TECHACADEMY

Data Science Project Guide: Honey and Bees

TechAcademy e.V.

Summer Term 2022

Contents

1 Welcome!

If you are a beginner, the first few chapters will introduce 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. If you are already advaced, it might be still worth a read. There will be a quick introduction to the Data Science track for you to get started with the project quickly.

Let's start with the basics! In all tracks you will work on your project in small groups with your peers. Not only will this help you to get the project done faster, but we also want you to compare your results and discuss your findings – remember, coding is just half the effort, understanding what the data and the output shows is even more important! Our experience shows: The different backgrounds of the members and discussing different opinions and ideas will produce the best results. Besides, it is way more fun to work on a project together than to code alone!

Your group should consist of approximately four team members. You can choose your teammates independently; we will not interfere with your arrangements. However, we can assist you to find your team members. 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 university faculties. This not only allows you to get to know people from other faculties, but may also give you a whole new perspective on the project and tasks. When submitting your project please note: for a certificate, each person must submit the project individually. However, hand-ins may be similar within your group. You can get more information at our first Coding Meetup on **May 18, 2022.**

Every semester, the Product Development Team at TechAcademy thinks hard about a new project that is suited for you to learn data science. This semester we decided to analyse a simple yet important part of our lives: honey and bees. While honey might not be the reason why we still exist on this planet, bees very well are, hence why honey production may also be. However, with climate change and increasing globalization, bee colonies are threatened daily. Though none of us are experts on bees and climate, we still believe that together we can all learn a lot from this case study.

This case study and the associated project guide was developed and written entirely from scratch by TechAcademy's Data Science team. Inga Lasys and Isabel Schnorr developed the project in `R`, while Thilo Leitzbach developed it in `Python`.

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

2.1 What's R?

R is a programming language developed by statisticians in the early 90s to calculate and visualize statistical results. A lot has happened since then, and by now, R is one of the most widely used programming languages in Data Science. You don't have to compile your 'R' code, but you can use it interactively and dynamically. Such an approach makes it possible to quickly gain basic knowledge about existing data and display it graphically.

R offers much more than just programming. The language provides a complete ecosystem for solving statistical problems. A large number of packages and interfaces are available, which you can use to expand the basic functionality of the programing language to, say, create a COVID-Tracker application.

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 start programming. However, it is challenging to carry out an entire project with a "raw" version of R. That's why there is RStudio, a free Integrated Development Environment (IDE) for R.

Such IDE includes many essential features that simplify programming with R. Among other things, an auto-completion of your code, a friendly user interface, and many expansion options. Think of R as your car's engine. And think of RStudio as your car's dashboard that shows fancy metrics, has a radio and allows you to adjust air-conditioning!

Experience has shown that installing R and RStudio locally on your computer takes some effort. Fortunately, RStudio also has a cloud solution that eliminates these steps: RStudio Cloud. You can edit your project in the same IDE in the browser without any prior installations on your computer. You can also easily switch your project from a private to a public project 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 can easily exchange ideas with 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 the courses on DataCamp. That 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 at TechAcademy. As a beginner, please stick to the courses of the "beginner" program. Ambitious beginners can, of course, take the advanced courses afterward. However, it would be best if you worked through the courses in the order we listed them.

The same applies to the advanced courses. Here, too, you should finish the specified courses in the given order. Since it can, of course, happen that you have already mastered the topics of an advanced course, you can replace some courses. If you are convinced that the course does not add value to you, feel free to replace it with one of the courses in the "Exchange Pool" (see list below). However, you should not pursue an exchange course until you finish all chapters from the advanced course: "Intermediate R."

To receive the certificate, both beginners and advanced learners must complete at least two-thirds of the curriculum (6/9 courses). For the beginners, this means until – and including – the course Cleaning Data in R (4h) and for the advanced until –and including – Modeling with Data in the Tidyverse (4h). In addition, you should complete at least *two-thirds* of the project tasks. After completing the curriculum and the project's (minimal) requirements, you will receive your TechAcademy certificate!

R Fundamentals (Beginner Track)

1. Introduction to R (4h)
2. Intermediate R (6h)
3. Data Manipulation with dplyr (4h)
4. Introduction to Data Visualization with ggplot2 (4h)
5. Introduction to Importing Data in R (3h)
6. Cleaning Data in R (4h)
7. Working with Dates and Times in R (4h)
8. Exploratory Data Analysis in R (4h)
9. Introduction to Statistics in R (4h)

Data Science in R (Advanced Track)

1. Intermediate R (6h)
2. Cleaning Data in R (4h)
3. Intermediate Data Visualization with ggplot2 (4h)
4. Exploratory Data Analysis in R (4h)
5. Intermediate Regression in R (4h)
6. Modeling with Data in the Tidyverse (4h)
7. Time Series Analysis in R (4h)
8. Supervised Learning in R: Classification (4h)
9. Machine Learning with caret in R (4h)

Data Science in R (Advanced Track) – Exchange Pool

- Interactive Maps with leaflet in R (4h)
- Interactive Data Visualization with plotly in R (4h)
- Multiple and Logistic Regression in R (4h)
- Supervised Learning in R: Regression (4h)
- Unsupervised Learning in R (4h)
- Machine Learning in Tidyverse (5h)
- Support Vector Machines in R (4h)
- Introduction to Writing Functions in R (4h)
- Writing Efficient R Code (4h)
- Optimizing R Code with Rcpp (4h)

2.1.3 Helpful Links

- RStudio Cheat Sheets
- RMarkdown Explanation (to document your analyses)
- StackOverflow (forum for all kinds of coding questions)
- CrossValidated (Statistics and Data Science forum)

2.2 What's Python?

`Python` is a dynamic programming language. You can execute the code in the interpreter, so you do not have to compile the code first. This feature makes `Python` very easy and quick to use. The excellent usability, easy readability, and simple structuring were and still are core ideas in developing this programming language.

You can use `Python` to program according to any paradigm, whereby structured and object-oriented programming is most straightforward due to the structure of the language. Still, functional or aspect-oriented programming is also possible. These options give users significant freedom to design projects the way they want and great space to write code that is difficult to understand and confusing. For this reason, programmers developed specific standards based on the so-called `Python` Enhancement Proposals (PEP) over the decades.

2.2.1 Anaconda and Jupyter

Before you can use `Python`, you must install it 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 no longer supported – and version 3, we decided to work with version 3.6 or higher.

One of the easiest ways to get Python and most of the best-known programming libraries is to install Anaconda. There are detailed explanations for installing 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 significant advantage of this is that you can quickly write shortcode 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 this DataCamp course first. There you will get to know many tips and tricks that will make your workflow with Jupyter much easier.

To make your work and, above all, the collaboration more accessible, we are working with the Google Colab platform that contains a Jupyter environment with the necessary libraries. You can then import all the data required 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 topic brings us to your curriculum in the next section!

2.2.2 Curriculum

The following list shows the required DataCamp courses for the Data Science with `Python` Track at TechAcademy. As a beginner, please stick to the courses of the "beginner" program. Ambitious beginners can, of course, take the advanced courses afterward. However, it would be best if you worked through the courses in the order we listed them.

The same applies to the advanced courses. Here, too, you should finish the specified courses in the given order. Since it can, of course, happen that you have already mastered the topics of an advanced course, you can replace some courses. If you are convinced that the course does not add value to you, feel free to replace it with one of the courses in the "Exchange Pool" (see list below). However, you should not pursue an exchange course until you finish all chapters from the advanced course: "Intermediate Python."

To receive the certificate, both beginners and advanced learners must complete at least two-thirds of the curriculum (6/9 courses). For the beginners, this means until – and including – the course "Joining Data with pandas (4h)" and for the advanced until –and including – "Introduction to Linear Modeling in Python (4h)." In addition, you should complete at least *two-thirds* of the project tasks. After completing the curriculum and the project's (minimal) requirements, you will receive your TechAcademy certificate!

Python Fundamentals (Beginner Track)

1. Introduction to Data Science in Python (4h)
2. Intermediate Python (4h)
3. Python for Data Science Toolbox (Part 1) (3h)
4. Introduction to Data Visualization with Matplotlib (4h)
5. Data Manipulation with pandas (4h)
6. Joining Data with pandas (4h)
7. Exploratory Data Analysis in Python (4h)
8. Working with Dates and Times in Python (4h)
9. Introduction to Importing Data in Python (3h)

Data Science with Python (Advanced Track)

1. Intermediate Python (4h)
2. Python Data Science Toolbox (Part 1) (3h)
3. Python Data Science Toolbox (Part 2) (4h)
4. Cleaning Data in Python (4h)
5. Exploratory Data Analysis in Phyton (4h)
6. Introduction to Linear Modeling in Python (4h)
7. Statistical Thinking in Python (Part 1) (3h)
8. Time Series Analysis in Python (4h)
9. Machine Learning for Time Series Data in Python (4h)

Data Science with Python (Advanced Track) - Exchange Pool

- Interactive Data Visualization with Bokeh (4h)
- Data Visualization with Seaborn (4h)
- Supervised Learning with scikit-learn (4h)
- Linear Classifiers in Python (4h)
- Unsupervised Learning in Python (4h)
- Introduction to Deep Learning in Python (4h)
- ARIMA Models in Python (4h)
- Web Scraping in Python (4h)
- Writing Efficient Python Code (4h)
- Writing Efficient Code with pandas (4h)

2.2.3 Helpful 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

Participant's Section:

- https://www.tech-academy.io/teilnehmerbereich

2.3.1 Prelude

Now that you have learned the theoretical foundation of Data Science 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 of this project in the following chapters of this project guide. If you still feel a little unsure about how to start off, there will be a **Coding Introduction** on **May 4, 2022**, where we will give you a general sense on how to start off with `Python` or `R`. Of course, we will also describe the project. We will discuss everything you need to know during the first **Coding Meetup**, which will take place on **May 18, 2022**. After that, your 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 your TechAcademy certificate, you must solve at least two-thirds of the "Exploratory Data Analysis" part of your Data Science project if you are a beginner. We added a "Seasonality and Geographic Differences – Application of Statistical Methods" for our advanced participants. In addition, you should complete two-thirds (6/9 courses) of the respective curriculum on DataCamp, as mentioned. You can find more detailed information about the curriculum in the "Curriculum" section of the different programming languages above.

2.3.2 Coding Meetups

To give you a little overview on the dates of our Meetups, you can find all the dates in one go here! - **27.04.2022**, Kick-Off Event - **04.05.2022**, Coding Introduction - **18.05.2022**, Coding MeetUp 1 - **01.06.2022**, Coding MeetUp 2 - **15.06.2022**, Coding MeetUp 3 - **29.06.2022**, Coding MeetUp 4

2.3.3 Deadline

For both your DataCamp courses and the project files hand-in the deadline is **03.07.2022, 23:59**

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, since we feel like it is 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 DataCamp, as well as developing skills in researching code independently. Always remember, **Google** is the best tool in your kit! 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. Since data science concepts are independent of specific programming languages, we will describe the general approach in a text chunck. Having understood the bigger picture and starting with the tasks, you will find language-specific tips and tricks in visually separated boxes (`R` -Track : bluw-borderd boxes, `Python` - Track : yellow-bordered boxes). From time to time, it might be interesting to check out the other language – though you can code the exact solutions in both, they sometimes have a different approach to an identical problem. 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 do not have to look the same way as ours do, just keep in mind which information is necessary for a meaningful plot. Furthermore, you can find helpful links in the introductory chapters (2.1.3 & 2.2.3 Helpful Links), where your questions might already have been answered. If not, and in the unlikely case that even Google cannot help you, the TechAcademy mentors will help you via Slack or directly during the coding meetups. At the end of the project guide (6. Exercise Checklist) 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.

3.2 What is this Project About?

As mentioned in the beginning and as per our project title, our data deals with honey production and bee colonies, specifically in the United States. This data was collected by the USDA National Agricultural Statistics Service, and compiled on kaggle. If you have any doubts about the meaning of the variables, you can have a look at the data documentation provided on kaggle. In addition, we have included weather data since both bee colonies and honey production are closely related to climate conditions. In the analogy of the typical data science workflow, we have split this project into two parts.

3.3 Exploratory Data Analysis – Getting to Know the Data Set

As a first step, you will get to know the data set and learn how to perform an Exploratory Data Analysis (EDA). This step means you will describe the data, get to know its different variables,

transform it, and answer related questions. For this purpose, you often use graphical tools and various visualisations like box plots or histograms. You can also use tables to look at your data and show your readers what you have been doing. We structured the first part of the project to let you 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 your TechAcademy certificate. However, if this first part inspires you to step up your analytic skills or your coding language has you in a spell, we encourage you to work on the second part! Once more: If you get stuck, **Google** and StackOverflow are amazing problem solver (besides the Mentors of course).

3.4 Seasonality and Geographic Differences – Application of Statistical Methods

As a second step the more advanced of you will dive into deeper depth. For this purpose you will analyse the given data with several (advanced) statistical approaches and draw insightful conclusions from them.

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 takes place there.

In your workspace on RStudio Cloud, we have already uploaded an "assignment" for you (Template HoneyAndBees). When you create a new project within the workspace *Class of '22 | 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 (a file that ends with `.Rmd` extension), with which you will be able to create a detailed report of your project. You can convert that file into an HTML document when you have finished coding the project in R. Open the file `Markdown_HoneyAndBees.Rmd` and see for yourself!

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

Next up is importing the data sets. It's best to this **once** on the top level of your notebook / script. You can always copy to new variables and work on slices of the data frames afterwards.

- Honey
- Bees
- Weather

4.1 Exploring Honey and Bees Data

4.1.1 Discovering the Data

Let's start by looking at the bee & honey data sets.

- Have a look at the frequency of the records (is there a difference across sets?)
- Do you see any missing values or data entries that are different from the other entries?
- What are the data formats (e.g. data types like strings or floats but also date formats such as yearly)?
- Look at the unit of every variable, can you make sense of the units?

Write down a couple of sentences to these questions, the goal is to show your readers what you are seeing. Also, comment on any errors or irregularities which you notice and that could be an issue later on in the project. What irregularities you ask? Look at the missing values, logical expressions, numbers, variables types, etc. But do not worry, there is no right or wrong here!

You can import all data files directly from your working directory by, for examples, using the command `bees <- read_csv("bees.csv")`. Use `head()`, `str()`, or `class()` to get an overview. If you need some additional, more general information on how to import data and different data types, check out this cheat sheet.

You can feed the links to the respective data files above to a method of the pandas package (you might want to specify the index column). Check out the resulting `pd.DataFrame` instance with the `head()` method and the `dtpyes` attribute. You can also dig into a specific column with `describe()`. Here's an additional pandas cheat sheet for you to reference

4.1.2 Give some overall statements

Now that you have imported all data sets, lets get into the details of the **honey data**. Since you have already got a feeling by now, it would be interesting to indicate some outstanding features of the set:

- Which state had the most producing colonies ever and in which year? Anything special about that state?
- What state had the lowest price for honey. How low was it?
- What is the total Honey production for 2016?

4.2 Data Cleaning and Useful Transformations

The exercises in this and the next section are about the **bees data** set. In the later stages of the project, we would like to merge the bee data set with the honey data set. To this end, we will now spend some time preparing the bees data set.

4.2.1 Date Formatting

From exploring the data in the previous tasks you might have notice the differences in frequency of the data. The date (format) of the bees set is quarterly, whereas we have yearly honey data. Recall also that dates, even just years, can be formatted differently than just as a string or integer. Maybe you have checked it before, but now is definitely the moment to translate the `Date` column to a fitting date format.

You can use the lubridate package to transform "years" into a date format. To things easier, check out the lubridate package cheat sheet. If you would like to read up on some general information on date formats with R in your spare time, have a look here

`pd.to_datetime()` is what I would look at for example. A corresponding DataCamp resource is section 4 in Working with dates and times in Python. Also, the Data Manipulation with pandas course is of great help for the following exercises.

4.2.2 More Data Types

So far you converted years into a more appropriate date format for further investigation. Have a look at the other columns of the bees data set: Some appear to contain numbers, but are they also of a numeric data type? Check out all columns so you don't miss out on one!

For further calculations, it is safer to convert these columns to a numeric data type. While there are vectorized methods that can convert data types at once, we would like you to write a (really) simple 'for' loop to conduct the conversions.

Have you heard of "sapply" ? This might help you. We recommend you to check out the documentation, which you can access by typing `help("sapply")` into the console, or by browsing the web. Remember, you want to convert multiple columns into the numeric format, so be careful to select all the columns!

You need an iterable to loop over, the semicolon after the keyword, and the indentation of the subsequent lines! Here is a cheat sheet in case you have forgotten.

4.2.3 Converting Units

From the column names you can see that the honey data set is in pounds as we are dealing with an U.S. data set. However, we are devoted followers of the metric system and do not easily understand pounds. Consequently, we would like to translate them to kilogram [kg]. It is best practice to create a new variable from the old one that you are modifying. Therefore, while you are at it, you might want to give your new variables, now in kg, a shorter name no spaces in it. For example "Honey producing colonies (thousand)" could be named "producing_colonies_thousand".

4.2.4 Rounding

Have a look at your column values. Some columns have long decimal numbers. To simplify your data frame, round up to the decimal place you prefer. Please write one or two sentences why you think rounding is O.K. and why you chose the decimal place X or even a conversion to integer. There is no right or wrong here, we want to encourage you to consider the option and find what is best for your analysis!

4.3 Simple Metrics

4.3.1 Missing Values

We now want to look at the bees data set and its missing values. Have a look how the missing values are encoded. Decide on how you want to treat them and give us a detailed explanation why you have decided to treat the missing values the way you did. Keep in mind that these

transformations affect all your subsequent work on the data and may bias your findings. We recommend you to brush up on your statistics knowledge and conduct some research on how to treat the missing values, as awlways, Google is a great place to start! Sometimes you are also lucky and can identify a pattern that will ease your pain of treating the values and deciding for a strategy.

There are many possibilities: From excluding the rows to forward / backward fills or even inserting an average / median.

We highly recommend the Datacamp course on missing values in R for this exercise.

We highly recommend the Datacamp course on missing values in Python for this exercise.

4.3.2 Lost Colonies

Calculate a new column holding the information of the percentage of lost colonies, in reference to the maximum colonies each quarter (it is as simple as it reads). Give the new column an appropriate name and decide whether you would like to keep this column in decimal numbers or percentage points.

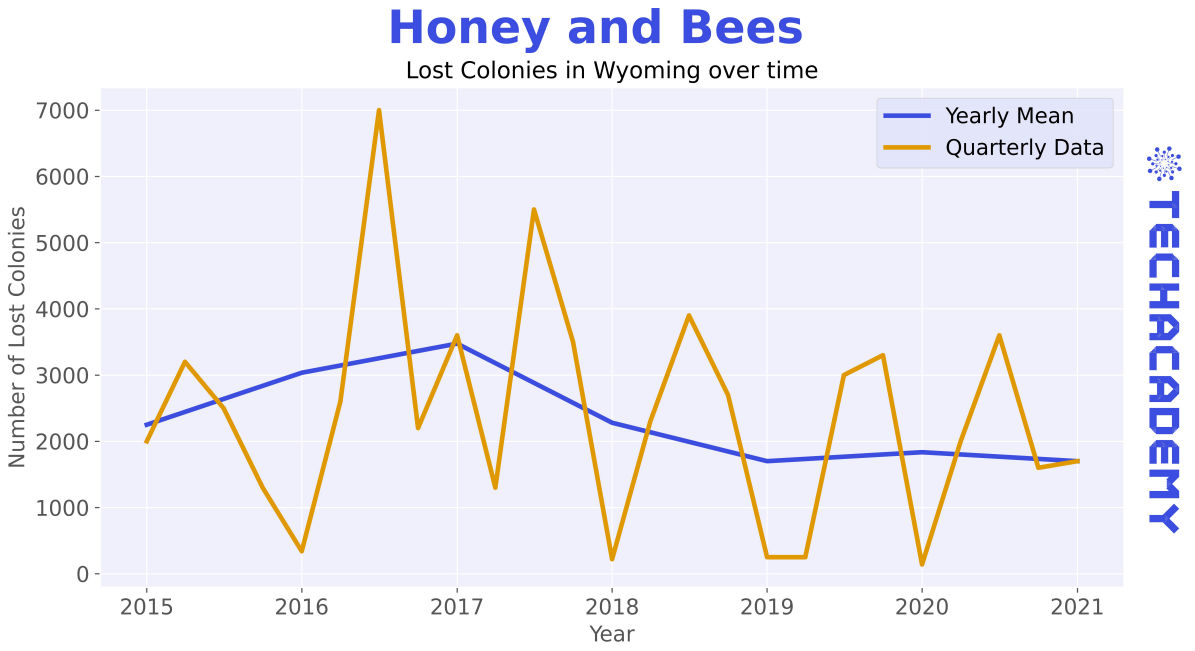
4.3.3 Group Means

To make things easier, we want to merge the honey and bees data set to analyze them jointly. As you probably have seen, both data sets have different frequencies. Therefore, we need to aggregate the quarterly bee data set to a yearly frequency. In the end, data for just the year from 2015-2020 for each state should be left. Accordingly, you should group by state and quarter and replace the observations by a yearly aggregate.

We have decided that it is best to aggregate the data (over the four quarters) by using the mean. While you are at it reproducing our steps, pay attention to how you are rounding your results. After you are done, please print the table of your aggregated data frame, and answer the following two questions:

- What are potential hazards of reducing data like this?
- What other measures could you use?

You may refer to the following graph.



Do not fret if you are unsure, there is no absolute answer to these questions since your approach most likely depends on what you are analyzing / trying to show or understand. We would still like you to give it a solid thought, which we want you to communicate to us, your readers!

Following the tidyverse approach with `dyplr` is the cleanest in this case. Use `mutate`, `group_by`, `select` and `summarize`. In the `summarize` call you can add multiple functions together. If you are unsure, check back with the dplyr cheat-sheet.

With two-dimensional (panel) data your `.groupby` statement on the data frame should consider both index values and the time dimension. The resulting `GroupBy` object supports various aggregation methods out-of-the-box.

4.4 Merging Data Sets

When preparing to merge data sets, it is wise to check the naming of columns again. You should avoid having columns with the same name to avoid confusion, except for variable that you are merging by. Keep in mind by which variables you want to merge by beforehand, if you are unsure have a peek at the next merging exercise to clarify what we mean! Please give your bees data columns appropriate names (we recommend names where empty spaces are replaced with an underscore "_"), if you have not done so for the honey data set please do the same.

Column names can be adjusted with tidyselect's `rename` function.

Column names can be replaced very neatly by passing a dictionary `{"x": "y", }` to a certain pandas method.

4.4.1 Joint Honey and Bees Set

Do an inner merge of the two sets by exploiting the index / State names and the respective date columns. Closely compare the sizes of the individual data frames and of the aggregate one.

- Do they match (*hint: they do not*)?
- Why not and what states are missing from which individual data frame?

As we would like you to follow tidyverse syntax, check out the dplyr merge functions cheat sheet and join documentation.

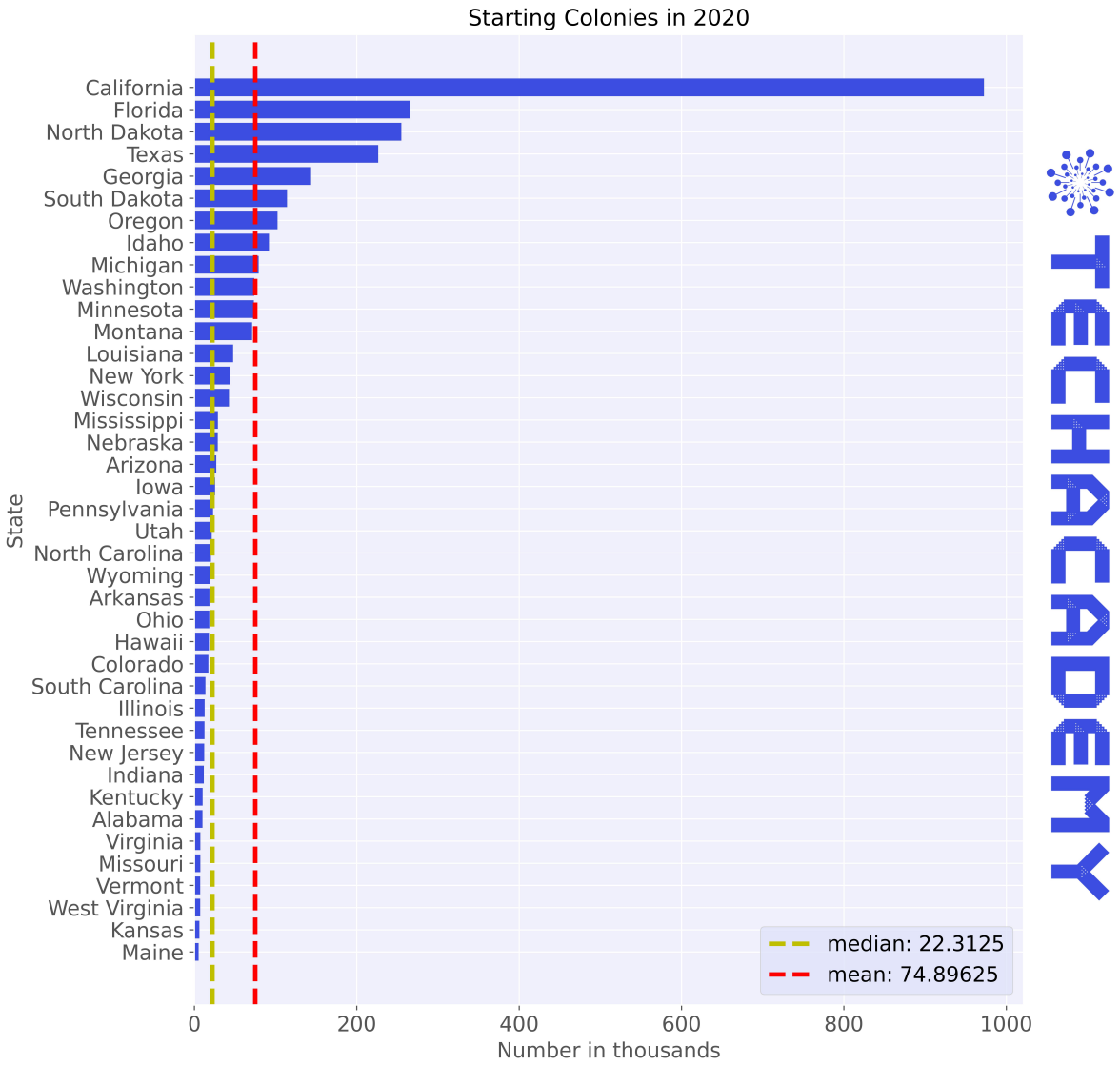
Check out the pandas merge function documentation. If it helps you, visualize what the identifiers in each table are and how they relate to each other. The DataCamp course Joining Data with pandas (4h) might also get you started.

4.5 States Bar Plot

Now that you have merged the data sets into one big data set, we want to continue with the big data set (not big data yet! :D). Use the aggregated (yearly) data set and create a bar plot with all states plotting starting bee colonies from the lowest to the highest value. Please also include the average and the median of the observations. Do not forget to label your graphs in a meaningful way!

You can decide for yourself whether you want to aggregate over all years or over a single year of your choice, but please make sure to let your audience know what you did and can observe.

Honey and Bees



The key commands that you can use for this within `ggplot()` are `geom_bar()` and `coord_flip()`. Check back on the ggplot2 cheat sheet if you are feeling unsure about the labeling of the plots! Also, you might want to store the median and mean seperatly and refer to them in your ggplot call.

In case you want to filter a certain year check out `.loc` indexing. Afterwards you are looking for matplotlib's horizontal bar plot. Should you get stuck adding the lines to the plot, you can look at the following stackoverflow question. Oftentimes, when you want to create plots that are a little bit more elaborate, you will find an initial statement like this:

```
# import statement on top, of course
from matplotlib import pyplot as plt
fig, ax = plt.subplots()
```

The trick is to add to the generated `Figure` and `Axes` object subsequently. For further resources the course Introduction to Data Visualization with Matplotlib (4h) is a good place to start!

4.6 Splitting the Data Set

Let's do some individual work! We ask you to please compare the compare to work wihtin in your group in the end. But first things first, your group needs to split a the data into subsets, one subset for each team member. You should have 40 States in your merged dataset. If you are working in groups of four team members, each of you would be looking at 10 states, neither of your states should be included in another team member's subset! If your team does not contain four members and you cannot come up with a way to split your data or are unsure about your current group dynamic, please consult your mentor.

The code for your 4 team member split is: `dataset$Group <- c(rep(1,60),rep(2,60), rep(3,60), rep(4, 60))`. If your group does not contain 4 members, this code can be easily to adjusted.

Make sure to sort your state index first, afterwards make use of `numpy's array_split(df, x)` function (it never hurts to make a sanity check/print afterwards).

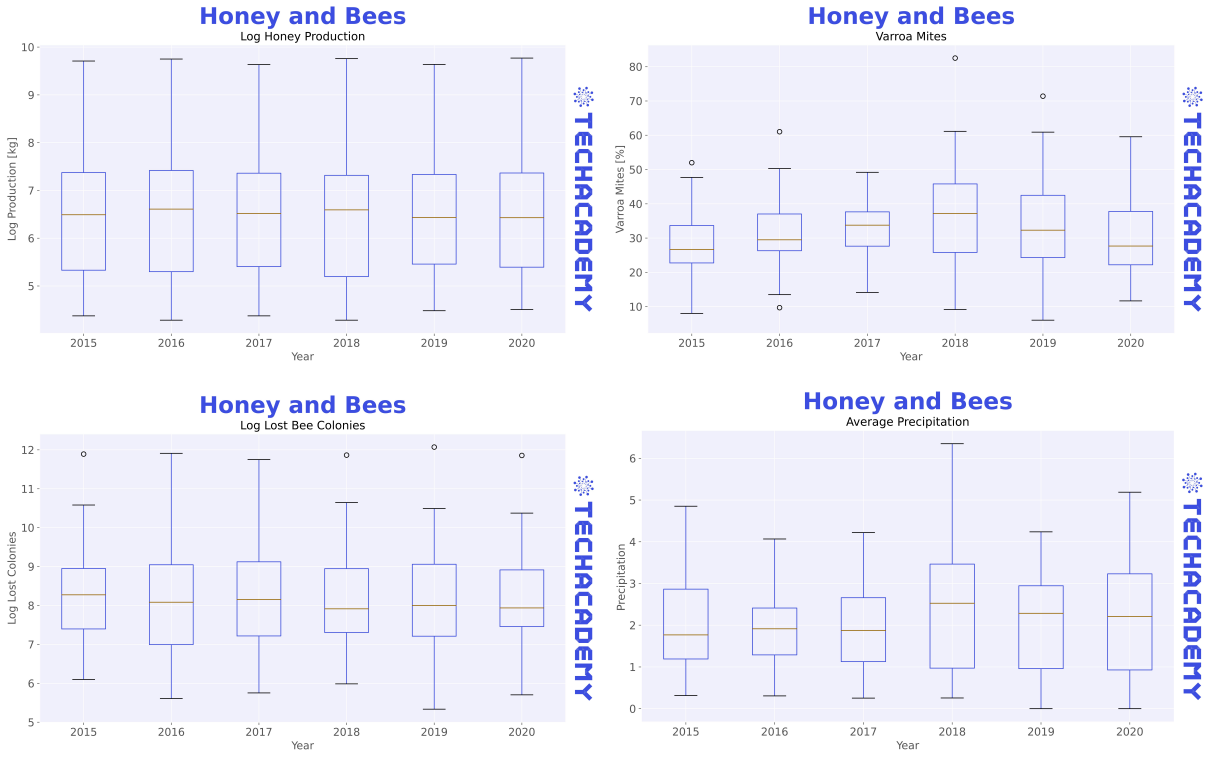
4.6.1 Yearly Boxplots

Great split! Next, let's visualize the split data with beautiful boxplots. Generate four Boxplot aggregated for each year. They should look at Honey Production, Parasite: Varroa Mite, Precipitation, and Bee Colonies for the subset of each team member. In the end we ask you to compare your four boxplots with the ones your team members have created. Keep in mind to communicate the same scaling, otherwise a visual comparison will not work out.

Tip: It will be six boxes for each boxplot since we look at six years, as shown in the example.

Note that applying the logarithm to the colony values can be a very useful thing to do. With this transformation you can achieve a reasonable visualization without excluding a state, and since California is a strong outlier in our data set we have decided to do so. To convince yourself, you can easily plot both boxplots (with and without log transformation) of maximum colonies. Can you see the difference? When applying logs, make sure to label your graphs accordingly!

On a side note, applying the logarithm "normalizes" the distribution of values (frequently seen in empirical statistics) assuming that the underlying values approximately follow a log-normal distribution. A very superficial take on this can be found here (5 min read).



Again, `ggplot()`, and this time `geom_boxplot()`, will be your best bet!

Surprise, you are looking for the `.boxplot(...)` function!

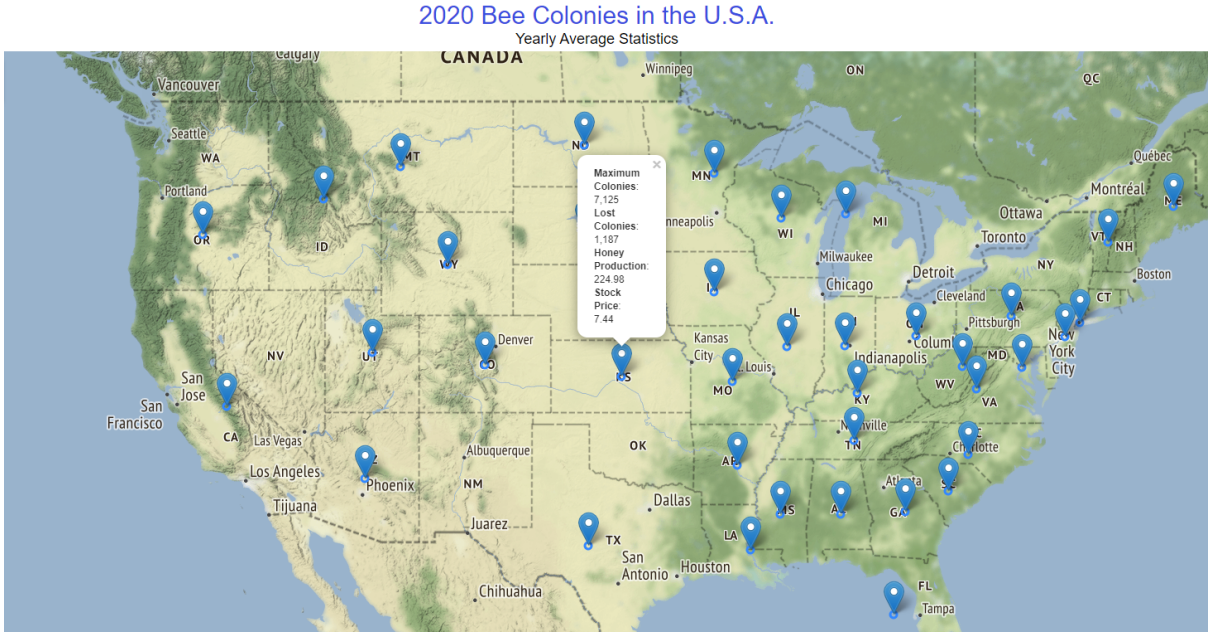
4.7 Visualization with Maps

Last but not least, we want to plot a more appealing visualization.One that gets you excited! The idea is to combine the geographic information on the states with the numeric data to provide a rich visual that reflects the two aspects.

4.7.1 Geographic Map with State Pop-ups

We start by exploiting the coordinates provided with the data and our joint honey and bees set to obtain a dynamic map of the United States.

The map should reflect information about some key variables like the maximum- and lost colonies, as well as the honey production and stock price in one year of your choice.



This time, we created the plot using the leaflet package. Why don't you try to find information on the package online?
You can create a basic map with leaflet and add additional layers by following these simple steps:

```
m <- leaflet() %>%
      addTiles() %>%
      addMarkers(lng= ,
                 lat= ,
                 popup= )
```

The popup is where the magic happens! Can you make the map show the mentioned four key variables, and one more variable that you find important for such a plot?

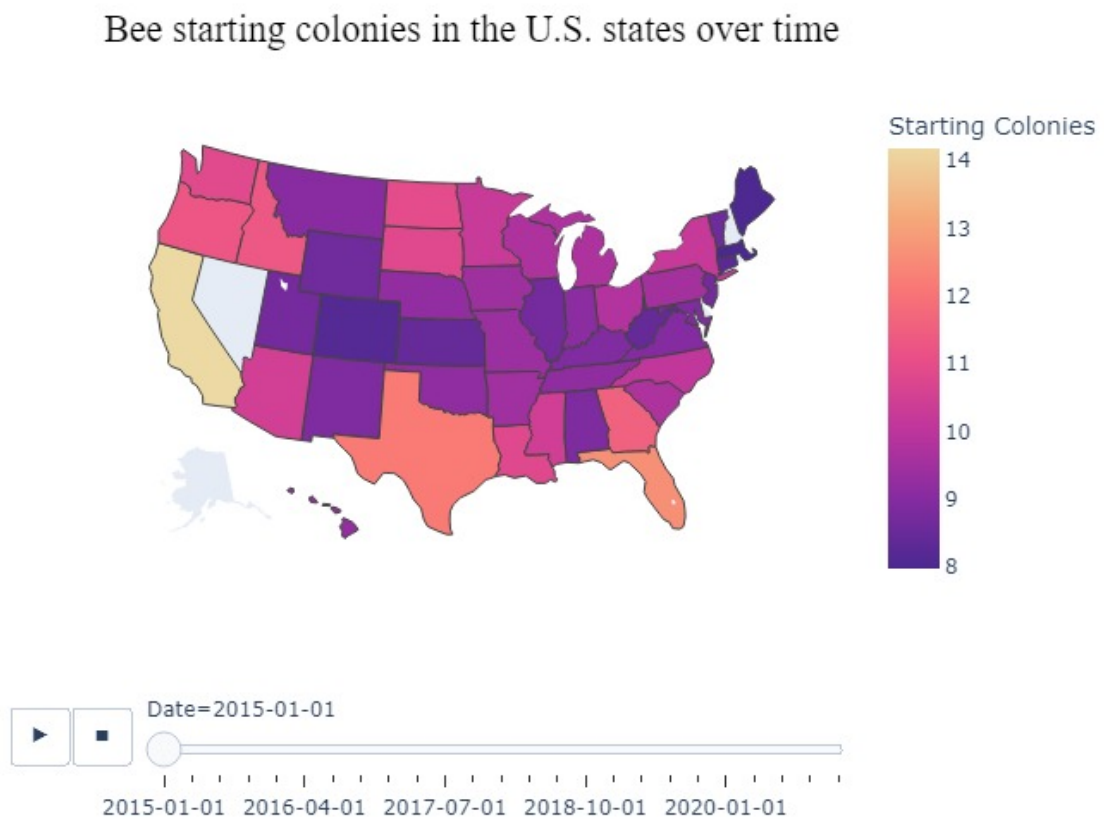
We are going to use yet another library, this one is called `folium`.
For this task you need to work with its documentation which you can find online.
Use folium's Circle (Marker) to draw a circle for each state at its centered coordinates (latitude, longitude). You will need to implement a for-loop to iterate over all states.

```
import folium
# Initiate the map
m = folium.Map(
    location=[39, -101],  # Somewhat central U.S.A
    zoom_start=3.5,
    tiles='Stamen Terrain'  # one of many Map Styles
)

# Use a for-loop to plot circles
for idx, row in df.iterrows() :
    # Your code here

m  # Displays the map
```

4.7.2 Animating Changes over Time

Since we have looked at a single year but multiple variables in the previous plot, we would like to mix things up and look at transitions here. To do so, we want to plot a heat map showing the transition of log starting colonies over time for each state, creating a dynamic plot.

Note again that we are using the log transformation for visualization purposes.



Now that you have familiarized yourself with the leaflet package, let's switch things up! For this map we will be using the `plotly` package, specifically, a choropleth map. If you would like to have some further information, have a look here. However, be careful what kind of map we are asking you to plot!
To add a dynamic slider representing years to your map, you would need to specify `add_trace(frame = )` correctly.

There is a really cool utility built-in to plotly express so you do not have to specify the tedious coordinates for your plots. In fact, the animation above is essentially one single function call.
For animation, the `choropleth` function takes an *animation_frame* and *animation_group* argument which you need to specify explicitly besides your data frame.

4.8 Surprise Us!

You discovered anything cool and / or want to experiment with a new visualization technique to the deliver a message? Here is the right place to put down anything you did and would like to do with the project that did not quite fit in the other exercise sections. If you are looking for some visualization inspiration, check out this page.

Congratulations! Based on your work with fundamental data transformations and many visualizations, you now have a solid understanding of the honey and bees data sets! With this, you have completed the *EDA* part of the project! For a TechAcademy certificate you should complete *two-thirds* of the above. Don't forget to send your project results to our project submission email address before the deadline (**03.07.2022, 23:59**). Thanks for being a part of TechAcademy!

If you are in the advanced track your coding journey goes on with the next section!

5 Seasonality and Geographic Differences – Application of Statistical Methods

5.1 Merging Bees and Weather

Hopefully the first part was easy for the more advanced of you. Regardless, let's continue with some more challenging exercises!

We want to investigate how diseases and different climates affect our beloved bee populations. To that end, you now need to merge the two data sets "Bees" and "Weather", and we will no longer be using the data that we have previously merged. As in the previous exercie, both data sets have different frequencies, can you spot the difference? To merge you will need to aggregate the daily weather data set to quarterly data. Be carefull when aggregating the data though, it might not be clever to replace period-specific max/min values with a mean!

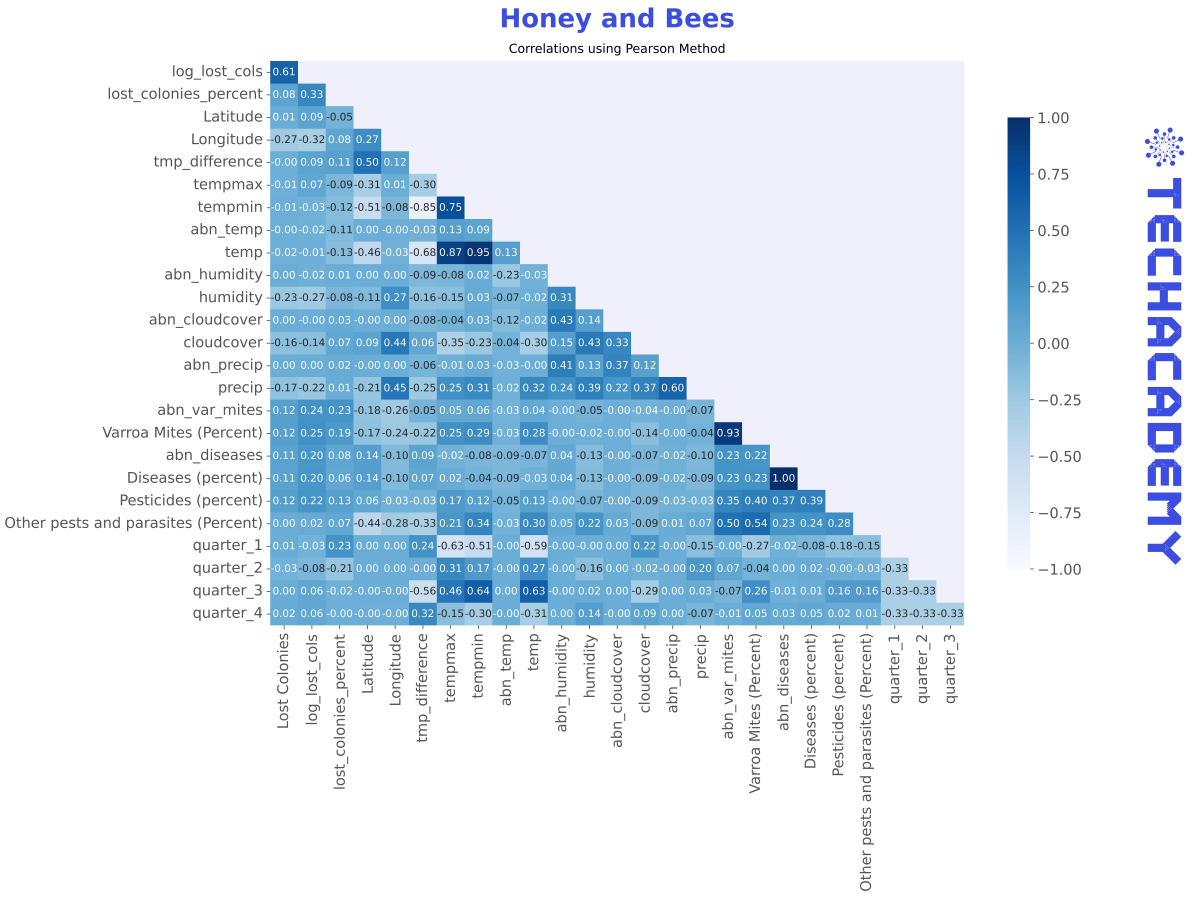
If you get lost, you might want to compare your approach to your solution of 4.4.1.

Please note that there is no weather data for **Florida** and **New Mexico**, you should exclude these states.

5.2 Correlations

5.2.1 Naive Approach

After merging the data sets into, let's start with some analysis. To cover our basis, we would like to start off by plotting a simple correlations plot for the variables of interest. You can pick which variables you would like to plot! This could be lost colonies, different parasites, and weather indicators for example.



- We have a question for you: Why is a naive correlation matrix across states likely a bad idea? Think for example about omitted variables and how information is lost when you aggregate fundamentally different groups.

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.

A handy library for plotting correlation matrices is the seaborn library:

```
import seaborn as sns
...
```

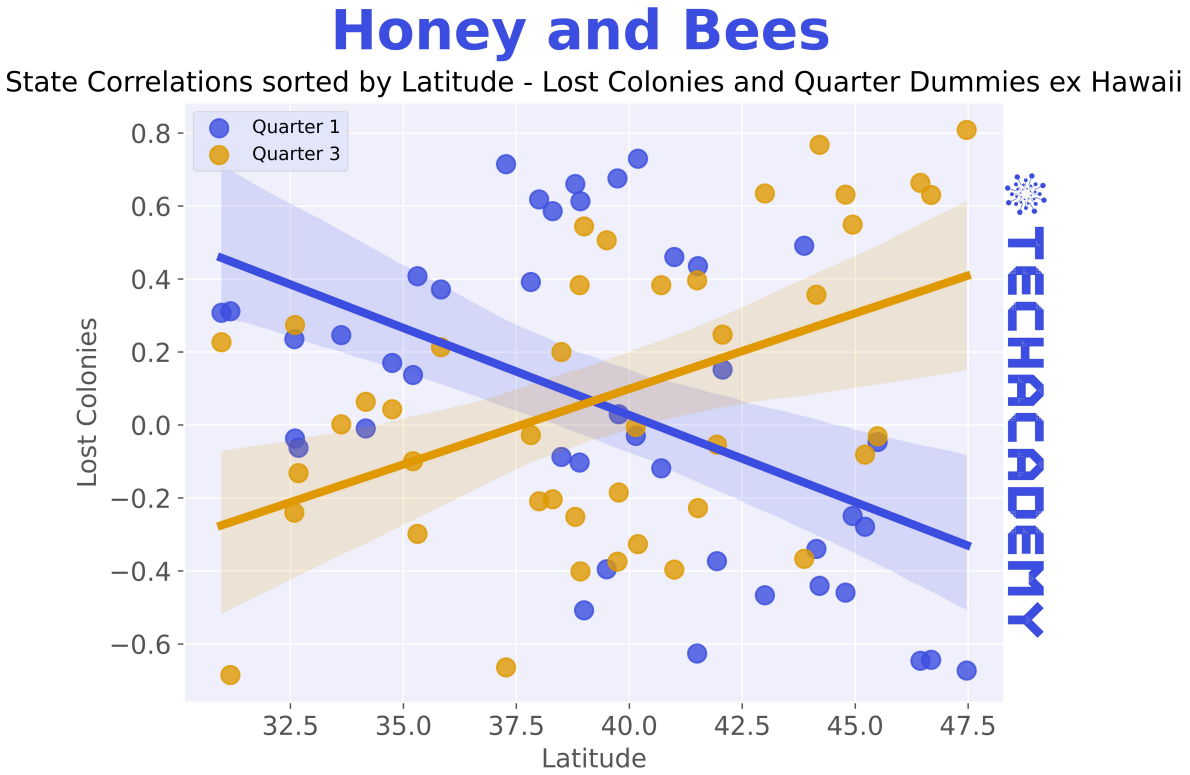
You can use its `pairplot` method and pass on the data frame with the selected columns to visualize distributions and correlations.
Alternatively/Additionally, you may want to plot a heatmap with `sns.heatmap(...)` which makes it even easier to see correlations.

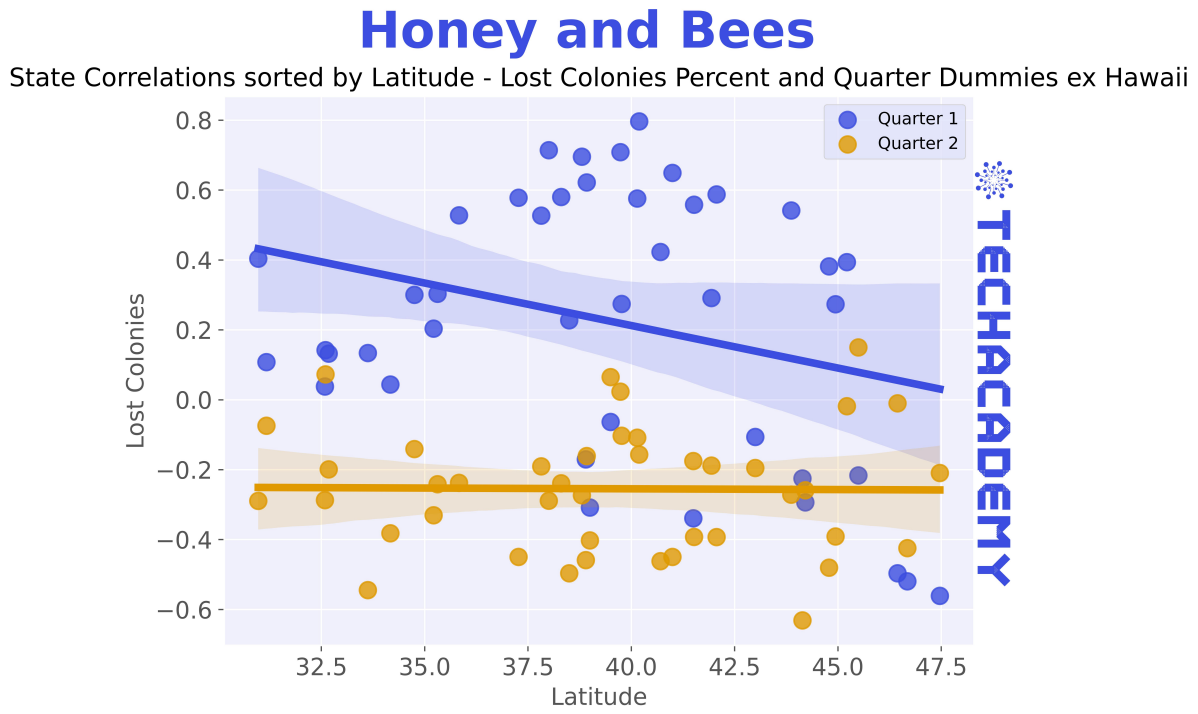
5.2.2 Statewise Correlations - Seasonality & Geography

Having checked a very simplistic correlation, we need to do some thinking. Our cross-section of states is characterized by heterogeneous geography and climate. Naturally, this also implies seasonality in our quarterly data set. However, due to the large szie of the United States, we must not forget that for some southern states, in our data Hawaii's, seasons are less pronounced than they are in North Dakota (if you do not believe us, check the max temperature differential …).

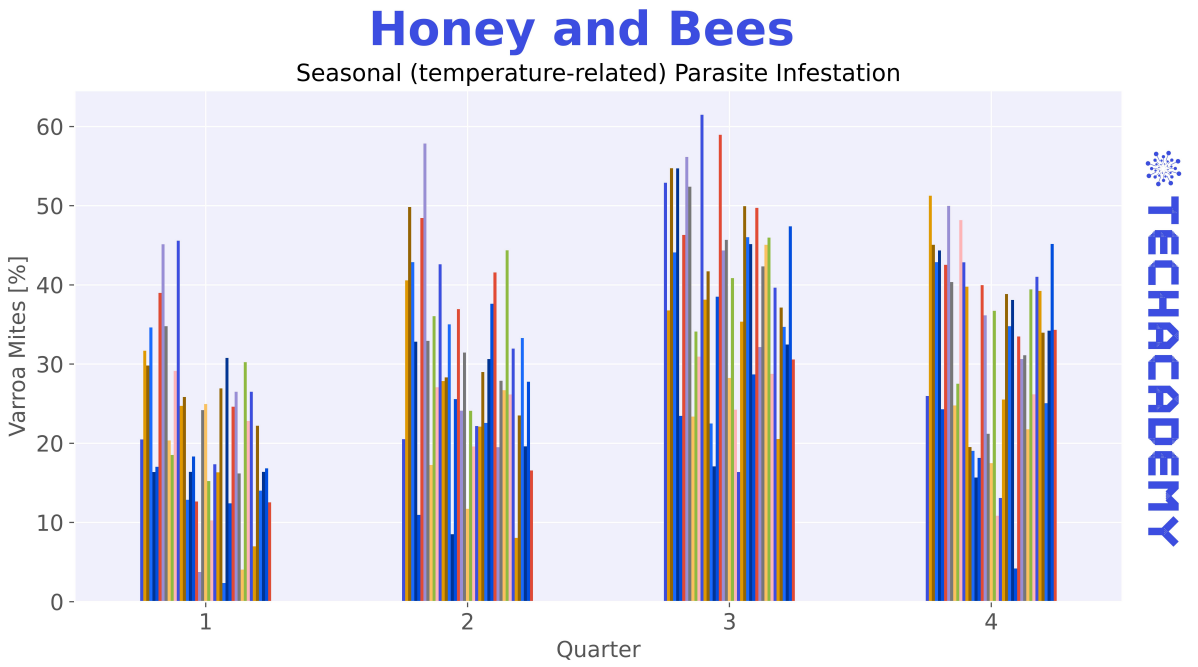
To visualize this, you want to compute statewise correlations for either "Lost Colonies", "log Lost Colonies" or "percent Lost Colonies" and sort them by latitude. As always, do not forget to label the graphs in a meaningful way, and please explain how the results of your plot can be interpreted to your audience!

Below are some examples:





Note, that there might also be interaction effects, e.g. the season/climate influences the parasite population or the like.



We recommend you using the tidyverse approach. From previous exercises you should be well versed in using `dplyr`commands. A combination of `group_by()`, `filter()`, `select()`, and `arrange()` should do the job nicely! If you unsure, check back with the dplyr cheat sheet.

You should be fairly familiar with `groupby` statements by now. To sort by an underlying property of the index values, it might be of help to calculate correlations in one data frame and get a list of ordered states from another using `.reset_index()`, `.drop_duplicates()`, and `.sort_values()`. Subsequently, you can easily use `.reindex()`.

5.2.3 Demeaned Variables

Create at least two own indications that you might want to use in the following regressions which can mitigate some of the issues:

- One measure that subtracts a quarter aggregate (median, mean …) over ALL states but by quarter (4 values in total) from each observation with that respective quarter
- One measure that subtracts a quarter aggregate by state AND quarter (values equal to the number of states * 4) from all observations with the respective state and quarter

Example for the 2nd: You create a measure of "abnormal" temperature by subtracting the average temperature of a given state in each quarter from the state's quarter observation in all years (6 years in total) in the data set.

Additionally, add quarter dummies to your data frame.

Add tip

You best work with a `Multiindex` and employ the `.dt.quarter` property of `Datetime` columns. After aggregating, you can merge the values back as an additional column and simply subtract as well as reassign the result.

5.3 Pooled OLS for State Panel Data

As we investigated correlation patterns above, we saw that panel data is not straightforward to analyze. Especially since we are only equipped with quarterly data of mere 6 years - that's not a lot. But you have just added a few more "normalized" variables, so we can get started with our regressions in the hope to make some interesting findings.

With panel data, the basic extension of OLS (Ordinary Least Squares) regression is Pooled OLS. A basic Pooled OLS regression equation could look like this (a simple extension of classic OLS with state effects α_i and error term $\epsilon_{i,t}$:

$$percent\_lost\_colonies_{i,t} = \mu + \beta_{1,i} \cdot varroa\_mites_{i,t} + \beta_2 \cdot quarter_1 + \cdots + \alpha_i + \epsilon_{i,t}$$

5.3.1 Seasonality Specification

Compute pooled OLS regression estimates that reflect only seasonal aspects. Subsequently, you will extend this model below. At the end of this section is a [#table] that may serve as an indication of what the end result could look like.

Add tip

A classic package for inference is statsmodels. Standard OLS is pretty straightforward but not needed here. linearmodels is an extension library that supports the analysis of panel data.

5.3.2 Adding Geographic Aspects

Extend your specification from above with at least one variable that reflects each states "unique" geography. Whenever you add more variables be careful to avoid (multi-) collinearity (= your regressors are highly correlated among themselves).

5.3.3 Adding indicators for weather and parasite infestation

Last but not least add variables (maybe some of the ones created above) that can explain even more, considering for example weather extrema and parasite infestation.

When you compare all the R^2 of the different specifications with special attention to *within* and *between* R^2 what do you observe?

Table 1: **Analyzing Lost Colonies using Pooled OLS**

	percent_lost_colonies			
	(1)	(2)	(3)	(4)
Constant	11.37***	20.48***	18.58***	16.98***
	(42.775)	(11.252)	(10.590)	(9.281)
$Quarter_1$	3.13***	3.13***	3.03***	2.87***
	(4.841)	(4.909)	(4.890)	(4.80)
$Quarter_2$	−2.42***	−2.42***	−2.77***	−3.04***
	(−5.353)	(-5.411)	(−6.350)	(−6.844)
Latitude		−0.23***	−0.18***	−0.17***
		(-5.115)	(−4.250)	(−4.041)
abnormal Varroa Mites [%]			0.11***	0.11***
			(9.458)	(9.433)
precipcover				0.14***
				(3.219)
abnormal temperature				−0.61***
				(−2.78)
R^2 overall	.075	.0952	.166	.185
R^2 within	.093	.093	.129	.1487
R^2 between	.000	.100	.312	.332
Observations	42	42	42	42
Time Periods	24	24	24	24
Standard Errors	Clustered	Clustered	Clustered	Clustered

t statistics in parentheses
* $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$
Note: The data set is balanced.

5.4 Alternative Model

You are completely free here. You want to check out a machine-learning approach and compare? This is your chance! All you have to do is estimate an alternative model (so not pooled OLS again) whose properties you can compare in a later stage.

5.5 Model Evaluation

Model evaluation is always important, even more so when using different frameworks that you would like to compare. At the end of the day, we are interested in the model that reflects our empirical observations best.

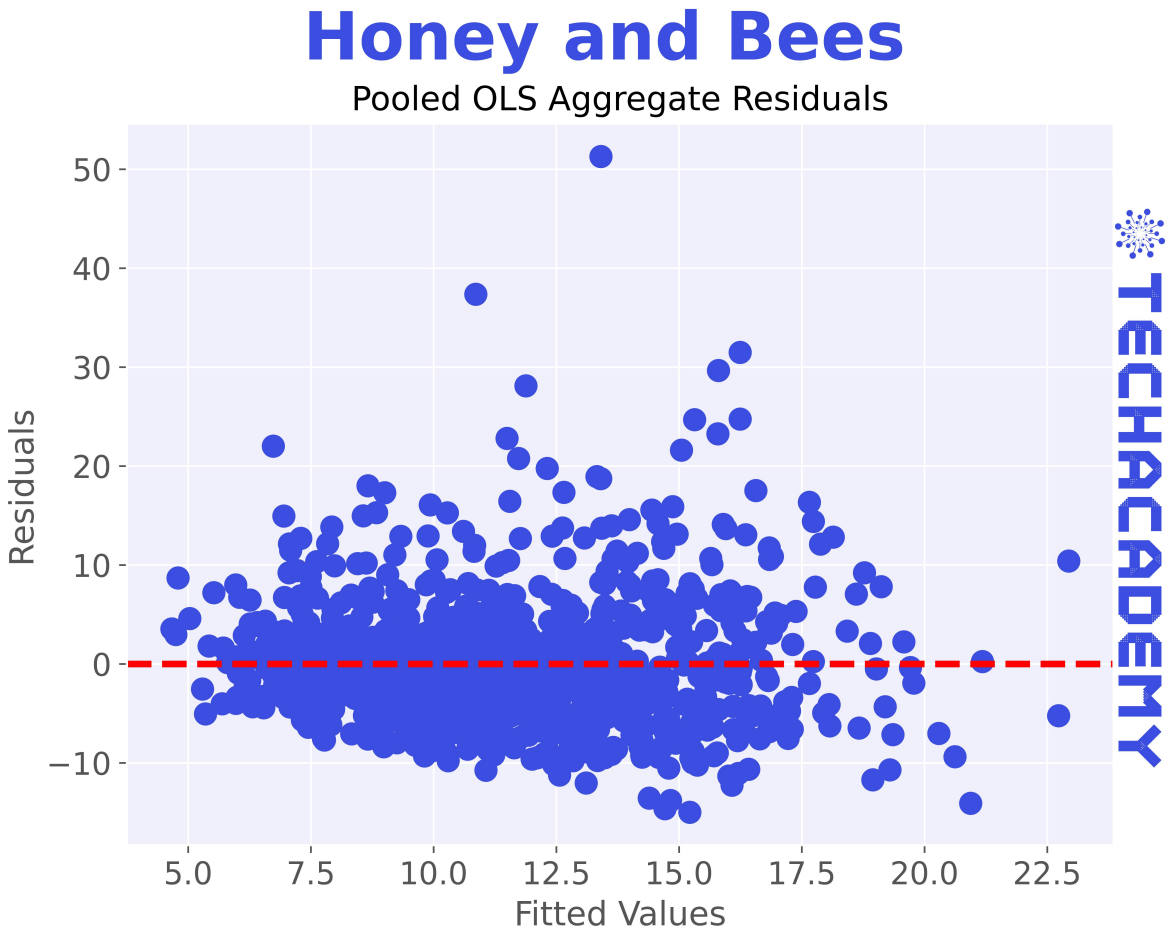
5.5.1 Residuals Distribution (in-sample)

When computing regression estimates the residuals as unexplained fraction of your model are the most important aspect to look at. Especially since each model assumes a certain distribution (most of the time we assume a normal distribution). Since we just assume a distribution, we must check whether these assumptions hold. Typical related questions are whether they are auto-correlated and whether the standard errors computer are robust (e.g. test for the presence of heteroscedasticity etc.).

Plot the residuals of one of your pooled OLS specifications and the ones of your alternative model. Do they differ fundamentally?

Add tip

Your fitted models will have something like a `.predict()` method which also yields the in-sample fitted values. Moreover, the fitted model will automatically compute the residuals, for example the `.resids` attribute that you can work with. Of course you could also use the regression estimates to compute both values yourself!



5.5.2 Model Selection

As a wrap-up, by considering the different specifications as well as your alternative model, write a few sentences about which specification/model you believe to be best-suited for the underlying data.

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 a statistical analysis in this section. If

you stuck around until this part and managed to code the better part of the exercises, you have definitely earned your certificate! We hope you had fun learning Data Science with this combination of data sets and enjoyed it – at least the parts where you did not get stuck forever because of some unexplainable coding error. Do not forget to send your project results to our project submission email address before the deadline (**03.07.2022, 23:59**). Thank you for being a part of TechAcademy!

6 Exercise Checklist

This checklist should help you keep 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 section 4 (EDA) only. If you're part of the advanced track, you have to hand in at least two-thirds of section 4 and 5. Hence, you'll need at least 66% in each of the two sections for a certificate.

6.1 Part 1: Exploratory Data Analysis (Beginner + Advanced Tracks)

You completed at least 6 courses on Datacamp.

You completed at least two-third of section 4.

You checked that you included everything you coded in the right order and added explanatory comments.

You have all your output in the desired format.

You compared your output with your group.

Finally, you can hand in your final work in time *(Remeber: Each team member has to hand in their final work individually, in any case it can be the same within the group)*

6.2 Part 2: Seasonality and Geographic Differences – Application of Statistical Methods (motivated Beginner + Advanced Tracks)

Everything of the above stated.

In addition you also completed at least two-third of section 5.

7 What's Next in Your Data Science Career?

7.1 Data Science in General

Version Control with Git

Advice for Non-Traditional Data Scientists

Learn from Great Data Scientists on Kaggle

Great insights and helpful tips on towardsdatascience

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 own computer. Here's a DataCamp tutorial on how to do that.

Version Control with Git
RStudio has a nice 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 extremely influential on 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 in this semester, especially the tidyverse (including great packages such as ggplot2 and dplyr). Follow him on Twitter to get great R advice and keep up to speed with everything new to R. Following the many people behind R (not only Hadley) is a great way for acquiring deeper understanding of the language and its developments.

Join the Campus useR Group in Frankfurt
There's a quite 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. A next step would be to install Python and Jupyter locally on your computer. This link contains the necessary information on how to install 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 and check out the Community Version of "PyCharm" which is the most common Python IDE (Integrated Development Environment.

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 next 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 which are used in the field of 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 might work for yourself, but often you will work on code with other people. But even if you're just 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.