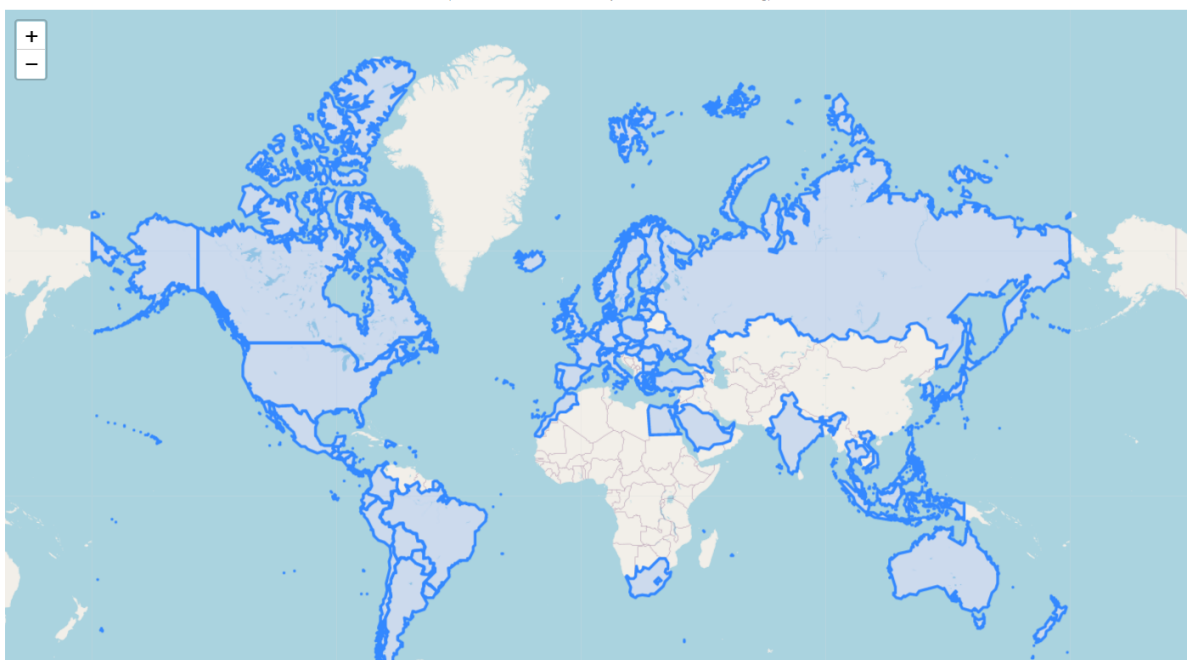
# request data from a url
response = requests.get("https://tinyurl.com/TechAcademy-GeoData")
# parse json data
geo_data = json.loads(response.content)
```

If you print out the data, you will notice each data subset belongs to a country, and that also contains the top song for the respective country: `print(geo_data["features"][0])` # prints data of the first country in JSON file

Now, "draw" the country borders by passing the `geo_data` as an argument to `folium.GeoJson(...).add_to(m)`, and assign it to the variable `geo`. Note: there is no other variable that needs to be passed as arguments here. Having done that, add the hover effect displaying each country's top song with `geo.add_child(folium.GeoJsonTooltip(...))` and specify fields and aliases through the method's parameters. There is no need to assign this to a variable. You can look up folium's `GeoJsonTooltip` class online to understand how are fields and aliases passed as arguments. Display the map by solely typing the letter `m`.

Each Country's Most Listened Song on Spotify for Week 10

(hover the cursor over a country to see it's most listened song)



## 4.4 Compare Artists Based on their Songs

### 4.4.1 Pairplot of different artists

Until now, we analyzed songs individually or compared how many songs individual artists produced. Now, we want to compare the features of songs from artists with each other. For example, how do songs from “Bob Marley” differ from “Nirvana”. Search in the data set for three different artists or bands which you like. We would recommend that these artists produce music from various genres to see more significant differences between them.



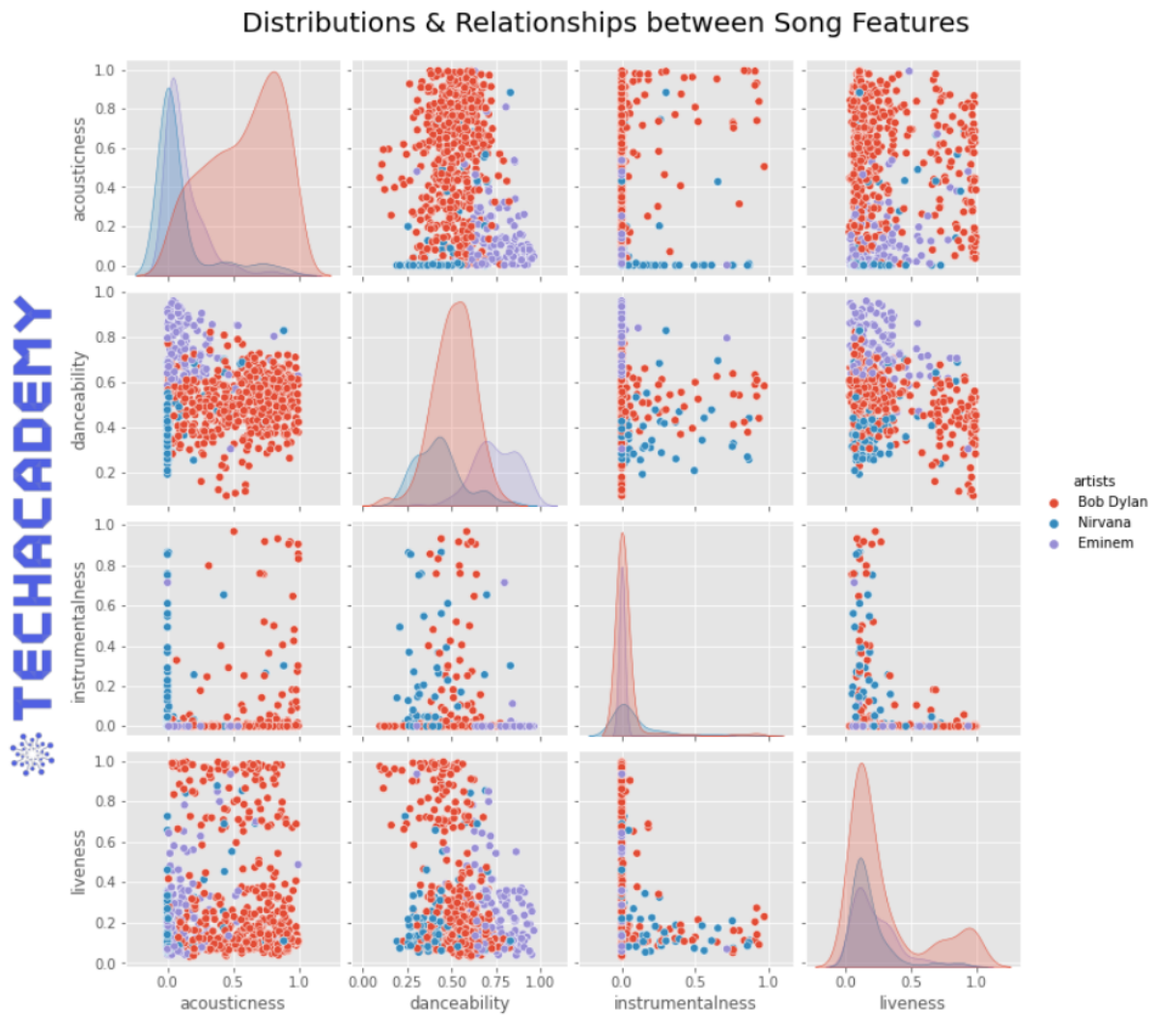
Pick the three artists of your choice (e.g., “Bob Dylan,” “Nirvana,” and “Eminem”). You could save your selection to a character vector, e.g., `my_artists_vec`.

You then need to filter/subset the data set for these three artists. That means finding the rows that contain any of those three artists’ names. While doing that, make sure to include the four columns of interest for these artists too. You could use {dplyr}’s `filter()` and `select()` verbs for this purpose.

Afterward, create a pairplot between three artists of your choice. Remember, you are comparing the four features between the three artists. You could use {GGally}’s `ggpairs()` for this purpose. Feel free to play around with the `ggpairs()` function’s arguments as described in [its documentation](#)



Pick three artists of your choice (e.g., “Bob Dylan,” “Nirvana,” and “Eminem”). Filter the data set so that it contains only songs of these three artists. Now, you can use the `seaborn` library for visualization. Import this library and use the `pairplot()` function where you pass the filtered data frame as a parameter, and as the `hue` value, you choose “artists”.



#### 4.4.2 Discover New Visualization Methods

Python and R can produce insightful visualizations, and in this exercise, we want to give you an impression of what wide range of visualization types are available. Therefore you can have a look at a gallery that shows a wide range of example plots. Choose one of these possible plots and generate such a plot with the Spotify data. Write in a comment what your visualization shows and what message it delivers. What insights about the data do you get from this plot?



Visit “[The R Graph Gallery](#)” to find inspiration for your figure. You could then use either the `{base}` R or the familiar `{ggplot2}` package to create the plot of your choice.



Visit the [“[Seaborn Gallery](https://seaborn.pydata.org/examples/index.html#)”] (<https://seaborn.pydata.org/examples/index.html#>), look for a plot that you could use to visualize an aspect of Spotify data and generate a visualization.

## 4.5 Analyse Personal Spotify Data

We dived into the provided Spotify data set, and now is the time to look into your personal streaming history. Therefore we use the data set which we requested from Spotify. If you do not use Spotify, you can also use data from a member of your team. If nobody on your team uses Spotify, contact your mentor, and we will provide you with some dummy data. Each year Spotify generates a personal wrap-up of your streaming history. Now, we want to produce something similar. First, we want to look at our favorite artists and afterward into our streaming behavior. On which days do we listen to more music or at which times of the day? This section only scratches the surface of what is possible after you receive your Spotify data. We will illustrate some ideas on how you can analyze your data. In the end, we will mention how you could expand your analysis.

### 4.5.1 Generate Bar Plot with Top 10 artists

We want to start with a Bar plot that shows your favorite artists. First, we have to load the data into our work environment. If you look at the downloaded data, you can see the `StreamingHistory0.json` JSON file. This file, however, is not a CSV file. But this is not a big problem as you will see.



First of all, you have to upload your data to access it on RStudio Cloud. Check [this](#) series of posts on how to do it. Click on the **Files** tab in the right pane of your RStudio dashboard. You will then find the **Upload** button right below the **Packages** button. After uploading your file, you will see it in your RStudio dashboard under the **Files** tab – ready to be loaded.

You can use the `fromJSON()` function available in `{jsonlite}`, `{rjson}` or `{RJSONIO}` package to load the `StreamingHistory0.json` file into your session. This works similar like, e.g., `read_csv()` function that you already used to load a CSV file.

Pick the package whose naming convention comes off most intuitively to you, load the JSON file via `fromJSON()` and save it as a familiar data frame object (e.g., `streaming_history_df`).



First of all, you have to upload your data to access it with Google Colab. On the left side, you can find a “Folder” symbol. Click on it, and you can upload your data with drag and drop. The path to your file is then “`/content/your_filename.json`”. Until now, we loaded CSV-Files with the pandas method: `pd.read_csv(file)` So, what pandas method could you use to load JSON files ;) ?

After you loaded the file into your environment, take some time to first look at the data. Print out a couple of rows and look at what columns this data set contains. It could be helpful again to compute a column `minutesPlayed` from the column `msPlayed`. Afterward, you can create a graph with the artists you listened to the most.



You could add a variable to your personal Spotify data set which counts the number of times you listened to a particular song. Take advantage, e.g., of {dplyr}'s `mutate()` verb. Name the new variable, for example `n_times_song_played`.

Afterward, you could arrange the data set by the newly created `n_times_song_played` variable. Finally, create a chart of the ten most played artists.



Because we want to group the data by the different artists and count how often they appear, the `df.groupby()` will be helpful again. Afterward, you need to sort the data and can then visualize it in a plot.

## 4.5.2 Visualize your Streaming Behavior

We computed and visualized our favorite artists, but when do we listen to music in general? This question is something we want to analyze now. There are different possibilities to explore this question, and we will start by visualizing how many songs you listened to each day with a line plot. Furthermore, you could plot if you listen to more music in the morning, during the day, or in the evening. On which weekday do you listen to more music? More during the week or on the weekend? There are many different ways to explore your data further.



Take advantage of {lubridate} package for processing dates and times in R. You can easily extract time information (e.g., year, month, day, hour or minute) from the raw `endTime` column using {lubridate}.

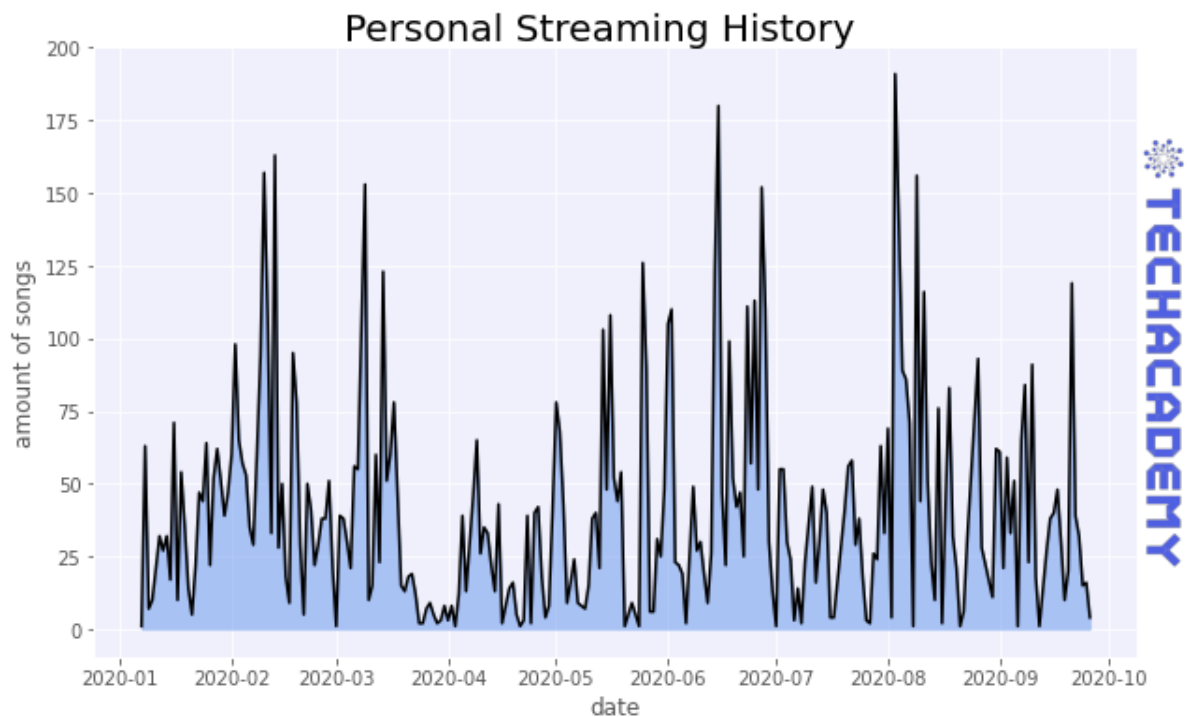
Then, add the extracted time information back to the data set, e.g., via the `mutate()` verb.

You could then add a song counter column to your data set, e.g., `n_listened_songs_per_day` using the familiar {dplyr} verbs. Make use of the extracted `day` information from the raw `endTime` column while grouping and summarizing the data.

Lastly, create a line chart of your data set using {base} R's `plot()` or {ggplot2} package.



First of all, we have to tell pandas that the column "endTime" is a date. This can be done with the `pd.to_datetime()` function. Afterward, you can group the entries by date with the following command: `df.groupby(by=df['endTime'].dt.date````). You can then, for example, count how many songs you listened to each day or sum up the amount of time you listened to and visualize this with a line plot. With df['endTime'].dt.date```` the dt you access the DateTime object, and with .date, you group the data by each date which occurs in the data. But it is also possible to group by day or hour instead of date. Try it out and visualize it in a plot.`



Congratulations – based on your work with fundamental data transformations and many visualizations, you now have a solid understanding of the Spotify songs data set as well as your personal streaming history. You have now successfully completed the first part of the project! If you are in the beginner group, you have met your minimum requirements. Nevertheless, we strongly recommend that you also have a look at the following part. There you'll be developing methods to predict the popularity of songs! Sounds exciting, doesn't it?



## Chapter 5

# Popularity Prediction – Application of Statistical Methods

In the last part, you got a feel for the data set. You now know which variables the data set contains and a few characteristic properties of it. But so far, we have only visualized the data set. This section goes a step further and uses statistical methods to predict a song's popularity as accurately as possible.

To compare your models, you will use a uniform metric to check the predictions for accuracy. In your case, this is the Root Mean Square Error (RMSE), i.e., the root of the average squared difference between the predicted ( $\hat{y}_i$ ) and the actual value ( $y_i$ ) :

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2}$$

The closer the *RMSE* is to 0, the better your model predicts prices. Therefore, in the following section, your goal is to reduce the *RMSE* of your various models as much as possible through continuous improvements.

We use the Spotify data set, which contains features about the songs as well as their popularity. When we train our model with the data, we need to estimate how well our model predicts popularity values for songs it has never seen before. Therefore we split our data set into two parts - a train and a test set.

Here is a brief description of why you need each data set:

- *train.csv* (70 %): You use this training data set to train your model. The model can learn the relationships between the variables based on the training data set containing both the variables needed to predict the popularity and the actual popularity variable.
- *test.csv* (30 %): With this test data set, you can test how well your model predicts the popularity using data it has not seen before. This prediction will help you, for example, to recognize under- or overfitting.



There are multiple ways to split/partition the data set in R. You could, e.g., use `createDataPartition()` function from `{caret}`, `sample.split()` function from `{caTools}` or — perhaps most intuitively — `split_train_test()` function from `{healthcareai}`.

Make sure, however, to include the `set.seed()` function in your R code before randomly dividing the data set into train/test parts. `set.seed()` function allows you to reproduce randomness each time you re-run your code. That means that you always get the same (random) result – a vital aspect of reproducibility within the scientific method.



The sklearn library has a function that you can use to split the data set, correspondingly from `sklearn.model_selection import train_test_split` `X_train, X_test, y_train, y_test = train_test_split(df, df.popularity, test_size=0.3)`

## 5.1 Examine the Correlation Between the Variables (train)

How are the individual variables related to each other? In other words, to what extent do the variables in the data set correlate with one another? Finding this out is vital for deciding which model specification to use later. A good place to start is to create a correlation matrix:



Use the `cor()` function from the `{base}` R to create a correlation matrix. Select all numerical variables in your data set with the help of `sapply()` or `{dplyr}`'s `select()` and create a correlation matrix.

Alternatively, use the `ggcorrplot()` from the same-name `{ggcorrplot}` package.

Check out [this article](#) by James Marquez if you're curious about other ways to create a correlation plot in R.



You can show the correlations between the features by using the `.corr()` function on a data frame. You can additionally get a colored version by using `df.corr().style.background_gradient()`

Which of your examined variables correlates most with popularity and which seems to be more independent from popularity? You now have a first impression of which variables could be vital for your model. So let's get to your first song popularity prediction model!

## 5.2 First Predictions with Simple Regression Models (train)

Now you can make use of your statistical knowledge. You will need a method with which you can predict the popularity of a Spotify song.

Have you ever heard of linear regression? Now you can use your statistics skills. First, set up a model with popularity as the dependent variable. In the previous exercise, you examined different variables. Now choose the variable with the highest correlation to popularity and use that as the only independent variable.

For example, your first regression model could look like this:

$$popularity = \beta_0 + \beta_1 \cdot danceability$$



In R, you can implement a simple linear regression with the `lm()` function. You then get the model summary with the `summary()` function.

Alternatively, check `{modelsummary}` or `{stargazer}` package to pretty-print the results of your regression model(s).

Define both dependent (`y_train`) and independent (`X_train`) variables and clean the data if necessary.



For the next step the `X_train` values need to be reshaped `.values.reshape(-1,1)`. Note: If you use more than one feature, you don't have to reshape your data.

Import `LinearRegression()` from `sklearn.linear_model` and train your model using `LinearRegression().fit()`.

Does your independent variable has a statistically significant impact on popularity? Probably yes, because we selected the variable most correlated with popularity. However, if we stick to this very simplified model, we make a typical mistake: the so-called Omitted Variable Bias (OVB). To put it simply, we neglect (in statistics jargon: “do not control for”) variables that have a significant influence on the dependent variable. One could suspect that other influencing factors also play an important role in estimating a song’s popularity. If we do not include them, the estimate of the effect of, e.g., danceability, is biased and hardly helpful. In our case, this is not a big problem for the time being since we are not interested in causal effects but rather in the best possible prediction. Your statistics professor would almost certainly object to such a model. Nonetheless, with just a single explanatory variable, this model will not necessarily predict the popularity well.

One possible solution is to include the omitted variables in the model. That is quite practical as the data set already includes these variables. So let’s set up a somewhat more extensive model that includes an additional variable:

$$popularity = \beta_0 + \beta_1 \cdot danceability + \beta_2 \cdot energy$$

Now compare the results of these two models. Does the larger model explain a higher proportion of the variance in popularity? In other words, which model shows the higher value for the  $R^2$  measure?



Use the `summary()` function of your model to compare the two models. Which model shows a higher value for  $R^2$ ? Then you can compare the RMSE between your two models.



You can now compare your two models, therefore compute for each model the  $R^2$  value.

```
import sklearn.metrics as metrics  r2=metrics.r2_score(y_true, y_pred)
```

## 5.3 From Training to Testing – Making Predictions

You have now trained your first model with the training data set. But how well does the model handle data it has not seen yet? This out-of-sample testing is a critical test to evaluate the quality of your model.

Did your model only “memorize” the existing patterns in the training data set? Then the relationships from the training data set would not be transferable to the test data set. With so-called overfitting, you trained the model too closely to the training data set and therefore received poor predictions for unknown data – for example, the data in your test set.

On the other hand, underfitting is also a problem: Your model did not sufficiently learn the actual relationships between the data. Thus, it makes poor predictions on the test data set. So it is essential to find a balance between the two problems.

Now the distinction between training and test data sets becomes essential. As a reminder: we use the train data to train a model and the test data to test our model’s quality.

To test your model on previously unseen data, you can apply it to the test data set.



Use the predict function for this:

```
predicted_popularity <- predict(your_saved_model, your_test_data_frame)
```

You have now created a vector with all popularity predictions for the test data set. You can now compare this with the actual values for popularity from the test data set.

In order to use a uniform comparison metric, please use the following function to measure your prediction accuracy:

```
# Function that returns Root Mean Squared Error while ignoring NAs rmse <-  
function(actual, predicted) { sqrt(mean((predicted - actual)^2, na.rm = TRUE)) }
```

After training, import the data from the `test.csv` file, define both variables `X_test` and `y_test`, and create a vector with price predictions applying `predict(X_test)` on your model. Store your prediction in the variable `predicted_popularity`.



Finally, compare your predicted values with the test values:

```
from sklearn.metrics import mean_squared_error # Function that returns Root Mean  
Squared Error while ignoring NaNs rmse = mean_squared_error(y_test, predicted_popularity,  
squared=False)
```

Now compare both regression models. Does the larger model have better prediction accuracy, i.e., a lower *RMSE*? Now you have a benchmark for your more advanced models, which you can beat in the next part.

## 5.4 Apply Advanced Machine Learning Algorithms

Now that you have created and tested an initial prediction using a simple regression model, you can apply more advanced methods. The goal is to get the lowest possible *RMSE* when applying the model to the test data set. Look for at least one other algorithm and check whether it gives you a more accurate prediction (expressed as a lower *RMSE*). You can find inspiration for

this in the advanced DataCamp courses listed at the beginning of the project guide. There are no limits – you can refine the regression using specific methods (e.g., LASSO) or set up a random forest model or a neural network. It is usually a good idea to briefly recall the respective algorithm’s functionality and consider whether this methodology makes sense with a continuous prediction variable.

You can also get a noticeable gain in predictive power by modifying existing variables or generating new variables from the data set (“feature engineering”). For example, you could create features based on artists. You could count the number of songs each artist produced or the mean popularity of each artist. You could also write a function that computes each artist’s mean popularity in the train data set and adds these values to the train data set. Afterward, you can add the same values to the test set, but keep in mind that there can be artists in the test set who are not in the train set. Therefore you could set the values for these artists to the mean popularity of all artists.

Always compare the *RMSE* of your advanced models with each other, as well as with the benchmark regression model from before.

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 definitively earned your certificate! We hope you had fun learning Data Science with this data set and that you enjoyed it – at least the parts where you didn’t get stuck forever because of some unexplainable coding error. Don’t forget to send your project results to our project submission email address before the deadline.



## Chapter 6

# Exercise 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 assignments. 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 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 that advanced if you only did half of it, right? :)

### Part A: Exploratory Data Analysis (Beginners + Advanced)

#### Data: Kaggle Data

#### 1. Getting Started

- 1.1 Read in data, show data, and summarize data (mean, count, ...)
- 1.2 Compute column `duration_minutes` from `duration_ms`
- 1.3 Compute mean/std of `duration_minutes`. Which song has the longest duration?
- 1.4 Plot histogram of song durations

#### 2. Time Series: How did songs change over the years?

- 2.1 Streaming leads to shorter songs? Group-By: song duration over years
- 2.2 Merge songs with `song_features` on `id`
- 2.3 Compare how features like `acousticness`, `danceability`, `energy`, `valence`, `speechiness`, and `instrumentalness` changed over time
- 2.4 For Advanced: Radar plot (aka spider plot or polar plot) over time

#### 3. Artists

- 3.1 Bar plot with artists with most songs
- 3.2 Map with top songs in each country

#### 4. Pairplot: Artist Comparison

- 4.1 Look for three different artists and compare the features of the songs with a pairplot
- 4.2 Look at the proposed gallery and look for a plot that you could use to visualize an aspect of the Spotify data and generate a plot
- 4.3 What is the message of your plot? What does it show?

#### **Data: Personal Spotify Data**

##### **5. Personal Data**

- 5.1 Generate a bar plot with top 10 artists
- 5.2 Visualize when you listened to music. On which days, at which times?

#### **Part B: Popularity Prediction Using Statistical Methods ((motivated) Beginners + Advanced)**

#### **Data: Kaggle Data**

- 1. Create a correlation plot between features
- 2. Regression
- 2.1 Create a simple linear model to predict the popularity of a song
- 2.2 Improve your model using more features
- 3. Train/test your model
- 4. Develop more advanced models



## Chapter 7

# What's Next in Your Data Science Career?

At the end of your TechAcademy semester, you've successfully coded your way through a whole Data Science project. You liked what you did and would like to bring your skills to the next level? Then this section provides you with many valuable resources to deepen your knowledge in Data Science in general or Python and R in particular. The first section is helpful for every aspiring Data Scientist, while the two following boxes introduce you to some language-specific resources. If you've come across some other worthy materials that we didn't mention here, please contact us – this list is far from complete!

### 7.1 Data Science in General

#### Version Control with [Git](#)

If you're serious about data science, you will need Git. Better learn it early and start enjoying and appreciating it before it's too late and you're pressured into learning it on the fly! You should version every project you do with Git. Regardless if you're working alone or with a big group of developers. Regardless if you write ten lines of code or a complex program. With Git, you can keep track of all your changes. It's like a Dropbox/Google Drive for developers, but way better. Pro-Tip: Get free GitHub Pro as a Student with the [GitHub Student Developer Pack](#). Besides all the perks of GitHub Pro, you'll also have free access to many other great tools. See the separate tutorials on how to set up your Git workflow.

#### Advice for [Non-Traditional Data Scientists](#)

Important advice from Gordon Shotwell, a former lawyer, on what it takes to have a successful data science career coming from a non-computer science background. Extraordinarily encouraging and helpful read on what you should and shouldn't do to reach that goal.

## Learn from Great Data Scientists on [Kaggle](#)

Kaggle is a platform that hosts data science challenges. The great thing about it is that you can browse many clever solutions to tricky machine learning tasks. And of course, you can also join the competition and measure your predictions with others. There are plenty of both Python and R notebooks.

## 7.2 R



### **Install R and RStudio Locally**

RStudio.Cloud is great for getting started with R without having to worry about installing anything locally. Sooner or later, you will have to install everything on your computer. Here's a [DataCamp tutorial](#) on how to do that.

### **Version Control with Git**

RStudio has a friendly interface that lets you enjoy the perks of Git without ever having to touch the command line – sounds great, does it? Learn how to set up the Git & R workflow with [Happy Git with R](#).

### **R Graph Gallery**

Get inspiration to take your plotting to the next level. Includes code to reproduce the plots.

### **Follow the R Master Himself and the R Community**

Hadley Wickham was and continues to be highly influential in the development of R and its rise to one of the most popular data science languages. He's behind many tools that we taught you this semester, especially the tidyverse (including great packages such as ggplot2 and dplyr). Follow him [on Twitter](#) to get excellent R advice and keep up to speed with everything new to R. Following the many people behind R (not only Hadley) is a perfect way for acquiring a deeper understanding of the language and its developments.

### **Join the [Campus useR Group in Frankfurt](#)**

There's a pretty active R community in Frankfurt that meets once a month. It's open for students, professors, industry practitioners, journalists, and all people that love to use R. In those meetings, you'll hear about other's work, discuss new developments, and ask questions.

**Listen to R Podcasts** Another great way to easily keep up with new developments in the Data Science/R community. Check out [Not So Standard Deviations](#) or [the R-Podcast](#)

## 7.3 Python



### Install Python Locally

Until now, you've only programmed using JupyterHub on the TechAcademy Server. The next step would be to install Python and Jupyter locally on your computer. This [link](#) contains the necessary information on installing the software on Windows, iOS, or Linux.

### Choosing the Right Editor

Using Jupyter is especially useful for short data analyses. But sometimes, you want to write longer scripts in Python. In these cases, it is often more convenient to use a code editor instead of Jupyter. [This tutorial](#) highlights the positive aspects of such an editor and how to choose the right one for you. Pro Tip: Also, check out the other tutorials on [Real Python](#).

### Python Graph Gallery

Get inspiration to take your plotting to the next level. Includes code to reproduce the plots.

### More Advanced Python Concepts

You know the basic data structures in Python, like lists and dictionaries. What are the subsequent steps to improve your knowledge? [This website](#) gives good explanations for slightly more advanced concepts, which can be very useful from time to time.

### A Deeper Understanding

If you want to get a deeper understanding of the Python programming language and into typical algorithms used in Data Science, this [free book](#) can be a good starting point.

### Writing Beautiful Python Code

"My code doesn't look nice, but it works!" This strategy might work for you, but often you will work on code with other people. But even if you're coding for yourself, it's a good idea to follow the PEP8 style guide. It's a useful convention on how to structure and code in Python. You'll find useful resources for PEP8 [here](#) and [here](#).

### Listen to Python Podcasts

When you don't have time for books, you can listen to [Talk Python](#) or the [Python Podcast](#).