coffee break pandas

Name	Profession
"Rieger"	"Author"
"Kravets"	"Scientist"
"Mayer"	"Teacher"

74 Pandas Puzzles to Build Your Data Science Superpower

Coffee Break Pandas

74 Pandas Puzzles to Build Your Pandas Data Science Superpower

Lukas Rieger, Kyrylo Kravets, and Christian Mayer

A puzzle a day to learn, code, and play.

Contents

— 1 —

Introduction

Data is the new oil—whoever develops the skills to harvest this largely untapped asset class will increase their value in the 21st-century marketplace.

Where does all the data come from? Computers and sensors create ever-growing data sets that become increasingly relevant in every part of our lives. Smartphones collect billions of GPS traces that reveal valuable information towards new mobility solutions and truly smart cities. Smartwatches track the heart rates of millions of people to detect health problems and prevent unnecessary deaths. Intelligent cars collect myriads of sample data points that increase traffic efficiency, prevent millions of car accidents, and save humanity hundreds of millions of hours in traffic jams. Data truly is a modern-world asset.

Data processing now plays a key part in organizing modern society. How did that happen? During the first years of the computer, hardware capacities were limited and expensive. Processing vast amounts of data was impossible during the first few decades. For-

tunately, Moore's Law—*the number of transistors on a microchip doubles roughly every two years*—lead to an exponential cost reduction for computing power. Hardware became more powerful and much cheaper—by orders of magnitudes. Today, it is financially viable to store, process, and analyze hundreds of terabytes of data. Big corporations such as Amazon and Facebook are avidly collecting data to get to know their users even better and increase their revenues.

On the other hand, the general public is discussing how to protect and restrict data usage. The discussion around how to set boundaries in data processing is relevant for every citizen—and especially for an ambitious programmer like you.

By reading this book, you'll learn the powerful skill of analyzing and processing data with Python's Pandas framework. However, please use your newly-acquired power wisely! *"With great power comes great responsibility."*

Since you are reading this book, you understand the importance of mastering Pandas. Yet, we'd like to present to you some facts to underline this:

1. Only a small fraction of the collected data is actually used for analysis or further processing.

2. There was an estimated need of 364 *thousands* new data scientists in the US in 2020 as a report by IBM.[1]

3. 50 billion connected smart devices collect and analyze data.

[1] `https://www.ibm.com/downloads/cas/3RL3VXGA`

4. The data science sector in India has prospects to grow to $16 billion until 2025.
5. More than 1 trillion pictures are taken in a single year, and billions of them are shared online.
6. Around 1.7 megabytes of new information is created every second for every person on the planet in 2020.
7. On Facebook's Open Graph, which connects websites with social media, 1 billion pieces of content are shared daily.
8. The average data scientist salary is $100,560, according to the US Bureau of Labor Statistics

What is Pandas anyway? To keep up with the latest technical developments, Wes McKinney started developing a high-performance flexible data analysis tool in 2008. Up to this date, Python was mainly used for data munging and preparation. Before you ask—data munging, also denoted as *data wrangling* refers to the process of data preparation so that a set of raw data can be used for dedicated purposes beyond its original intent. Typically, raw data can not be used directly by a data scientist or machine learning algorithm. For example, the raw data could be incomplete or have the wrong format. The process of data preparation can include several steps, such as cleaning, standardizing, and filtering.

The result of his effort was Pandas, a high-performance, open-source Python library. The name *Pandas* is derived from the word *"**pan**el **da**ta"*, a term for multidimensional data.

Pandas is not only used in data science but also in other fields such as:

1. **Advertising**: Machine learning and deep learning have enabled companies to understand their customers' needs and preferences. To achieve this, they use the functionality of the Pandas library.

2. **Economics**: Python and Pandas are useful business tools because analyzing huge data sets helps understand and predict changing behavioral patterns in the economy.

3. **Recommendation Systems**: It's surprising how accurately Amazon and Netflix predict which product to buy or which movie to watch next. The more data, the better the models' predictions.

4. **Neuroscience**: Over the last years, machine learning has helped unravel many mysteries of the human body and mind. Pandas played a key role due to its capabilities to manipulate huge amounts of data.

5. **Natural Language Processing**: The objective of NLP is understanding and deciphering the human language. For example, algorithms analyze product reviews and identify the customer's mood. Did the customer like the product or not? Is the customer satisfied or even enthusiastic? And once again, Pandas and other Python libraries like sklearn make a deeper insight possible.

Pandas is built around a two-dimensional data structure: the *DataFrame*. Think of a DataFrame as a table with columns and rows—a bit like an Excel sheet. In fact, Pandas is like Excel on steroids!

Pandas allows you to load data of different types (for example, CSV, TSV, Excel, SQL) into in-memory data objects like DataFrames. DataFrames are efficient and fast and offer customized indexing based on index labels, location, or slicing. You can add or delete columns and rows from DataFrames, and you can join DataFrames like SQL tables. In the puzzles, you'll learn about all of this and much more!

You can download the ebook PDF and related training material such as cheat sheets and video tutorials on our *Coffee Break Pandas* book page.

Coffee Break Pandas Book Page: `https://blog.finxter.com/coffee-break-pandas-book-page/`

Make sure to visit the page, especially if you feel like you may need to refresh your Pandas skills. On the book page, you'll also find a *10-minutes to Pandas video tutorial*.

What you'll learn: You're an aspiring coder and seek ways to advance your coding skills. Nurturing your ambition to learn will pay lifelong dividends to you and your family. Providing value through information technology, automation, and digitization will give you confidence and make you a respectable member of society. So, keeping your ambition to learn intact is a worthy goal to pursue.

If at some point, you feel frustrated and overwhelmed, think of this: Mastery comes from intense, structured training. The author Malcolm Gladwell formulated the famous rule of 10,000 hours that states that, you will reach mastery in any discipline after investing approximately 10,000 hours of intense training if you have average

talent. Bill Gates, the founder of Microsoft, reached mastery at a young age due to coding for more than 10,000 hours. He was committed and passionate about coding and worked long nights to develop his skills. He was anything but an overnight success.

Coffee Break Pandas is part of the popular Coffee Break Python series, which consists of the following books: *Coffee Break Python*[2], *Coffee Break Python Slicing*, *Coffee Break NumPy*[3], *Coffee Break Python Workbook*[4], *Coffee Break Python Mastery Workouts*[5].

This book aims to be a stepping stone on your path to becoming a Python master. It helps you to learn faster by using the principles of good teaching. It contains 20-25 hours of Pandas data science training using one of the most efficient learning techniques: *practice testing*. This technique is guaranteed to improve your ability to read, write, and understand Pandas source code.

The idea is that you solve code puzzles that start simple and become more and more complex as you read the book. In essence, you play the role of the Python interpreter and compute each code snippet's output in your head. Then you check whether you were right. Using the accompanying feedback and explanations, you will adapt and improve your coding skills over time. To make this idea a reality, we developed the online coding academy `Finxter.com`.

The next section explains the advantages of the Finxter method of puzzle-based learning. If you already know about the benefits of

[2] `https://blog.finxter.com/coffee-break-python/`

[3] `https://blog.finxter.com/coffee-break-numpy/`

[4] `https://blog.finxter.com/coffee-break-python-workbook/`

[5] `https://blog.finxter.com/coffee-break-python-mastery-workout-ebook/`

puzzle-based learning from previous books, and you want to dive right into the puzzles, please skip the following chapter and start at Chapter 4.

— 2 —

A Case for Puzzle-based Learning

> **Definition:** A *code puzzle* is an educational snippet of source code that teaches a single computer science concept by activating the learner's curiosity and involving them in the learning process.

Before diving into practical puzzle-solving, let's first study 10 reasons why puzzle-based learning accelerates your learning speed and improves retention of the learned material. There is robust evidence in psychological science for each of these reasons. Yet, none of the existing coding books lift code puzzles to being first-class citizens. Instead, they mostly focus on one-directional teaching. This book attempts to change that. In brief, the 10 reasons for puzzle-based learning are:

1. Overcome the Knowledge Gap (Section 2.1)
2. Embrace the Eureka Moment (Section 2.2)

3. Divide and Conquer (Section 2.3)
4. Improve From Immediate Feedback (Section 2.4)
5. Measure Your Skills (Section 2.5)
6. Individualized Learning (Section 2.6)
7. Small is Beautiful (Section 2.7)
8. Active Beats Passive Learning (Section 2.8)
9. Make Source Code a First-class Citizen (Section 2.9)
10. What You See is All There is (Section 2.10)

2.1 Overcome the Knowledge Gap

The great teacher Socrates delivered complex knowledge by asking a sequence of questions. Each question was built on answers to previous questions provided by the student. This teaching is more than 2400 years old and is still in widespread use today. A good teacher opens a gap between their and the learner's knowledge. The knowledge gap makes the learner realize that they do not know the answer to a burning question. This creates tension in the learner's mind. To close this gap, the learner waits for the missing piece of knowledge from the teacher. Better yet, the learner starts developing their own answers. The learner *craves knowledge*.

Code puzzles open an immediate knowledge gap. When you first look at the code, you do not understand the meaning of the puzzle.

The puzzle's semantics are hidden. But only you can transform the unsolved puzzle into a solved one.

The problem of many teachers is that they open a knowledge gap that is too large. The learner feels frustrated because they cannot cross this gap. *Rated* code puzzles solve this problem because, by design, they are not too great a challenge. You must stretch yourself to solve them, but you can do it if you go all-out. Continually feeling a small but non-trivial knowledge gap creates a healthy learning environment. Stretch your limits, overcome the knowledge gap, and become better—one puzzle at a time.

2.2 Embrace the Eureka Moment

Humans are unique because of their ability to learn. Fast and thorough learning has always increased our chances of survival. Thus, evolution created a brilliant biological reaction to reinforce learning in your body. Your brain is wired to seek new information; it's wired to keep processing data, to keep learning.

Did you ever feel the sudden burst of happiness after experiencing a eureka moment? Your brain releases endorphins the moment you close a knowledge gap. The instant gratification from learning is highly addictive, and this addiction makes you smarter. Solving puzzles gives your brain instant gratification. Easy puzzles lead to more challenging puzzles, which open large knowledge gaps. Each one you solve shortens the knowledge gap, and you learn in the process.

2.3 Divide and Conquer

Learning to code is a complex task. You must learn a myriad of new concepts and language features. Many aspiring coders are overwhelmed by this complexity. So they seek a clear path to mastery.

People tend to prioritize specific activities with clearly defined goals. If the path is not clear, we tend to drift away toward more specific paths. Most aspiring coders think they have a goal: becoming a better coder. Yet, this is not a specific goal at all. So what is a specific goal? *Watching Game of Thrones after dinner, Series 2 Episode 1* is as specific as it can be. The formulation "watching Netflix" is more specific than the fuzzy path of learning to code. Hence, watching Netflix wins most of the time.

As any productivity expert will tell you: break a big goal into a series of smaller steps. Finishing each tiny step brings you one step closer to your big goal. *Divide and conquer* makes you feel in control and takes you one step closer to mastery.

Code puzzles do this for you. They break up the huge task of learning to code into a series of smaller steps. You experience laser focus on one learning task at a time. Each puzzle is a step toward your bigger goal of mastering computer science. Keep solving puzzles, and you keep improving your skills.

2.4 Improve From Immediate Feedback

As a child, you learned to walk by trial and error—try, receive feedback, adapt, and repeat. Unconsciously, you will minimize

negative and maximize positive feedback. You avoid falling because it hurts, and you seek the approval of your parents. To learn anything, you need feedback so that you can adapt your actions.

However, an excellent learning environment provides you not just with feedback but with *immediate* feedback for your actions. In contrast, poor learning environments provide feedback only with a considerable delay, or not at all. If you were to slap your friend each time he lit a cigarette—a not overly drastic measure to save his life—he would quickly stop smoking. If you want to learn fast, make sure that your environment provides immediate feedback. Your brain will find rules and patterns to maximize the reinforcement from the immediate feedback. Puzzle-based learning with this book offers you an environment with immediate feedback. This makes learning to code easy and fast. Over time, your brain will absorb the meaning of a code snippet quicker and with higher precision. Learning this skill will take you to the top 10% of all coders. There are other environments with immediate feedback, like executing code and checking correctness, but puzzle-based learning is the most direct one: Each puzzle educates with immediate feedback.

2.5 Measure Your Skills

Think about an experienced Python programmer, you know. How good are their Python skills compared to yours? On a scale from your grandmother to Bill Gates, where is that person, and where are you? These questions are difficult to answer because there is no simple way to measure the skill level of a programmer. This creates a problem for your learning progress. The concept of being

a good programmer has become fuzzy and diluted. What you can't measure, you can't improve.

So what should be your measurable goal when learning to program? To answer this, let's travel briefly to the world of chess. This sport provides an excellent learning environment for aspiring players. Every player has an Elo rating number that measures their skill level. You get an Elo rating when playing against other players—if you win, your Elo rating increases. Victories against stronger players lead to a higher increase in the Elo rating. Every ambitious chess player simply focuses on one thing: increasing their Elo rating. The ones that manage to push their Elo rating very high earn grandmaster titles. They become respected among chess players and in the outside world.

Every chess player dreams of being a grandmaster. The goal is as measurable as it can be: reaching an Elo of 2400 and master level (see Section 3). Thus, chess is a great learning environment. Every player is always aware of their skill level. A player can measure how their decisions and habits impact their Elo number. Do they improve when sleeping enough before important games? When training opening variants? When solving chess puzzles? What you can measure, you can improve.

The main idea of this book and the associated learning app `Finxter.com` is to transfer this method of measuring skills from the chess world to programming. Suppose you want to learn Python. The Finxter website assigns you a rating that reflects your coding skills. Every Python puzzle has a rating number according to its difficulty level. You 'play' against a puzzle at your difficulty level. The puzzle and you will have a similar Elo rating so that your learning

is personalized. If you solve the puzzle, your Elo increases, and the puzzle's Elo decreases. Otherwise, your Elo decreases, and the puzzle's Elo increases. Hence, the Elo ratings of the difficult puzzles increase over time. But only learners with high Elo ratings will see them. This self-organizing system ensures that you are always challenged but not overwhelmed. Plus, you frequently receive feedback about how good your skills are in comparison to others. You always know exactly where you stand on your path to mastery.

2.6 Individualized Learning

Today, the education system is built around the idea of classes and courses. In these environments, all students consume the same learning material from the same teacher applying the same teaching methods. This traditional idea of classes and courses has a strong foundation in our culture and social thinking patterns.

Yet, science proves, again and again, the value of individualized learning. Individualized learning tailors the content, pace, style, and technology of teaching to the students' skills and interests. Of course, truly individualized learning has always required a lot of teachers. But paying a high number of teachers is expensive (at least in the short term) in a non-digital environment.

In the digital era, however, computer servers and intelligent machines provide individualized learning with ease. Puzzle-based learning is a perfect example of automated, individualized learning. The ideal puzzle stretches the student's abilities and is neither boring nor overwhelming. Finding the perfect learning material for

each learner is an important and challenging problem. Finxter uses a simple but effective solution to solve this problem: the Elo rating system. The student solves puzzles at their skill level. This book and the book's web back-end Finxter push teaching towards individualized learning.

2.7 Small is Beautiful

The 21st century has seen the rise of microcontent. Microcontent is a short and accessible piece of information such as the weather forecast, a news headline, or a cat video. Social media giants like Facebook and Twitter offer a stream of never-ending microcontent. Microcontent has many benefits: the consumer stays engaged, and it is easily digestible in a short amount of time. Each piece of microcontent pushes your knowledge horizon a bit further. Today, millions of people are addicted to microcontent - so make sure you get addicted to the right type of content.

However, this addiction will become a problem. The computer science professor, Cal Newport, shows in his book *Deep Work* that modern society values deep work more than shallow work. Deep work is a high-value activity that needs intense focus and skill. Examples of deep work are programming, writing, or researching. On the other hand, shallow work is a low-value activity that anybody can do, e.g., posting cat videos to social media. The demand for deep work has grown with the rise of the information society. At the same time, the supply has stayed constant or decreased. One reason for this is the addictiveness of shallow social media. People that see and understand this trend can benefit tremendously. In a free market, the prices of scarce and high-demand resources

rise. Because of this, surgeons, lawyers, and software developers earn $100,000+ per year. Their work cannot easily be replaced or outsourced to unskilled workers. If you can do deep work and focus your attention on a challenging problem, society pays you generously.

What if we could marry the concept of microcontent with deep work? This is the promise of puzzle-based learning. Finxter offers a stream of self-contained microcontent in the form of hundreds of small code puzzles. But instead of just being unrelated nonsense, each puzzle is a tiny stimulus that teaches a coding concept or language feature. Hence, each puzzle pushes your knowledge *in the same direction.*

Puzzle-based learning breaks the big goal of *reach mastery level in Python* into tiny actionable steps: solve and understand one code puzzle per day. A clear path to success.

2.8 Active Beats Passive Learning

Robust scientific evidence shows that active learning doubles students' learning performance. In a study on this matter, test scores of active learners improved by more than a grade compared to their passive learning counterparts.[1] Not using active learning techniques wastes your time and hinders you if you want to reach your full potential. Switching to active learning is a simple tweak that instantly improves your performance.

[1] `https://en.wikipedia.org/wiki/Active_learning#Research_evidence`

How does active learning work? Active learning requires the student to interact with the material rather than consume it. It is student-centric rather than teacher-centric. Great active learning techniques are asking and answering questions, self-testing, teaching, and summarizing. A popular study shows that one of the best learning techniques is *practice testing*.[2] In this technique, you test your knowledge before you have learned everything. Rather than *learning by doing*, it's *learning by testing*.

The study argues that students must feel safe during these tests. Therefore, the tests must be low-stakes, i.e., students have little to lose. After the test, students get feedback on how well they did. The study shows that practice testing boosts long-term retention by almost a factor of 10. So, solving a daily code puzzle is not just another learning technique—it is one of the best.

Although active learning is twice as effective, most books focus on passive learning. The author delivers information; the student passively consumes the information. Some programming books include active learning elements by adding tests or asking the reader to try out the code examples. Yet, I've always found this impractical when reading on the train, in the bus, or in bed. If these active elements drop out, learning becomes 100% passive again.

Fixing this mismatch between research and everyday practice drove me to write this book about puzzle-based learning. In contrast to other books, this book makes active learning a first-class citizen. Solving code puzzles is an inherent active learning technique. You must figure out the solution yourself for every single puzzle. The teacher is as much in the background as possible—they only ex-

[2] `http://journals.sagepub.com/doi/abs/10.1177/1529100612453266`

plain the correct solution if you couldn't work it out yourself. But before telling you the correct solution, your knowledge gap is already ripped wide open. Thus, you are mentally ready to digest new material.

2.9 Make Code a First-class Citizen

Each chess grandmaster has spent tens of thousands of hours looking at a nearly infinite number of chess positions. Over time, they develop a powerful skill: the intuition of the expert. When presented with a new position, they can name a small number of strong candidate moves within seconds. They operate on a higher level than normal people. For normal people, the position of a single chess piece is one chunk of information. Hence they can only memorize the position of about six chess pieces. But chess grandmasters view a whole position or a sequence of moves as a single chunk of information. Extensive training and experience have burned strong patterns into their biological neural networks. Their brains can hold much more information, resulting from the excellent learning environment they have put themselves in.

Chess exemplifies principles of good learning that are valid in any field you want to master.

First, transform the object you want to learn into a stimulus and then look at it repeatedly. In chess, study as many chess positions as you can. In math, read mathematical papers containing theorems and proofs. In coding, expose yourself to lots of code.

Second, seek feedback. Immediate feedback is better than delayed feedback. However, delayed feedback is still better than no feed-

back at all.

Third, take your time to learn and understand thoroughly. Although it is possible to learn on-the-go, you will cut corners. The person who prepares beforehand always has an edge. Some people recommend to work on practical coding projects and doing nothing more. Chess grandmasters, sports stars, and intelligent machines do not follow this advice. They learn by practicing small parts again and again until they have mastered them. Then they move on to more complicated bits.

Puzzle-based learning is code-centric. You will find yourself staring at the code for a long time until the insight strikes. This creates new synapses in your brain that help you understand, write, and read code fast. Placing code in the center of the learning process means you will develop an intuition as powerful as the experts. *Maximize the learning time you spend looking at code rather than other things.*

Grandmasters of chess see the best moves intuitively. Grandmasters of code see the output or theme of a puzzle immediately. They can look at code and see the potential problems, the main themes, the style, the semantics. And they do this fast. Making code a first-class citizen enables the development of the intuition of the expert.

2.10 What You See is All There is

My professor of theoretical computer science used to tell us that if we stare long enough at a proof, the meaning will transfer to our brains by osmosis. This fosters deep thinking, a state of mind

where learning is more productive. In my experience, his staring method works—but only if the proof contains everything you need to know to solve it. It must be self-contained.

A good code puzzle beyond the most basic level is self-contained. You can solve it by staring at it until your mind follows your eyes, i.e., your mind develops a solution from rational thinking. There is no need to look things up. If you are a great programmer, you will find the solution quickly. If not, it will take more time, but you will still find the solution—it is just more challenging.

My gold standard was to design each puzzle such that they are (mostly) self-contained. However, to ensure you learn new concepts, puzzles must introduce new syntactical language elements as well. Even if the syntax in a puzzle challenges you, develop your own solutions based on your imperfect knowledge. This probabilistic thinking opens the knowledge gap and prepares your brain to receive and digest the explained solution. After all, your goal is long-term retention of the material.

— 3 —

The Elo Rating for Python

Pick any sport you've always loved to play. How good are you compared to others? The Elo rating answers this question with surprising accuracy. It assigns a number to each player that represents their skill in the sport. The higher the Elo number, the better the player.

Table 3.1 shows the ranks for each Elo rating level. The table is an opportunity for you to estimate your Python skill level. In the following, I'll describe how you can use this book to test your Python skills.

3.1 How to Use This Book

This book contains 74 code puzzles and explanations to test and train your Pandas skills. The puzzles start from beginner-level and become gradually harder. In the end, you will be an intermediate-level coder. The Elo ranges from 1500 to 1900 points (between *intermediate* and *professional* level in the table). Follow-up books

Elo rating	**Rank**
2500	World Class
2400-2500	Grandmaster
2300-2400	International Master
2200-2300	Master
2100-2200	National Master
2000-2100	Master Candidate
1900-2000	Authority
1800-1900	Expert
1700-1800	Professional
1600-1700	Experienced Intermediate
1500-1600	Intermediate
1400-1500	Experienced Learner
1300-1400	Learner
1200-1300	Scholar
1100-1200	Autodidact
1000-1100	Beginner
0-1000	Basic Knowledge

Table 3.1: Elo ratings and skill levels.

cover more advanced levels. This book is perfect for you if you are between the beginner and intermediate levels. Yet, even experts will improve their speed of code understanding if they follow the outlined strategy.

3.2 How to Test and Train Your Skills?

I recommend solving at least one code puzzle every day, e.g., as you drink your morning coffee. Then spend the rest of your learning time on real projects that matter to you. The puzzles guarantee that your skills will improve over time, and the real project brings you results.

To test your Python skills, do the following:

1. Track your Elo rating as you read the book and solve the code puzzles. Write your current Elo rating in the book. Start with an initial rating of 1000 if you are a beginner, 1500 if you are an intermediate, and 2000 if you are an advanced Python programmer. Of course, if you already have an online rating on `finxter.com`, start with that. Two factors impact the final rating: how you select your initial rating and how good you perform (the latter being more important).

2. If your solution is correct, add the Elo points given with the puzzle. Otherwise, subtract the points from your current Elo number.

Solve the puzzles sequentially because they build upon each other. Advanced readers can solve them in the sequence they wish—the

Elo rating will work just as well.

Use the following training plan to develop a healthy learning habit with puzzle-based learning. Small but regular daily improvements will boost your skills more than you ever expected. Ride the power of compounding knowledge.

1. Choose or create a daily trigger, after which you'll solve code puzzles for 10 minutes. For example, solve puzzles during your coffee break, as you brush your teeth, or sit on the train to work, university, or school.

2. Scan over the puzzle and ask yourself: what is the unique idea of this puzzle?

3. Dive deeply into the code. Try to understand the purpose of each symbol, even if it seems trivial at first. Avoid being shallow and lazy. Instead, solve each puzzle thoroughly and take your time. It may seem counter-intuitive initially, but you must take your time and allow yourself to dig deep to learn faster. There is no shortcut.

4. Stay objective when evaluating your solution—we all tend to lie to ourselves.

5. Look up the solution and read the explanation with care. Do you understand every aspect of the code? Write any questions down that you have and look up the answers later. Or send them to me (`admin@finxter.com`). I will do everything I can to come up with a good explanation.

6. If your solution was 100% correct—including whitespaces, data types, and formatting of the output—you get the Elo points for this puzzle. Otherwise, your solution was wrong, and you should subtract Elo points. This rule is strict because code is either right or wrong.

As you follow this simple training plan, your ability to understand source code will improve quickly. This will have a huge impact on your career, income, and work satisfaction in the long term. You do not have to invest much time because the training plan requires only 10–20 minutes per day. But you must be persistent with your effort. If you get off track, get right back on the next day. When you run out of code puzzles, feel free to checkout `Finxter.com`. It has more than 400 hand-crafted code puzzles, and I regularly publish new ones.

The *Coffee Break Pandas* page with more extensive material, such as cheat sheets, video tutorials, PDF files, and pointers to more detailed tutorials, can be found at `https://blog.finxter.com/coffee-break-pandas-book-page/`.

— 4 —

Pandas Code Puzzles Elo 1500-1600

The first 24 Pandas puzzles are in the *intermediate level*. For a Pandas expert, they'll e relatively easy while a Pandas beginner may already struggle with the puzzles. Anyways, give all the puzzles your best effort and study the explanations carefully.

4.1 Create a Series From a Scalar Value

```
# Elo 1500
import pandas as pd

scalar_series = pd.Series(42, index=[0, 1, 2],
                          dtype=float, name='CONST')
result = scalar_series.sum()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`126.0`

Explanation
In this puzzle, you create a Series that contains three times the value `42.0`. You pass the scalar value `42` as data and a list of integers as index values. Since there are three index values, you obtain three entries in the Series. These entries are of data type `dtype=float`.

You can access the name of a Series through the attribute `name`—but this has no impact on the puzzle's result.

Series objects offer a range of helper methods. One such helper is the `sum()` method that adds up all values in the Series.

In the puzzle, the Series contains three time `42.0` so that the final result is `42.0 + 42.0 + 42.0 = 126.0`.

4.2 Create a Series From a List of Data

```
# Elo 1510
import pandas as pd

snakes = ['Python', 'Cobra', 'Anaconda', 'Viper']
snakes_series = pd.Series(snakes)
result = snakes_series.values[2]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`Anaconda`

Explanation
The columns of a DataFrame in Pandas consist of multiple Series—one Series per column. Each Series is a one-dimensional object that holds values. You can create a Series with different values by passing a list of values into the Series constructor. In this case, the index values are computed automatically. The index of the first element will be 0, the index of the second element will be 1, and so on. Using the `values` attribute of a Series, you get the values as an array. Subsequently, you can access the values with an index. This is not the Series index but the standard index of arrays. For example, if you retrieve the value at index 2, you'll get the third element, which is `'Anaconda'`.

4.3 Create Series with Custom Index Values

```
# Elo 1515
import pandas as pd

fibs = [0, 1, 1, 2, 3, 5, 8]
idxs = [0, 1, 2, 3, 4, 5, 6]

fib_series = pd.Series(data=fibs, index=idxs)
result = fib_series.shape

print(result)
```

What's the output of this code snippet?

Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`(7,)`

Explanation
In this puzzle, you can see two interesting facts about Series. First, you can create a Series with custom index values. In this case, the index values are the same as would have been created automatically. Second, Series objects have an attribute called `shape` that holds a tuple with one element. Series are one-dimensional objects, so the tuple's only element (=dimension) indicates the number of values in the Series. Since there are seven values in our Series `fibs`, the output is `(7,)`.

4.4 Create a Series From a Dictionary

```
# Elo 1520
import pandas as pd

salary = {'Alice':1000, 'Bob': 750, 'Carol': 1250}
salary_series = pd.Series(salary)
result = salary_series.index[1]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`Bob`

Explanation
This puzzle shows you how to create a Series from a dictionary. The dictionary's keys become the index labels, and the dictionary values become the values in the Series. The index labels of a Series can be accessed using the attribute `index` of a Series object.

In the puzzle, you retrieve the index label at index 1, which returns the second index label. Thus the output is `'Bob'`.

4.5 The Dimension of a Series

```
# Elo 1505
import pandas as pd

colors = pd.Series(['red', 'blue', 'green', 'brown'])
result = colors.ndim

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`1`

Explanation
You already learned how to get the shape of a Series object. Because Series are one-dimensional, the shape contains only one entry. You access the dimension by using the Series attribute `ndim` which results in the integer value 1 for any Series.

4.6 The Size of a Series

```
# Elo 1506
import pandas as pd

data = [1, 2, 3, 5, 7, 9, 11]
index = [2, 4, 6, 8, 10, 12, 14]

primes = pd.Series(data, index)
result = primes.size

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
7

Explanation
The `size` attribute of the Series object tracks its number of elements. Because there are seven values in our Series, the puzzle results in the integer `7`.

On a side note: You pass the `index` variable into the Series constructor to define a custom index with even numbers. However, this doesn't change the number of elements in the Series.

4.7 Checking Series for NaN-Values

```
# Elo 1507
import pandas as pd
import numpy as np

data = pd.Series([0, np.NaN, 2])
result = data.hasnans

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
Series can contain `NaN`-values—an abbreviation for *Not-A-Number*—that describe undefined values. To check if a Series contains one or more `NaN` value, use the attribute `hasnans`. The attribute returns `True` if there is at least one `NaN` value and `False` otherwise. There's a `NaN` value in the Series, so the output is `True`.

4.8 Integer Location Index

```
# Elo 1510
import pandas as pd

data = {
    'Spain':'Madrid',
    'France':'Paris',
    'Germany':'Berlin',
    'Italy':'Rome'
}
capitals = pd.Series(data)
result = capitals.iloc[-1]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`Rome`

Explanation
In this puzzle, you create a Series from a dictionary. The dictionary's keys are country names; the values are the respective capitals' names. To retrieve a value from a Series by its position in the Series, use the integer-location index `iloc[]`. With the index value `-1`, you access the last value in the Series. Thus the output is `Rome`.

Exercise: Create the same Series from two lists, one for the values and one for the indices. Go ahead and try it in your own shell!

4.9 Location Index

```
# Elo 1512
import pandas as pd

xs = pd.Series(range(7))
bs = [False, False, True, False, True, False, True]
result = xs.loc[bs].sum()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`12`

Explanation
To retrieve values from a Series, you use the *integer location index* `iloc[]` that accepts only integer values and throws a `TypeError` if you pass anything else to it. This puzzle demonstrates how the `loc[]` index works. In general, you can use it with row/column labels or Boolean indexing. The Series `xs` contains seven values: the numbers from `0` to `6`. The list `bs` consists of seven Boolean values. By passing the Boolean list `bs` to the `loc[]` index, you explicitly define which element you want to access from the `bs` Series. The i-th Boolean value defines whether the i-th value of the Series should be selected or not. The Boolean values at index `2`, `4` and `6` are `True`—therefore you obtain the values `2`, `4` and `6` from the Series. Because you call the method `sum()` directly on the result, the output is `2 + 4 + 6 = 12`.

4.10 Methods of Series: `all()` and `any()`

```
# Elo 1525
import pandas as pd

bools = pd.Series([True, False, True, True])
result = bools.all() | bools.any()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
The methods `all()` and `any()` check if *all* or *at least one* Boolean value in the Series object is `True`, respectively. In the puzzle, you use the *or* operator `|` on the output of the two method calls. As there are `True` values in the Series the call `bools.any()` returns `True`, so that the result of the or connection yields `True` as well.

4.11 The Methods `min()` and `max()`

```
# Elo 1526
import pandas as pd

xs = pd.Series(range(100))
result = xs.max() + min(xs)

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`99`

Explanation
To find the largest or smallest value in a Series of integer values you can call the methods `max()` or `min()` on the Series object. As an alternative, you can also use Python's built-in function `max()` and `min()` and pass the Series as an argument. Since the puzzle Series `xs` contains all values from `0` to `99`, the smallest value is `0` and the largest value is `99`.

4.12 Update Values in a Series

```
# Elo 1530
import pandas as pd

cities = {1111:'New York', 2222:'Marseille', 3333:'Warsaw'}
zips = pd.Series(data=cities)
zips = zips.apply(lambda x: x.upper())

result = zips.loc[2222]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`MARSEILLE`

Explanation
The Series method `apply()` applies a function to all values in the Series. You may think that it changes all values in the Series, depending on the function's output. However, the function does not change the Series elements in-place, but creates a new Series object. Therefore, we reassign the result to the variable `zips`. The index `loc[]` retrieves the element with key `2222`. Thus, the output is `MARSEILLE` in capital letters.

Exercise: Retrieve the same value using the index `iloc[]`!

4.13 Index Operators I

```
# Elo 1533
import pandas as pd

xs = pd.Series([5, 1, 4, 2, 3])
xs.sort_values(inplace=True)
result = xs[0]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`5`

Explanation
If you dive deeper into the Pandas library, you'll realize that many Pandas object methods allow an argument `inplace`. If you set it to `True`, Pandas applies the changes in-place, so that you don't need to reassign the result to a new variable or overwrite an existing one.

In fact, if you use the `inplace`, you cannot even reassign the result because the method call doesn't return anything. While this can improve readability and reduce the likelihood of bugs in your code, this also means that you can't do *method chaining*.

The puzzle demonstrates how to sort values in a Series. However, it's tricky because you have to understand the difference between retrieving a value directly with the index operator `[]` (or `loc[]`) and retrieving a value with `iloc[]`.

After sorting the values in the Series it looks like this:
1 1
3 2
4 3
2 4
0 5
The first column contains the indices, the second column the Series values. For each value, you keep the index to which it was initially assigned. By accessing the values via index (`[0]`), you obtain the value with the index `0`—which is `5`. To get the first element by index, you use `iloc[0]` which leads to the final result `1`.

4.14 Index Operators II

```
# Elo 1535
import pandas as pd

s = pd.Series(range(0, 100, 50))
t = s > 50
result = t.any()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
You can compare the whole Series to a value using standard comparison operators. The semantics of this *broadcasting operation* is simple: you compare each Series element to the scalar value. The result is a Series of Boolean values that represent the result of every single comparison. As there are only the two values 0 and 50 in our Series, all comparisons yield `False`, and the method `any()` therefore returns `False`.

4.15 Filtering Series

```
# 1550
import pandas as pd

xs = pd.Series([5, 1, 4, 2, 3])
xs.where(xs > 2, inplace=True)
result = xs.hasnans

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
The method `where()` filters a Series by a condition. Only the elements that satisfy the condition remain in the resulting Series. And what happens if a value doesn't satisfy the condition? Per default, all rows not satisfying the condition are filled with `NaN`-values. This is why our Series contains `NaN`-values after filtering it with the method `where()`.

4.16 Method Chaining

```
# Elo 1555
import pandas as pd

xs = pd.Series([2, 0, 1])
xs.sort_values().sort_index()

print(xs)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
0 2
1 0
2 1

Explanation
The Series method `sort_values()` sorts the Series by value. Because you don't set the argument `in-place = True`, the method returns the sorted Series, and you can call another method on the returned Series. This is denoted as *method chaining*. You call the method `sort_index()` on the value-sorted Series. The index values doesn't change as long as you don't explicitly tell it to do so. Sorting the Series by the index undoes the sorting by value. The output is a Series whose values are in the same order as in the original Series.

4.17 The `axis` Argument

```
# Elo 1540
import pandas as pd

s = pd.Series(range(10))
result = s.cumsum(axis=0)[4]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`10`

Explanation
A common argument of many functions in Pandas is the `axis` argument. However, as a Series is a one-dimensional object, you may wonder where you should use it. After all, Series only have a single axis. Yet, in order to be able to understand more advanced puzzles, you must understand how the `axis` argument works. A Pandas DataFrame is a table—a two-dimensional object. For a DataFrame, it does make sense to use the `axis` argument. For instance, you could either sum up the values of the first row or the first column.

As a rule of thumb: If you want to apply a method on a column (top to bottom), set `axis` to `0`. If you want to apply a method on a row (left to right), set `axis` to `1`. The default value of the parameter `axis` is `0`; thus, in this puzzle, you actually wouldn't need it.

The method `cumsum()` computes cumulative sums—the i-th sum is computed as the (i-1)-th sum plus the i-th element. The result of calling the method `cumsum()` is another Series. In the example, you have all values from `0` to `9` in a Series, so the resulting Series is this:
0 0
1 1
2 3
3 6
4 10
5 15

6 21
7 28
8 36
9 45
At index `4` the cumulative sum is `10`.

4.18 Working with Multiple Series

```
# Elo 1560
import pandas as pd

s = pd.Series(range(0, 10))
t = pd.Series(range(0, 20))
result = (s + t)[1]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
2

Explanation
To add two Series element-wise, use the default addition operator `+`. The Series do not need to have the same size because once the first Series ends, the subsequent element-wise results are `NaN` values. At index `1` in the resulting Series, you get the result of `1 + 1 = 2`.

4.19 Create a DataFrame from a List of Lists

```
# Elo 1580
import pandas as pd

data = [
    ['apple', 1.0],
    ['bread', 2.5],
    ['egg', 0.25]
]

df = pd.DataFrame(data, columns=['Item', 'Price'])
shopping_list = pd.Series([5, 1, 4])
result = (df['Price'] * shopping_list).sum()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`8.5`

Explanation
One way of creating a DataFrame in Python is using a list of lists. Each list represents a row of the DataFrame, and in the argument `columns`, you pass another list containing the column labels. You can access a column in a DataFrame with the syntax `df[column_name]`. A single column is represented as a Series. This is why you can multiply the Series `shopping_list` with the column `Price`. The multiplication result is another Series. You can sum up all the results by calling the method `sum()` directly. The total price for the shopping bag is `8.5`.

4.20 Create a DataFrame from a Dictionary of Lists

```
# Elo 1582
import pandas as pd

items = ['apple', 'bread', 'egg']
prices = [1.0, 2.5, 0.25]
data = {'Item':items, 'Price':prices}

df = pd.DataFrame(data)
df['Calories'] = [100, 2500, 25]

print(df)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer

The puzzle prints the following result:

```
    Item  Price  Calories
0  apple   1.00       100
1  bread   2.50      2500
2    egg   0.25        25
```

Explanation

Another way to create a DataFrame in Pandas is with a dictionary. The keys are the column labels, and the values are the column entries.

In this puzzle, you add a new column to the existing DataFrame `df`, select the new column, assign a list of values to it, and add the new column to the DataFrame.

4.21 List of Dict to DataFrame

```
# Elo 1584
import pandas as pd

data = [{'Car':'Mercedes', 'Driver':'Hamilton, Lewis'},
        {'Car':'Ferrari', 'Driver':'Schumacher, Michael'},
        {'Car':'Lamborghini'}]

df = pd.DataFrame(data, index=['Rank 2', 'Rank 1', 'Rank 3'])
df.sort_index(inplace=True)
result = df['Car'].iloc[0]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`Ferrari`

Explanation
You create a DataFrame from a list of dictionaries. The dictionaries' keys define the column labels, and the values define the columns' entries. Not all dictionaries must contain the same keys. If a dictionary doesn't contain a particular key, this will be interpreted as a `NaN`-value.

This puzzle uses string labels as index values to sort the DataFrame. After sorting, the row with index label `Rank 1` is at location `0` in the DataFrame and the value in the column `Car` is `Ferrari`.

4.22 Create a DataFrame from a List of Tuples

```
# Elo 1586
import pandas as pd

names = ['Alice', 'Bob', 'Clarisse', 'Dagobert']
ages = [20, 53, 42, 23]
data = list(zip(names, ages))

df = pd.DataFrame(data, index=['A', 'B', 'C', 'D'],
↪  columns=['Name', 'Age'])
result = df.loc['C', 'Age']

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
42

Explanation
In this puzzle, you create the DataFrame from a list of tuples and use string labels for the index. With the index `loc[]`, you can use these labels to access values. Since a DataFrame is a two-dimensional object, you need a row and column index. In row `C`, column `Age`, you find the age of Clarisse to be `42`.

4.23 Create a DataFrame from Series

```
# 1598
import pandas as pd

data = {}
idx = range(1, 11)
for i in range(1, 11):
    column = pd.Series(range(i, i * 10 + 1, i), index=idx)
    data[i] = column

mult_table = pd.DataFrame(data, index=range(1, 11))
result = mult_table.loc[4, 5]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
20

Explanation
Another way to create DataFrames is from a dictionary where the keys are the column labels, and the values are Series representing the columns. In this puzzle, you create a multiplication table that contains the result of `i*j` at `loc[i, j]`.

Thus, the result is `4 * 5 = 20`.

4.24 Create a DataFrame from a CSV File

```
# Elo 1580
import pandas as pd

# Contents of cars.csv
# -----------------------
# make,fuel,aspiration,body-style,price,engine-size
# audi,gas,turbo,sedan,30000,2.0
# dodge,gas,std,sedan,17000,1.8
# mazda,diesel,std,sedan,17000,NaN
# porsche,gas,turbo,convertible,120000,6.0
# volvo,diesel,std,sedan,25000,2.0

cars = pd.read_csv('cars.csv')
result = cars['make'].values[-1]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`volvo`

Explanation
Pandas simplifies the creation of DataFrames from CSV files. To do so, use the function `read_csv()` from the Pandas module. The final value in the column `make`, and output of the code puzzle, is `volvo`. The function `read_csv()` is very common in real-world projects. Therefore, you'll use it to create DataFrames in subsequent puzzles.

— 5 —

Pandas Code Puzzles Elo 1600-1700

The next 33 Pandas puzzles are in the *experienced intermediate level*. You'll find them slightly more difficult than the previous batch. After studying this chapter, you've learned many advanced concepts such as replacing values, renaming columns, dropping values, handling missing data, Boolean indices, aggregation, concatenation, and various join operations.

5.1 DataFrame Head

```
# Elo 1600
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
```

```
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df.head(2)
result = selection.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 1]`

Explanation
The DataFrame's `head()` method returns the first `n` rows. Per default, `n` is set to `5`. Pass any other integer value to get the desired number of rows from the DataFrame.

5.2 DataFrame Tail

```
# Elo 1600
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df.tail(2)
result = selection.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[3, 4]`

Explanation
The method `tail()` of a DataFrame returns the last `n` rows. Per default, `n` is set to `5`. Pass any other integer value to get the desired number of rows, starting from the last row, from the DataFrame.

5.3 Slices of DataFrames I

```
# Elo 1600
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df.iloc[0:3]
result = selection.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 1, 2]`

Explanation
The index `iloc[]` of Pandas DataFrames is integer-location based. It expects either an index value from the range `0` to `len(x)-1` or a list of Booleans: to select the i-th row, you set `values[i]` to `True`. The index `iloc[]` also accepts slices (`start_index:stop_index`) and returns the rows from `start_index` to `stop_index - 1`.

5.4 Slices of DataFrames II

```
# Elo 1605
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df.loc[0:3]
result = result.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 1, 2, 3]`

Explanation
The index `loc[]` of a DataFrame is label-based, but you can also use it with a list of Boolean values. Note that `[0:3]` refers to the index labels, and not to the integer positions of the rows.

5.5 2D Slicing

```
# Elo 1610
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

result = df.loc[0:2, "price"].max()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
30000

Explanation
If you use a `loc` index on a DataFrame, it returns a subsets of data. You can then apply other methods on it. In the puzzle, you want to find the most expensive car among the first three cars. To accomplish this, you select the first three rows and the column `price` and apply the function `max()` to it. The return value is the most expensive car.

5.6 DataFrame vs. Series

```
# Elo 1610
import pandas as pd

dict_ = {1: ["a"], 2: ["b"], 3: ["c"]}

result1 = pd.Series(dict_)
result2 = pd.DataFrame(dict_)

print(len(result1) == len(result2))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
Creating a Series from a dictionary results in a single column, whereas creating a DataFrame from a dictionary results in three columns. Why? Have a look at the both data structures in the Python shell:

```
>>> result1
1    [a]
2    [b]
3    [c]
dtype: object
>>> result2
   1  2  3
0  a  b  c
```

The data is structured differently—the DataFrame takes the dictionary keys as column indices. Here's the result of applying the `len()` function on top of both:

```
>>> len(result1)
3
>>> len(result2)
1
```

Thus, the number of rows of the Series and the DataFrame are different.

5.7 Modifying Column Labels

```
# Elo 1625
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

df = df.add_prefix("tbl_")
df = df.add_suffix("_1")
print(df.columns[1])
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`tbl_fuel_1`

Explanation
The method `add_prefix()` adds a prefix and the method `add_suffix()` adds a suffix to all column labels of the DataFrame. Since we added `tbl_` as prefix and `_1` to all columns, the label of the second column (`index = 1`) is `tbl_fuel_1`.

5.8 Sorting a DataFrame by Column

```
# Elo 1625
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

selection = df.sort_values(by="engine-size")
result = selection.index.to_list()[0]
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
1

Explanation
In this puzzle, you sort the rows of the DataFrame by the values of the column `engine-size`. The main point is that `NaN` values are always moved to the end in Pandas sorting. Thus, the first value is `1.8`, which belongs to the row with index value `1`.

5.9 Replacing Values

```
# Elo 1625
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

other_df = df.copy()

df["aspiration"].replace("std", "standard", inplace=True)
list1 = df["aspiration"].to_list()

other_df.replace("std", "standard", inplace=True)
list2 = other_df["aspiration"].to_list()
```

```
result = list1 == list2
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
You can either update values on a single column or on the DataFrame as a whole. To update values only in a column, select the column first and then call the method `replace()` on the column. To update a value in the whole DataFrame, call `replace()` directly on the DataFrame.

In the puzzle, updating the value `std` only in the column `aspiration` or in the whole DataFrame doesn't make a difference since the value `std` only occurs in this column.

5.10 Renaming Columns

```
# Elo 1625
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

second_df = df.copy()

df.rename(columns={"aspiration": "body-style",
```

```
                    "body-style": "aspiration"},
          inplace=True)

second_df.rename(columns={"aspiration": "body-style"},
                 inplace=True)
second_df.rename(columns={"body-style": "aspiration"},
                 inplace=True)

list1 = df.columns.to_list()
list2 = second_df.columns.to_list()
result = list1 == list2

print (result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
You can change a DataFrame's column labels by using the method `rename(columns={"old_name": "new_name"})` on the DataFrame. In the puzzle, you swap the column labels `aspiration` and `body-style` in the DataFrame `df` by passing both column names in the call of the method `rename()`.

After creating a copy of the DataFrame `df`, the DataFrame `second_df` points to a different object than the variable `df`. When renaming the first column `aspiration` to `body-style` the DataFrame contains two columns with the same label `body-style`. Thus, it renames both columns and not only the one that was called `body-style` initially. Ultimately, the lists of column labels are different and the comparison of both yields `False`.

5.11 Column Datatypes

```
# Elo 1630
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
```

```
# ----------

print(len(df.select_dtypes(include="number").columns))
print(len(df.select_dtypes(include="object").columns))
print(len(df.select_dtypes(
          include=["datetime", "timedelta"]).columns))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
2 4 0

Explanation
This example demonstrates how to audit data types of columns in a DataFrame. To display all columns of a specific type, use the method `df.select_dtypes()` with the parameter `include` that is the string or list of strings of the type(s) you want to select.

- For numerical columns: `include = "number"`,
- For string columns: `include = "object"`,
- For datetime columns: `include = ["datetime", "timedelta"]`.

Furthermore, you can specify the exact column type you are looking for, e.g. `include = ["int8", "int16", "int32", "int64", "float"]`.

5.12 Integer Location Index: Multiple Values

```
# Elo 1635
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
```

```
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df.sort_values(by="price", ascending=False, inplace=True)
result = df.iloc[2, 4]

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
25000

Explanation
You sort the DataFrame by the column `price` in descending order. Because `inplace` is set to `True`, the sorting happens on the DataFrame instance itself. Using the DataFrame's `iloc[]` index, you retrieve the value from the cell in the third row and fifth column.

5.13 Count Non-NaN Values

```
# Elo 1635
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df.count()[5]
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`4`

Explanation
The method `count()` returns the number of non-`NaN` values for each column. The DataFrame `df` has five rows. The fifth column contains one `NaN` value. Therefore, the count of the fifth column is `4`.

5.14 Drop NaN-Values

```
# Elo 1640
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection1 = df.dropna(subset=["price"])
selection2 = df.dropna()
print(len(selection1), len(selection2))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`5 4`

Explanation
The DataFrame's `dropna()` method drops all rows with at least one `NaN` value. In the puzzle, this drops line `2` because of the `NaN` value in the column `engine-size`.

By passing a list of column labels to the optional parameter `subset`, you can define which columns you want to be scanned for `Nan` values. When you restrict the columns only to `price`, no rows will be dropped, because no `NaN` value is present.

5.15 Adding Columns I

```
# Elo 1640
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df['origin'] = ["Germany", "USA", "Japan", "Germany", "Sweden"]
result = df[df["make"] == "porsche"]["origin"].values[0]
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`Germany`

Explanation
You can add a new column to a DataFrame: use indexing with the new column name and assign a list of values to it: `df["Column_Name"] = [value1, ..., valueN]`.

In this puzzle, you add the column `origin` to the DataFrame. Then, you select all cars where column `make` is set to the string `'porsche'`.

After that, you take column `origin` from the subset of rows and return the first value. Since there is only one row where `make` equals `'porsche'` the first value of the column `origin` is also the only one: `'Germany'`.

5.16 Adding Columns II

```
# Elo 1645
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------
```

```
df.loc[df.index.max() + 1] = ["ford", "gas", "std", "sedan",
↪  10000, 2.0]
df.index[df["price"] <= 17000].to_list()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[1, 2, 5]`

Explanation
You can add a new row to a DataFrame by assigning a list of values to the maximal index plus `1`. In the puzzle, you select all cars with a price less or equal than `17000` and return a list of their indices. Two cars cost exactly `17000` and the `ford` you added costs `10000`. Thus, there are three results.

5.17 Boolean Indexing I

```
# Elo 1650
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df[df["aspiration"] == "turbo"]["body-style"].values
result = list(selection)
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`['sedan', 'convertible']`

Explanation
In this powerful one-liner, you first select the column with the label `'aspiration'` with `df['aspiration']`.

Second, you compare this column's values to the value `'turbo'`, which results in a list of Boolean values.

Third, with this list, you select all lines where `aspiration` is set to `'turbo'`. This is also called *Boolean indexing*.

Fourth, after selecting theses lines, you only select the column `body-style` and retrieve the cell entries as `numpy.ndarray`.

Fifth, you convert the `numpy.ndarray` to a standard Python list.

5.18 Drop NaN-Values II

```
# Elo 1650
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------
```

```
df.drop([0, 1, 2], inplace=True)
df.reset_index(inplace=True)
result = df.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 1]`

Explanation
The method `drop()` on a DataFrame deletes rows or columns by index. You can either pass a single value or a list of values.

By default the `inplace` parameter is set to `False`, so that modifications won't affect the initial DataFrame object. Instead, the method returns a modified copy of the DataFrame. In the puzzle, you set `inplace` to `True`, so the deletions are performed directly on the DataFrame.

After deleting the first three rows, the first two index labels are `3` and `4`. You can reset the default indexing by calling the method `reset_index()` on the DataFrame, so that the index starts at `0` again. As there are only two rows left in the DataFrame, the result is `[0, 1]`.

5.19 Drop Columns

```
# Elo 1650
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
```

```
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df.drop("fuel", axis=1, inplace=True)
result = df.columns[1]
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`aspiration`

Explanation
The DataFrame method `drop()` either removes rows or columns (per default: rows with `axis=0`). To delete columns, specify the parameter `axis=1`. As the drop was performed in-place, the second column (`index = 1`) of the DataFrame is `aspiration`.

5.20 Drop Selected Values

```
# Elo 1655
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

drop_idx = df[df["fuel"] == "gas"].index
df.drop(drop_idx, inplace=True)
result = df.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[2, 4]`

Explanation
Using Boolean indexing, you select all rows from the DataFrame, where the value in the column `fuel` is set to `gas`. The `index` property returns all indices of the selected rows. Thus, the DataFrame's method `drop()` deletes the rows from the DataFrame in-place.

5.21 Sort In-Place

```
# Elo 1660
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

df.sort_values(by="price")
first = df.index.to_list()
df.sort_values(by="price", inplace=True)
second = df.index.to_list()
print(first == second)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
By default the method `sort_values()` method of the DataFrame does not work in-place but returns a copy of the DataFrame where the rows are ordered by the given column label. The initial DataFrame is unchanged.

To modify the initial DataFrame, set the parameter `inplace` to `True`. Doing so causes the method `sort_values()` to return `None`. In the puzzle, the results of the calls of the method `sort_values()`, with and without `inplace` set to `True`, are not assigned to a variable. Therefore, the variable `first` contains the index labels in the initial order whereas the variable `second` contains the index labels sorted by the column `price`. Thus, the comparison yields `False`.

5.22 Reverse Column Order

```
# Elo 1660
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
```

```
# ----------

a = df.columns.to_list()[0]
df = df.loc[:, ::-1]
b = df.columns.to_list()[0]

print(a == b)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
You can reverse the column order of a DataFrame with `df.loc[:, ::-1]`. The variable `a` contains the label `make` whereas the variable `b` contains the label `engine-size`. Thus, the comparison yields `False`.

5.23 Reverse Row Order

```
# Elo 1660
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

df = df.loc[::-1]
i = df.index.to_list()[0]
print(i)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
4

Explanation
You can reverse the order of a DataFrame with `df.loc[::-1]`. Therefore, the index of the first row is 4.

5.24 Reset the Index

```
# Elo 1665
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

a = df.index.to_list()[0]
df = df.loc[::-1].reset_index(drop = True)
b = df.index.to_list()[0]
print(a == b)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
In this puzzle, you reverse the row order and reset the index. The index labels before and after reversing the row order are equal.

5.25 Reference Confusion

```
# Elo 1665
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

second_df = df
third_df = df.copy()
df.drop("fuel", axis=1, inplace=True)
result = second_df.columns[1] == third_df.columns[1]
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
The variables `df` and `second_df` both refer to the same object in memory. In such cases all changes applied to the DataFrame `df` will be applied to `second_df` as well. The method `drop()` on DataFrame `df` modifies the same DataFrame object as `second_df`.

The variable `third_df` points to another memory address because of the DataFrame `copy` method. As the DataFrame to which `third_df` points remains unchanged, the result of the comparison is `False`.

5.26 Select Values From a List

```
# Elo 1665
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

list1 = ["dodge", "audi", "volvo"]

total1 = df[df["make"].isin(list1)]["price"].sum()
```

```
total2 = df[~df["make"].isin(list1)]["price"].sum()

print(total2 - total1)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
65000

Explanation
A DataFrame's rows can be accessed by passing a list of Boolean values as an index. This *Boolean indexing* operation returns the row at index `i` only if `list_of_bools[i] == True`. In this puzzle, you sum up the prices of a subset of cars whose names are given in a list. To achieve this, you use the method `isin()` on the selected column, which returns a list of Boolean values. The method `isin()` can be negated with the operator `~`, so it becomes a is-not-in check.

You then use the resulting list of Boolean values as Boolean index to retrieve the rows for the given names. In particular, you select the column `price` from the filtered rows and sum up the prices.

In the final step, you compute the difference between the sums of prices of the cars not in the list and the ones that are.

5.27 Boolean Indexing II

```
# Elo 1665
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
```

```
# 3  porsche     gas       turbo  convertible  120000          6.0
# 4    volvo  diesel         std        sedan   25000          2.0
# ----------

result = df[(df["price"] > 20000) | (df["price"] < 15000)]
result = result.index.to_list()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 3, 4]`

Explanation
In this puzzle, you select all rows where the `price` column is greater than `20000` or smaller than `15000`. You use two conditions—in parenthesis—and connect them with the "or" operator `|`. Note that to connect conditions with "and", use the operator `&`.

5.28 Aggregation

```
# Elo 1670
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

df2 = df.groupby("fuel").agg(list)
result = df2.index.to_list()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`['diesel', 'gas']`

Explanation
You can easily group a Dataframe's values by a column. DataFrames offer a method `group()` which expects a column label and returns a `DataFrameGroupBy` object. With the `DataFrameGroupBy` object's method `agg()`, you specify how to aggregate the values from the other columns in the resulting DataFrame. In this new DataFrame, the index labels are the unique values from the column by which the data was grouped. The values from the other columns of all rows in the group appear in a list.

5.29 DataFrame Concatenation I

```
# Elo 1675
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
```

```
#      make    fuel aspiration body-style  price  engine-size
# 0   skoda  diesel      turbo      sedan  18500          1.8
# 1  toyota     gas        std      sedan  22000          2.0
# 2    ford  diesel        std        suv  29000          3.0
# ----------

df3 = pd.concat([df, df2], axis=0)
result = sum(df3.index)

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`13`

Explanation
In this puzzle, you concatenate two DataFrames along the rows (`axis=0`). The key takeaway of this puzzle is that the concatenation of two DataFrames doesn't update the index labels of the new DataFrame. Therefore, the indices of DataFrame `df3` are `0, 1, 2, 3, 4, 0, 1, 2` and the sum is `13`. If you want to have a clean, sequential index, pass `ignore_index=True` when calling the method `concat()`.

5.30 DataFrame Concatenation II

```
# Elo 1680
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

# Additional Dataframe "df2"
# ----------
#      make   origin
```

```
# 0   skoda  Czechia
# 1  toyota    Japan
# 2    ford      USA
# ----------

try:
    result = pd.concat([df, df2], axis=0, ignore_index=True)
    print("Y")
except Exception:
    print ("N")

\textbf{Answer}\\
\texttt{Y}
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Explanation
Even if DataFrames have different columns, you can concatenate them. If DataFrame 1 has columns A and B and DataFrame 2 has columns C and D, the result of concatenating DataFrames 1 and 2 is a DataFrame with columns A, B, C, and D. Missing values in the rows are filled with `NaN`.

5.31 Inner Merge

```
# Elo 1685
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
#     make   origin
# 0  skoda  Czechia
# 1  mazda    Japan
# 2   ford      USA
# ----------
result = pd.merge(df, df2, how="inner", left_on="make",
↪  right_on="make")
```

```
print(result.iloc[0, 0])
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`mazda`

Explanation
Joining DataFrames is as easy as in SQL: The function `merge()` creates a new DataFrame from two input DataFrames. In the puzzle, you set the parameter `how` to `inner`—this is the inner join in SQL. The resulting DataFrame consists of the joined rows with values from both input DataFrames.

To clarify the join operation, let's have a look at the puzzle example. You join on the make column, so the function `merge()` takes the first entry from the column `make` which is `'audi'` and searches for it in the column `make` in `df2`. It doesn't find it, so no row will be added to the resulting DataFrame. Only for make `'mazda'` there is a corresponding value in `df2`. Thus, the resulting DataFrame has only one row that contains all values for `'mazda'`.

With `iloc[0, 0]`, you access the first column of the first row which is the `make` column in our example. Therefore, the output is `mazda`.

5.32 Right Merge

```
# Elo 1690
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
```

```
# 0     audi     gas       turbo        sedan   30000          2.0
# 1    dodge     gas         std        sedan   17000          1.8
# 2    mazda  diesel         std        sedan   17000          NaN
# 3  porsche     gas       turbo  convertible  120000          6.0
# 4    volvo  diesel         std        sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
#     make   origin
# 0  skoda  Czechia
# 1  mazda    Japan
# 2   ford      USA
# ----------

result = pd.merge(df, df2, how="right", left_on="make",
                  right_on="make")
print(len(result))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
3

Explanation
This puzzle shows a right-join to join DataFrames `df` and `df2`. In a right-join, the right DataFrame is the leading one. Thus, all rows present in DataFrame `df2` will also be contained in the result DataFrame. All rows values from DataFrame `df2` that don't have a corresponding row in DataFrame `df` are filled with `NaN`. Because DataFrame `df2` has three rows, the resulting DataFrame also has three rows.

5.33 Outer Merge

```
# Elo 1695
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
#     make   origin
```

```
# 0  skoda  Czechia
# 1  mazda    Japan
# 2   ford      USA
# ----------

result = pd.merge(df, df2, how="outer", left_on="make",
                  right_on="make")
print(len(result["fuel"]))
print(result["fuel"].count())
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer

```
7 5
```

Explanation

With Panda's function `merge()` and the parameter `how` set to `outer`, you can perform an outer join.

The resulting DataFrame of an outer join contains all values from both input DataFrames; missing values are filled with `NaN`.

In addition, this puzzle shows how `NaN` values are counted by the `len()` function whereas the method `count()` does not include `NaN` values.

Because the values `'skoda'` and `'ford'` only appear in the column `make` of the second input DataFrame, the DataFrame result contains two `NaN` values in the column `fuel`.

— 6 —

Pandas Code Puzzles Elo 1700-1800

You'll find that the next 15 Pandas puzzles are in the *expert level*. Again, the general difficulty has improved—and compared to the first batch of puzzles, you may find them really challenging. But fear not—by repeatedly opening and closing your knowledge gap, you'll make quick progress towards a more skilled use of the powerful Pandas library and your data science skills will soar as a result. So, let's get started!

6.1 Fun With NaN

```
# Elo 1700
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
```

```
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

for label in df.columns.values:
    s = df[label].isna().sum()
    if s > 0:
        print(label, s)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`engine-size 1`

Explanation
The code snippet prints the column label for each column with one or more occurrences of `NaN`, and how many times `NaN` occurs.

To access all column labels, use `df.columns.values`. To get the entire column, use `df[label]` (type: Series) with the given label.

You use the method `isna()` to get all the cells that contain `NaN` and by chaining it with the method `sum()`, you compute the count of `NaN` columns.

The if-statement becomes `True` only if the column contains one or more occurrences of `NaN`. Thus, the code prints `engine-size 1`.

6.2 DataFrame Information

```
# Elo 1700
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

print(df.info())
```

```
# ANSWER - OPTIONS
# a) count, max, std, min etc. values
# b) column, number of non-NA values, data types, memory usage
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
b)

Explanation
The method `info()` prints information about the DataFrame—including the index data type and column data types, non-null values and memory usage.

6.3 DataFrame Statistics

```
# Elo 1700
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

print(df.describe())

# ANSWER - OPTIONS
# a) count, max, std, min etc. values
# b) column, number of non-NA values, data types, memory usage
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
a)

Explanation
This method `describe()` generates descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding `NaN` values. It analyzes numeric and object series, as well as DataFrame column sets of mixed data types.

6.4 DataFrame Memory Usage

```
# Elo 1700
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

result = df.memory_usage(deep=True)
print(result)

# ANSWER OPTIONS
# a) Total memory used by DataFrame
# b) Memory used for each column.
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
b)

Explanation
The method `memory_usage()` of Pandas DataFrames returns the memory usage in bytes for each column. It's useful for optimizing the memory footprint of machine learning models that may need a lot of memory.

6.5 Numpy Arrays

```
# Elo 1705
import pandas as pd
import numpy as np

a = pd.np.array([0, 1, 2, 3])
b = np.array([0, 1, 2, 3])
c = a == b
print(c.all())
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`True`

Explanation
Pandas is based on the NumPy library. Thus, you can access all NumPy functionality from Pandas.

The comparison of the two arrays yields a list of Boolean values where the i-th value is the result of `a[i] == b[i]`.

The method `all()` of NumPy's `ndarray` returns `True`, only if all values in the array are `True`. As the elements from both arrays are equal, the `all()` method also returns `True`.

6.6 Regexing Column Labels

```
# Elo 1705
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df = df.filter(regex="-")
count = len(df.columns)

print(count)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
2

Explanation
With the method `filter()` on a DataFrame you can filter the columns. In the puzzle, the regular expression argument is applied to the column labels. Since two columns contain a dash in their column label, the output is 2.

6.7 Replacing NaN-Values

```
# Elo 1715
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

df.fillna(2.0, inplace=True)
result = df["engine-size"].sum()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`13.8`

Explanation
The method `fillna()` replaces `NaN` values with a new value. Thus, the sum of all values in the column `engine-size` is `13.8`.

6.8 Dummy Values

```
# Elo 1720
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

temp = pd.get_dummies(df["aspiration"])
temp.rename(columns={"std": "aspiration-std", "turbo":
↪  "aspiration-turbo"}, inplace=True)
df = pd.concat([df, temp], axis=1)
df.drop("aspiration", axis=1, inplace=True)

print(df["aspiration-std"].sum(), df["aspiration-turbo"].sum())
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`3 2`

Explanation
The function `get_dummies(column)` returns a DataFrame with one column for each value from the input column. The values in the new DataFrame are only `1` and `0` where `1` means that the value was contained, and `0` means that the value was not contained.

In the puzzle, you pass the column (type: Series) into the function `get_dummies()`. As there are two different values in the column (`turbo` and `std`), the output DataFrame contains two columns, one for `turbo` and one for `std`. Here is how it looks:

```
# Dataframe "temp"
# ----------
#       std turbo
# 0       0     1
# 1       1     0
# 2       1     0
# 3       0     1
# 4       1     0
# ----------
```

You rename the column labels to `aspiration-std` and `aspiration-turbo` and concatenate both DataFrames. On the columns `aspiration-std` and `aspiration-turbo`, you compute the sum to add up all the `1` values.

6.9 Method Chaining II

```
# Elo 1725
import pandas as pd
```

```
df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

result = df["engine-size"].fillna(1.0).apply(lambda x: x+1).sum()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`17.8`

Explanation
First, you fill the `NaN` values with `1.0`. Second, you use the method `apply()` to increase each value by `1.0` and sum up all the values from the column `engine-size`.

6.10 Length vs. Count

```
# Elo 1735
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
#     make   origin
# 0  skoda  Czechia
# 1  mazda    Japan
# 2   ford      USA
# ----------
```

```
result = pd.merge(df2, df, how="left", left_on="make",
                  right_on="make")
print(len(result["fuel"]))
print(result["fuel"].count())
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
3 1

Explanation
A left join includes all data values from the left DataFrame in the resulting DataFrame. The resulting DataFrame contains three rows for `'skoda'`, `'mazda'`, and `'ford'`. The remaining columns from the right DataFrame `df` that cannot be filled with meaningful data, such as `fuel`, receive `NaN` values.

But as the string values `'skoda'` and `'ford'` don't appear in the DataFrame `df`, only the row for `'mazda'` contains a non-`NaN` value.

The function `len()` also includes `NaN` values, while the method `count()` does not count `NaN` values.

6.11 Modifying Values

```
# 1745
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df["aspiration"] = df["aspiration"].map({"std": "Standard",
```

```
                                                "turbo": "Turbo"})
print(set(df["aspiration"]))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`'Turbo', 'Standard'`

Explanation
In this puzzle, you replace the column values of the column `aspiration`. To this end, you select the column `aspiration` and call its method `map()`. You define the mappings in a dictionary. However, you can also pass a function that computes the mappings. After the mapping was applied, the values in the column `aspiration` are `'Turbo'` and `'Standard'`.

6.12 Value Clusters

```
# Elo 1750
import pandas as pd
import numpy as np

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo         sedan   30000          2.0
# 1    dodge     gas        std         sedan   17000          1.8
# 2    mazda  diesel        std         sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std         sedan   25000          2.0
# ----------

bins = np.linspace(min(df["price"]), max(df["price"]), 4)

df["price-category"] = pd.cut(df["price"], bins,
                              labels=["Low", "Medium", "High"],
                              include_lowest=True)
```

```
print(len(df[df["price-category"] == "Medium"]))
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`0`

Explanation
This puzzle shows how to group values into evenly spaced bins. You use it to cluster values in three steps:

First, NumPy's `linspace()` function creates a range from the lowest to the highest price containing four evenly spaced numbers: `[17000.0, 51333.33, 85666.66, 120000.0]`. The four numbers define three bins, which comprise all values between each pair of numbers. The lowest value of the pair does not belong to the bin; the highest value does. Except for the first bin where both values are included because of the argument `include_lowest=True`.

Second, Pandas function `cut()` assigns each value from the column `price` into the corresponding bin. The three bins have the labels: `Low, Medium, High`. You store the category name for each price in a new column called `price-category`.

Third, you select all rows from the DataFrame where the value of `price-category` equals `Medium` and count how many rows there are. Since there is no price between `51333.33` and `85666.66` the result is `0`.

6.13 Exploding Values

```
# Elo 1765
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
```

```
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df = df.groupby("fuel").agg(list)
result = df["make"].explode().count()

print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
5

Explanation
The method `groupby()` of the DataFrame class groups values by a given column—so all rows with the same column values will be aggregated. You can reverse the grouping with the method `explode()`.

Since there are two different values in the column `fuel`, grouping by the column `fuel` yields a DataFrame with two rows. After applying the method `explode()`, the DataFrame has five rows again, exactly like the original DataFrame.

6.14 Comparison: equals() vs. ==

```
# Elo 1785
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df["engine-size_copy"] = df["engine-size"]
check1 = (df["engine-size_copy"] == df["engine-size"]).all()
```

```
check2 = df["engine-size_copy"].equals(df["engine-size"])
print(check1 == check2)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`False`

Explanation
This puzzle shows how to compare columns or entire DataFrames regarding the shape and the individual elements.

The comparison operator `==` returns `False` for our DataFrame because the comparing `NaN`-values with `==` always yields `False`.

The comparison operator `df.equals()` compares two Series or DataFrames. In this case, `NaN`-values in the same location are considered to be equal. The column headers do not need to have the same type, but the elements within the columns must be of the same `dtype`.

As the result of `check1` is `False` and the result of `check2` yields `True`, the final output is `False`.

6.15 Merged DataFrames Value Sources

```
# Elo 1790
import pandas as pd

df = pd.read_csv("Cars.csv")
df2 = pd.read_csv("Cars2.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
```

```
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

# Additional dataframe "df2"
# ----------
#     make   origin
# 0  skoda  Czechia
# 1  mazda    Japan
# 2   ford      USA
# ----------

result = pd.merge(df, df2, how="outer", indicator=True)
print(result.loc[2, "_merge"])
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`both`

Explanation
The solution to this puzzle is not obvious—you must understand the parameter `indicator`. If indicator is set to `True`, an additional column called `_merge` is added to the result DataFrame. In this column there is one of the following three values for each row: `left_only`, `both`, `right_only`. They indicate if values from a row come only from the left, only from the right, or from both DataFrames that were merged. In the puzzle, you take the row for `'mazda'` whose values come from both input DataFrames.

— 7 —

Pandas Code Puzzles Elo 1800+

Great work! If you've worked through all the puzzles in this book, you've significantly improved your Pandas code understanding by now. Let's finish this puzzle session with two advanced-level bonus puzzles. But be warned: only skilled Pandas experts can solve those! Can you?

7.1 Index iloc and Lambdas

```
# Elo 1800
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
```

```
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

selection = df.iloc[lambda x: x.index % 2 == 0]
result = selection.index.to_list()
print(result)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`[0, 2, 4]`

Explanation
When passing a lambda function into `iloc[]`, you select the rows for which the lambda returns `True`. The parameter `x` defines the data frame to be processed. After retrieving the selected rows, you get the index values of those rows by calling `index.to_list()` on the resulting DataFrame. In this puzzle, you select only those rows whose index value is even.

7.2 Pivot Tables

```
# Elo 1900
import pandas as pd

df = pd.read_csv("Cars.csv")

# Dataframe "df"
# ----------
#       make    fuel aspiration   body-style   price  engine-size
# 0     audi     gas      turbo        sedan   30000          2.0
# 1    dodge     gas        std        sedan   17000          1.8
# 2    mazda  diesel        std        sedan   17000          NaN
# 3  porsche     gas      turbo  convertible  120000          6.0
# 4    volvo  diesel        std        sedan   25000          2.0
# ----------

df2 = df.pivot_table(index=["fuel"], values=["make", "price"],
                     aggfunc={"price": "sum", "make": "count"},
                     ↪  columns=["aspiration"])
print(df2.shape)
```

What's the output of this code snippet?
Correct: +10 Elo points / Wrong: -10 Elo points

Answer
`(2, 4)`

Explanation
This puzzle demonstrates how to build a basic pivot table—a table with another shape and values based on the original table.

The new index is the old `fuel` column, aggregation columns are `make` and `price`, and the columns' labels come from the values of the column `aspiration`. In addition, you choose how to aggregate the values—`price` as sum and `make` by concatenation.

As a result, the new Dataframe `df2` looks as follows:

```
# Dataframe "df2"
# ----------
#             make             price
# aspiration  std  turbo       std  turbo
# fuel
# diesel      2.0  NaN     42000.0       NaN
# gas         1.0  2.0     17000.0  150000.0
# ----------
```

It has two columns and four rows.

— 8 —

Final Remarks

Congratulations, you made it through this whole Python Pandas book! You have worked through 74 Pandas puzzles and enhanced your ability to write and understand Pandas code quicker. In my experience, the main difference between an average and a master coder is the ability to grasp the meaning of source-code quickly. By studying 74 Pandas stimuli, you've trained your "Pandas eye" and it'll help you solve data problems quicker and with higher accuracy.

Your Skill Level

You should have a fair estimate of your skill level compared to others—be sure to check out Table 3.1 again to get the respective rank for your Elo rating. This book is all about pushing you from beginner to intermediate Pandas coding level.

Consistent effort and persistence is the key to success. If you feel that solving code puzzles has advanced your skills, make it a daily habit to solve one Python puzzle a day, and watch the related video

on the Finxter.com web app. Build this habit into your life—e.g., use your morning coffee break routine—and you will soon become one of the best programmers you know.

Where to Go From Here?

We publish a fresh code puzzle every couple of days on our website `https://finxter.com`. Our goal with Finxter is to make learning to code easy, individualized, and accessible.

- We worked hard to make this book as valuable for you as possible. But no book can reach perfection without feedback from early adopters and highly active readers. For any feedback, questions or problems you may have, please send us an email at `admin@finxter.com`.

- We highly appreciate your honest book review on your preferred bookseller (e.g., Amazon). We don't spend tons of money on advertising and rely on loyal Finxters to spread the word. Would you mind leaving a review to share your learning experience with others?

- To grow your Python skills on autopilot, feel free to register for the Python email academy with many free email courses and cheat sheets here: `https://blog.finxter.com/subscribe/`.

- You have now stretched your Python skills beyond the intermediate level. There is no reason why you should not start selling your skills in the marketplace right now. If you want

to learn how to sell your skills as a Python freelancer effectively, watch the free *"How to Build Your High-Income Skill Python"* webinar at `https://blog.finxter.com/webinar-freelancer/`.

- This is the sixth book in the *Coffee Break Python* series, which is all about pushing you—in your daily coffee break—to an advanced Python level. Please find the other books below.

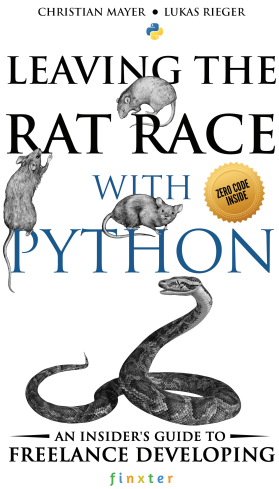
Finally, we would like to express our deep gratitude that you have spent your time—solving code puzzles and reading this book. Above everything else, we value your time. The ultimate goal of any good textbook should be to *save you time*. By working through this book, you have gained insights about your coding skill level. If you apply your Python skills in the real world, you will experience a positive return on invested time and money. Keep investing in yourself, work on practical projects, and stay active within the Finxter community. This is the best way to improve your Python skills continuously.

More Python Textbooks

This Pandas Book extends the "Coffee Break Python" textbook series. It helps you master computer science with a focus on Pandas and Data Science basics. The other textbooks are:

Leaving the Rat Race with Python: An Insider's Guide to Freelance Developing

Leaving the Rat Race with Python shows you how to nurture, grow, and harness your work-from-home coding business online — and live the good life!



Get the ebook:
`https://blog.finxter.com/book-leaving-the-rat-race-with-python/`

Get the print book:
`https://smile.amazon.com/gp/product/B08G1XLDNB/`

Coffee Break Python: 50 Workouts to Kickstart Your Rapid Code Understanding in Python

The first bestselling book of the "Coffee Break Python" series offers 50 educative code puzzles, 10 tips for efficient learning, 5 Python cheat sheets, and 1 accurate way to measure your coding skills.

Get the ebook:
`https://blog.finxter.com/coffee-break-python/`

Get the print book:
`http://bit.ly/cbpython`

Coffee Break Python Slicing: 24 Workouts to Master Slicing in Python, Once and for All.

Coffee Break Python Slicing is all about growing your Python expertise—one coffee at a time. The focus is on the important technique: slicing. You use this to access ranges of data from Python objects. Understanding slicing thoroughly is crucial for your success as a Python developer.

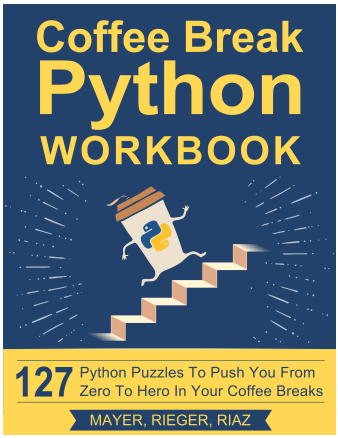
As a bonus, you track your Python coding skill level throughout the book.

Get the ebook:
`https://blog.finxter.com/coffee-break-python/`

Get the print book:
`http://bit.ly/cbpslicing`

Coffee Break Python Workbook: 127 Workouts to Kickstart Your Rapid Code Understanding in Python.

This books contains 127 additional Python puzzles with detailed explanation for each. If you liked the first book, this is the perfect way to deepen your understanding of beginner to intermediate Python concepts.



Get the ebook:
`https://blog.finxter.com/coffee-break-python-workbook/`

Get the print book:
`http://bitly.ws/9VTi`

Coffee Break Python Mastery Workouts: 99 Tricky Python Puzzles to Push You to Programming Mastery

Are you a Python master? Try to solve these 99 tricky Python puzzles! The book contains cheat sheets and offers detailed explanations for each puzzle. But be careful! The puzzles are only for advanced Python coders and require an advanced understanding of Python. Check them out if you want to reach Python mastery!

Get the ebook:

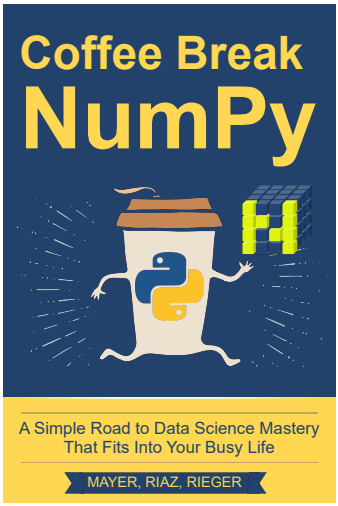
`https://blog.finxter.com/coffee-break-python-mastery-workout-ebook/`

Get the print book:

`http://bitly.ws/9VTK`

Coffee Break NumPy: A Simple Road to Data Science Mastery That Fits Into Your Busy Life.

Coffee Break NumPy is a new step-by-step system to teach you how to learn Python's data science library faster, smarter, and better. Solve practical Python NumPy puzzles as you enjoy your morning coffee.



Get the ebook:
`https://blog.finxter.com/coffee-break-numpy/`

Get the print book:
`http://bit.ly/cbnumpy`