DATA SCIENTIST

INTERVIEW

TIPS & CHEATSHEET

EBOOK by
SANJAY KV

CONTENTS

What's in this guide?

"Information is the oil of the 21st century, and analytics is the combustion engine" Peter Sondergaard.

I've broken this guide down into 4 sections: **Product**, **SQL,** and **Statistics & Probability**, **Data science Cheatsheet** This follows from my interviewing experience

In the Product Sense section, you'll find a breakdown of common interview questions, my personal answer frameworks, and resources to dig deeper. .

In the SQL and Statistics & Probability sections, I provide resources I used to prepare and tips for learning and studying the concepts.

In the Product Sense section, you'll find a breakdown of **common interview questions,** my personal answer frameworks, and resources to dig deeper.

The final results you will get after reading this book:

- *These buckets capture most of what's expected in data science/data analyst interviews.*
- *These resources are helpful if you're interviewing for anything related to the product (product management, product marketing, product design, etc.) and want to bring a data science perspective to the table.*
- *Will able to answer most of the basic interview questions*
- *The Cheatsheet will help you to get more direct approach to the machine learning and implemntation. Remember there is no point remebering those cheatsheet attached at the end of this book if you dont execute. Make projects and publish it on GitHub and try to do more open source projects.*
- *Refer my GitHub to get started with real time Data science projects.*

I didn't include any tips for networking or landing your first interview, since I think there's a good amount of documentation on that realm of recruiting already. That said, if you ever want to chat more about getting your foot in the door, I'm here!

Introducing Data Science

What is Data Science?

Data science — specifically, "product data science" or "**product analytics**" — is defined differently by every tech company.

But at its core, data science (in the context of a technology product) involves analyzing data to better understand users and then leveraging that understanding to drive better outcomes for the organization.

1. **What's the buzz about data science?**
   Let's sift through the buzzwords for a sec. You already know that apps like Facebook use data to serve you personalized ads. But Facebook (and all modern products) tackle even tougher problems using data, too.
   For example:
   How does Facebook know if a friend request you received is potentially dangerous? How does Netflix decide which new show or movie to spotlight when you open the app? How does Uber determine if you canceled a ride because it was taking too long or because you decided not to make the trip anymore?

2. Such questions are critical to these organizations, but they don't have clear answers. That's why these companies and their employees use data to drive their decisions on these issues. Without data, they'd be fumbling around in the dark, relying on untrustworthy intuition and anecdotal evidence. Analyzing data is like putting on night-vision goggles, allowing them to navigate these ambiguous situations with accuracy and confidence and thus make better decisions.

3. At a high level, I like to think of data science as user research at a macro scale. Instead of conducting surveys and interviews with a limited number of users, we can analyze data on how huge numbers of users use a product, better understand what they value and what problems they're facing, and leverage that understanding to determine what features to build, how to fix bugs, whether a product is successful, and so on.

Taking another step back, we can even view data science in the context of time. In data science, we're essentially asking: "What can we learn from the past, and how can we apply it to the future?" With the tools of math, statistics, and computation, we turn data into knowledge

→ (past to present). Then, with soft skills like communication, empathy, and critical thinking, we turn that knowledge into decisions (present → future).

3 Important Steps:

1. *Data: Measurement of what has occurred*
2. *Knowledge: Understanding of why it occurred*
3. *Decisions: Informed choices based on knowledge*

There are many ways to conceptualize the role of data science in tech, so I encourage you to create your own mental model, too. Having a clear understanding of the role will be infinitely helpful to crushing your interviews. *Those details has been given at the end of the book, you can refer different domains in Data Science and respective salary.*

Product Sense:

What is product sense?

While there are many ways to think about product sense, here's what I think it is: Product sense is an understanding of how the interaction of People, Products, and Business creates the product experience, successfully or unsuccessfully. Technology ("Products") delivers value to users ("People"), and that value is captured through a Business model.

- **People.** *What do people want or need?*
- **Products.** *How can teachnology deliver what people want or need?*
- **Business.** *How to moetize the relationship between people and products?*

Company	People (User needs)	Product (Technology)	Business (Monetization)
LinkedIn	Building network	A social platform for people	Advertisments
Spotify	Stream music	Unlimited access to music	Monthly subscription fee
Airbnb	Stay while Travelling	Platform to connect	Travelers pay Airbnb a fee

Pretty simple, right? Of course, there are other ways these organizations make money, but these examples show that you can apply this framework to understand how any tech company works at a fundamental level. While you're interviewing, you can always fall back on these three principles — people, products, and business — to steer your answer in the right direction.

users needs? | ***Product Technology?*** | ***Business Monetization?***

Product & Data scientists?

Why is product sense important for data scientists?

Data scientists drive product decisions, so the better they understand a product's context, the more impact they'll have. In many conversations I had with people at **Google** and other companies, that idea was the recurring theme: data scientists are valuable because they draw on quantitative skills and strong product instincts to connect data with business goals.

Showcasing this ability to juggle both the technical and non-technical elements of the role is how you stand out as a candidate, especially as a student or new grad. In product sense interviews, your interviewers aren't looking for you to give the right answer — because there isn't one. Instead, they're looking for you to 1) ground your answer in logic and evidence and 2) communicate your answer so that others can understand your reasoning. It's tough to do both, but it's what makes this role so valuable — and, in my opinion, so cool!

FRAMEWORKS HOW TO ANSWER

While there are many types of questions you can be asked, I think they can be bucketed into 2 categories:

1) how you'd measure the performance of a product ("**product success**")

2) how you'd diagnose a change in a product's metrics ("**metric change**")

Accordingly, I've created 2 different frameworks to answer these questions. Of course, these frameworks don't cover all of the questions you could be asked, and the questions are never asked point-blank (there are typically part of a broader discussion between you and your interviewer). But I hope that having this baseline of logical questions, trains of thought, and communication patterns helps you crush your interviews . Put your own spin on them, too!

GENERAL FRAMEWORK

Your The 2 frameworks I describe below follow the same 4 steps:

Contextualize, Hypothesize, Analyze, Synthesize

The specifics of the Analyze step vary between frameworks, but the Contextualize, Hypothesize, and Synthesize steps are all the same

Let's go through each step together.

CONTEXTUALIZE

The questions asked in these interviews are typically vague and open-ended. You need to understand the context surrounding the problem in order to give a meaningful answer. In general, interviewers expect you to ask questions: it shows them that you're not just trying to answer the question — you're trying to answer the right question.

For example, suppose you're asked: "We see that engagement on Instagram Stories is dropping. What data should we analyze to understand why?"

You could answer this question simply by rattling off factors that negatively impact engagement (usability issues, competition from other apps, etc.). But you'll give a more precise and actionable answer if you better understand the context in which this situation occurs. A few details to consider from this example:

HYPOTHESIZE

Initially, the idea of "guessing" or "hypothesizing" the answer felt counterintuitive to me. Isn't the whole point of data science not to guess, and instead use data to make decisions?

It is indeed! But a number of recruiters, interviewers, and mentors recommended this to me because a key skill for data scientists is prioritization.

There are always many ways to solve a data science problem, but there isn't time to try every solution. Starting with what you think is the most logical answer — your hypothesis — simply increases efficiency.

> *Why has the overall number of views on IG stories declined over the past three months in the US?*

➲ *To hypothesize an answer, fall back on the 3 aspects of product sense: people, products, and business.*

EXAMPLE

People: Have people's needs changed in the past 3 months?

Ex. Did the easing of COVID-19 restrictions in the US reduce the downtime people had to scroll through IG Stories?

Product: Did the product change in the past 3 months? Would we expect this change to produce a decline in engagement?

Ex. Was a new feature of IG Stories launched unsuccessfully? Has another feature on IG launched and taken away ("cannibalized") some viewership from IG Stories?

Business: Did the business model change in the past 3 months?

Ex. Did IG Stories start showing more ads, resulting in a worse user experience and lower viewership?

ANALYZE

This step varies between frameworks because different questions call for different data, so **see the frameworks below for more details.**

The fundamental idea here, though, is to explain how you'd methodically evaluate your hypothesis using data. The thinking goes like this: break down your hypothesis into its underlying assumptions, and then discuss how you'd use data to investigate the truth of those assumptions. If those assumptions can be proven to be valid, then your hypothesis — your answer to the question — is sound.

And that's the **key to answering these interview questions:** verbalize what you'd learn from the data, not just what data you'd look at. Returning to our IG Stories example, you might suggest investigating the overall time spent by users on Instagram in the past 3 months. The "overall time spent on the app" is a great metric to look at here, but to give the best answer, you should explain why. Namely:

- *"The overall time spent on the app is a good indication of the performance of the IG app overall. If this metric has also declined in the past 3 months, then there is likely a problem with the app beyond just IG Stories. If this metric has not shown a similar decline, then we know this issue is specific to IG Stories." ?*
- *Once you've analyzed your hypothesis, you're ready to put it all together. →*

SYNTHESIZE

To wrap things up, give a 2-sentence summary of 1) the problem you're trying to solve, 2) your hypothesis, and 3) how you'd use data to validate or invalidate your hypothesis. Doing so not only reminds your interviewer of what you've said, but it also demonstrates your ability to communicate effectively.

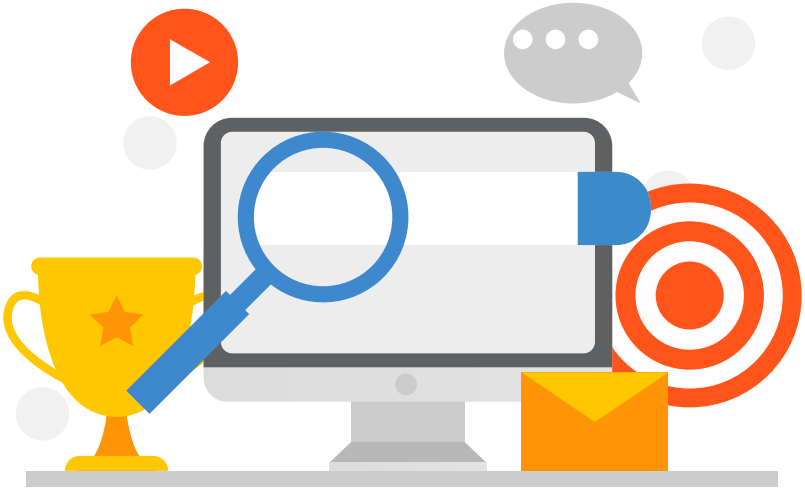
Finally, as a cherry-on-top, mention that you'd work with other stakeholders (product managers, designers, researchers) to conduct this analysis and describe how these results would be useful to them.

Usually, the conversation won't end here. Your interviewer will likely ask follow up questions to either broaden the discussion or venture into specifics. At that point, you can still utilize these 4 steps in your following answers, but make sure to allow ample space for a back-and-forth conversation between you and your interviewer. **Collaboration is key!**

In addition, you should also briefly tie up any loose ends. Are there lingering hypotheses you have? What blind spots might your analysis have?

For example, if the question involves building a new feature, you can say that you'd work with the product manager to figure out the **product roadmap** (the timeline along which the product is built). Or, if the question involves improving a product, you can say that you'd work with product designers to implement the changes you've discussed.

Now, let's discuss the "Analyze" step for the 2 frameworks. →



These questions focus on understanding if a product is performing as it should. It sounds intuitive, but defining what a product "should" be doing and actually measuring its performance against that standard is tough. For example:

"Product Success" Framework

- *(Uber) "How do we measure the success of Uber's new car-rental feature?"*
- *(Spotify) "How do we know if Spotify Wrapped performed well?"*
- *(Gmail) "How do we know if the "Undo send" feature is useful to users?"*

"Metric Change" Framework

Data scientists are often in charge of monitoring a product's health by tracking key metrics that measure the success of a product. These questions focus on understanding how you'd approach a situation in which one of those key metrics has changed negatively. For example:

1. *(Facebook) "We see that DAU (daily active users) of Facebook Marketplace has consistently declined in the past week. How would you figure out why this is occurring?"*
2. *(YouTube) "The number of new videos posted weekly has been trending down for the past month. What analysis would you conduct to figure out why?"*
3. *(Yelp) "The number of new reviews posted monthly has declined every month for the past 4 months. What would you do to figure out why?"*

COMPETITION

Has a new product launched that's taken some of our engagement? Can we measure how much of our declining engagement can be attributed to these competitors?

PUBLIC RELATION

Has there been any bad press related to the feature? How many people have been exposed to that press, and how much can we expect that to impact users' engagement?

USER BEHAVIOR

Has there been a shift in user preferences? Over what time period has this occurred? How rapidly will this impact our metrics?

WRAPPING UP

Remember, there's no right answer here, so don't get caught up in giving a buzzword-heavy, technically-correct answer. Your perspective is valuable regardless of your experience — just focus on sharing your thoughts clearly and succinctly.

Interview Tips

While it's helpful to have these frameworks, avoid making your answers feel too scripted. Unlike consulting case interviews, your interviews aren't expecting you to follow a specific thought process — they just want to see how you think and communicate.

My favorite piece of interview advice I've ever received is to adopt the mindset that you're already a data scientist, and this interview is just another meeting you're having with a coworker. You're trying to solve the problem together; there's no right or wrong answer. I found that this mentality helps calm my nerves because I focus less on the "correctness" of my answers and more on collaborating with a team member.

How I'd recommend preparing: a month before your first interview, review one of these types of questions every day (see below for where to find example questions). It only takes ~10 minutes, and this consistency will cultivate a sense of ease when answering the questions. Instead of trying to hit every point of your framework, the thoughts will follow more smoothly.

Here are a few questions you can ask:

- *How will i be evaluated?*
- *How will the projects I work on align to business goals?*
- *Who will I be working with?*
- *How does the data science team collaborate with other departments?*
- *What training and professional development opportunities are available?*
- *What tool set do you use, and are you open to using new ones?*

SQL:

What is SQL?

SQL (**Structured Query Language**) is one of the most popular coding languages for data scientists/analysts, and it's becoming increasingly popular for anyone who uses data (business analysts, financial analysts, operations analysts, etc.) to learn SQL as well. That's because SQL is easy to learn and simple to use, but it's also powerful: it lets you apply data analysis concepts like **sums, averages**, and **aggregations** to huge data sets that are too large for a program like Excel.

Is SQL actually easy to learn?

Relative to other coding languages, it absolutely is! In my opinion, it's the most natural and non-"computery" language.

SQL lets you analyze data by structuring a series of "queries" (i.e. "requests" to pull data) like this one. I think the systematic nature of SQL makes it fun to learn; the different query types are like Legos that fit together, but you have to arrange them in a specific order to get it right.

While many schools don't teach SQL in classes, it's a great skill to pick up on your own to make your resume more competitive, and you'll often need to code in SQL for interviews, too.

Below is summarised repeated questions in interview.

- *What is an SQL View?*
- *As a data architect, have you faced any challenges related to the company's data security? How did you ensure the integrity of the data was not compromised?*
- *A lot of companies use data from both internal and internal sources. Have you faced any problems while trying to integrate a new external data source into the existing company's infrastructures? How did you solve these problems?*
- *State and describe the different types of SQL joins.*
- *How many types of data structures does R have?*
- *What modeling tools have you used in your work so far? Which do you consider efficient or powerful?*
- *In your role as a data architect, what metrics have you created or used to measure the quality of new and existing data?*

Resource to imporve SQL Knowledge

Free self-guided course: -> HERE

Mode is a great place to start. Its beautiful interface and organized content make learning SQL digestible and enjoyable. Below are the topics importanat in SQL

1. *Relational Databases - Relational Model*
2. *Relational Databases - Querying*
3. *Database Management Notes*
4. *Basic SQL Commands*
5. *Role of SQL in Machine learning*
6. *SQL Technique : Limiting Data, Slicing Data, Extreme Value Identification, Aggregation Functions, Counting Rows and Items*
7. *Groupings, Rolling up Data and Filtering in Groups*
8. *SQL Technique : Filtering Patterns*

Advanced SQL Topics for Data Scientists

- **Cursors in SQL.**
- **Stored Procedures Automation Scripts**
- **Views in SQL.**
- **Linked Server in SQL**
- **Subqueries and window in SQL**

SQL is self Explanatory, not going deeper or basics query explanation, covering above topics helps to get start.

Statistics & Probability:

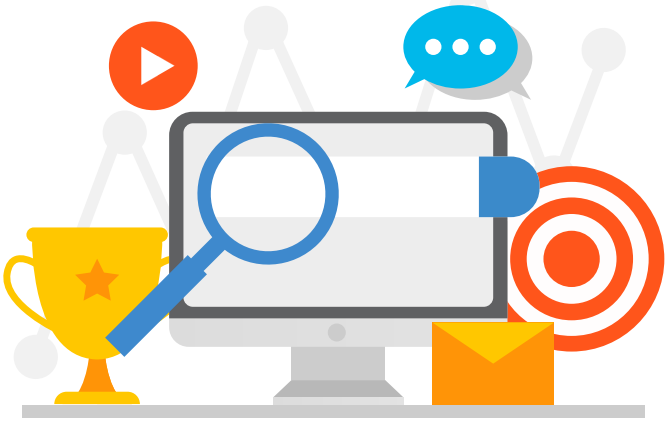
How are statistics and probability different?

As explained in this lesson by Brilliant.org, you can think of statistics as "solving backward" and probability as "solving forward." In statistics, we start with data, and we use math to understand the phenomenon that generated that data. In **probability,** we use our understanding of the data (usually gained through statistics) **to describe how likely different outcomes are in the future.** There's considerable overlap between the fields, and in data science, we use both.

Why are statistics and probability important for data scientists?

The power of data science lies in the information you can juice out of existing data (this process is called "**data mining**", in the sense that we "**mine**" data for information). As data scientists, the more statistics & probability you know, the more tools you have to extract insights from data, and thus the more helpful you are to the organization.

For example, let's say you're a data scientist at **Amazon.com** analyzing sales of a particular product. You could run a descriptive analysis to produce basic insights, such as "55% of buyers of this product were female," "20% of buyers live in California", etc. Having these statistics is **better than having no information at all**, but it's hard to take action based on these findings. Alternatively, you could fit a **multiple regression** model to the same data, which would let you calculate how likely it is that a buyer will buy the product based on their profile (location, gender, etc.). Then, you could decide that if a buyer is more than 60% likely to purchase it, Amazon should target them with an ad for the product.



INTERVIEW PREPRATIONS?

This depends on the role you're applying for. Since data science is so broad, roles can lean toward machine learning, business analytics, experiment design, and many other niches, which require different extents of statistics and probability knowledge. Often, job descriptions will describe the extent of stats and probability you're expected to know, but if not, definitely feel free to ask your recruiter to clarify expectations before heading into an interview or online skills exam.

KHAN ACADEMY

Khan Academy free course: **khanacademy.org** You can't go wrong with Khan Academy — it's a great place to brush up on concepts you've probably already learned in school.

TERENCE SHINE's

This is a compilation of a ton of additional resources for statistics and probability for data science, including more advanced topics like machine learning, regression, and linear algebra.

BRILLIANT.ORG

Brilliant.org free course: **brilliant.org**
Brilliant is an artsier, more interactive version of Khan Academy. It focuses on beautiful visualizations that teach the intuition behind concepts. I personally found their content extremely helpful for understanding how to explain complicated mathematical concepts

CRACKING INTERVIEW:

Python Importance

The first of many benefits of Python in data science is its simplicity. While some data scientists come from a computer science background or know other programming languages, many come from backgrounds in statistics, mathematics, or other technical fields and may not have as much coding experience when they enter the field of data science. Python syntax is easy to follow and write, which makes it a simple programming language to get started with and learn quickly.

Python doesn't convince you of the importance of Python for data science, maybe the libraries available to make your data science coding easier will. A library in Python is a collection of modules with pre-built code to help with common tasks. They essentially allow us to benefit from and build on top of the work of others. In other languages, some data science tasks would be cumbersome and time consuming to code from scratch. There are countless libraries like **NumPy, Pandas, and Matplotlib** available in Python to make data cleaning, data analysis, data visualization, and machine learning tasks easier. Some of the most popular libraries include:

In addition to all of the general data manipulation libraries available in Python, a major advantage of Python in data science is the availability of powerful machine learning libraries.

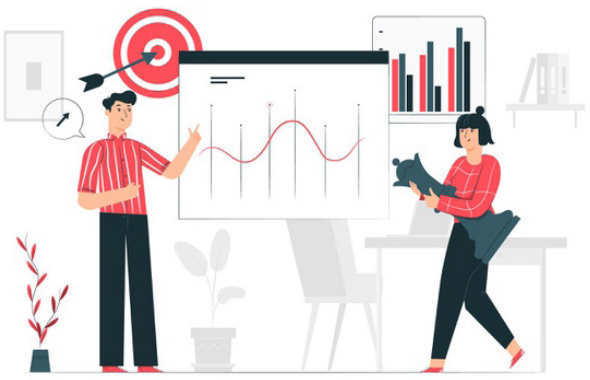
These machine learning libraries make data scientists' lives easier by providing robust, open source libraries for any machine learning algorithm desired. These libraries offer simplicity without sacrificing performance.

PY LIBRARY

PYTHON

- **NumPy.** *N*umPy is a Python library that provides support for many mathematical tasks on large, multidimensional arrays and matrices.
- **Pandas:** The Pandas library is one of the most popular and easy-to-use libraries available. It allows for easy manipulation of tabular data for data cleaning and data analysis.
- **Matplotlib:** This library provides simple ways to create static or interactive boxplots, scatterplots, line graphs, and bar charts. It's useful for simplifying your data visualization.
- **Seaborn:** Seaborn is another data visualization library built on top of Matplotlib that allows for visually appealing statistical graphs. It allows you to easily visualize beautiful confidence intervals, distributions, and other graphs.
- **Statsmodels:** This statistical modeling library builds all of your statistical models and statistical tests including linear regression, generalized linear models, and time series analysis models.
- **Scipy:** Scipy is a library used for scientific computing that helps with linear algebra, optimization, and statistical tasks.

- **Scikit-learn:** Popular machine learning library is a one-stop-shop for all of your machine learning needs with support for both supervised and unsupervised tasks
- **Tensorflow** Tensorflow is a high-level library for building neural networks. Since it was mostly written in C++, this library provides us with the simplicity of Python without sacrificing power and performance.
- **Keras:** Keras is a popular high-level API that acts as an interface for the Tensorflow library. It's a tool for building neural networks using a Tensorflow backend that's extremely user friendly and easy to get started with.
- **Pytorch:** Pytorch is another framework for deep learning created by Facebook's AI research group.



QUESTIONS:

Tricky Questions

1. **What is pep 8?**

Ans: PEP stands for Python Enhancement Proposal. It is a set of rules that specify how to format Python code max readability.

2. **Is indentation required in python?**

Ans: Indentation is necessary for Python. It specifies a block of code. All code within loops, classes, functions, etc is specified within an indented block. It is usually done using four space characters. If your code is not indented necessarily, it will not execute accurately and will throw errors as well.

3. What is the difference between Python Arrays and lists?
Ans; Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

4.What is a lambda function?
Ans: An anonymous function is known as a lambda function. This function can have any number of parameters but, can have just one statement.

5. What is the difference between range & xrange?
Ans: For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and x range returns an xrange object.

6.What are docstrings in Python?
Ans: Docstrings are not actually comments, but, they are documentation strings. These docstrings are within triple quotes. They are not assigned to any variable and therefore, at times, serve the purpose of comments as well.

7.What is a dictionary in Python?
Ans: The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

8. What is a dictionary in Python?

Ans: The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

9. What does this mean: *args, **kwargs? And why would we use it?

Ans: We use *args when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. **kwargs is used when we don't know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise.

10. What are negative indexes and why are they used?

Ans: The sequences in Python are indexed and it consists of the positive as well as negative numbers. The numbers that are positive uses '0' that is uses as first index and '1' as the second index and the process goes on like that.

The index for the negative number starts from '-1' that represents the last index in the sequence and '-2' as the penultimate index and the sequence carries forward like the positive number.

The negative index is used to remove any new-line spaces from the string and allow the string to except the last character that is given as S[:-1]. The negative index is also used to show the index to represent the string in correct order.

Below are further sample snippets on Python

- *Array elements can be removed using pop() or remove() method.*
- *Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance.*
- *Deep copy is used to store the values that are already copied. Deep copy doesn't copy the reference pointers to the objects.*
- *Python has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it.*
- *Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time.*
- *The compiling and linking allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure.*
- *Python libraries are a collection of Python packages. Some of the majorly used python libraries are - Numpy. Pandas. Matplotlib. Scikit-learn and many more.*

DATA SCIENCE FUTURE CAREERS

- *Data Scientist - : $139,840*
- *Machine Learning Engineer - $114,826*
- *Machine Learning Scientist - $114,121*
- *Data Architect - $108,278*
- *Data Engineer - $102,864*
- *Business Intelligence (BI) Developer - $81,514*
- *Data Analyst - $62, 453*
- *Statistician - $76,884*

The above Roles and salary has been summarised based on the real time database from Glassdoor and LinkedIn salary on USA Region. If you would like to know how much these roles getting paid in your country, go to the above site and filter by your country.

Note From Author: Sanjay K V

Hola,

I created this overview to help you to crack the Data Scientist Interview. I hope this will help you to an extend for data science realated Interviews.

Few of the resource is picked from the Data course i undergone and Interview experience i had so far. credits has given to the same. If you would like to connect with me click **below**

CONNECT WITH ME

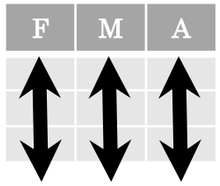
Keep Learning
Sanjay

Grab your (free) **GITHUB TUTORIALS** here

Data Wrangling
with pandas Cheat Sheet

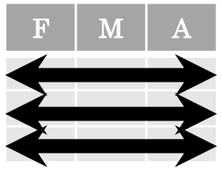
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



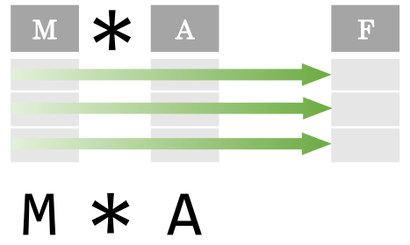
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
      .rename(columns={
              'variable' : 'var',
              'value' : 'val'})
      .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

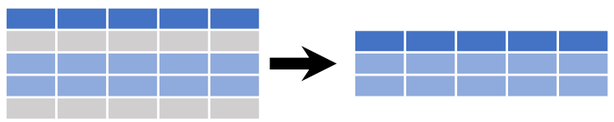
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data

`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

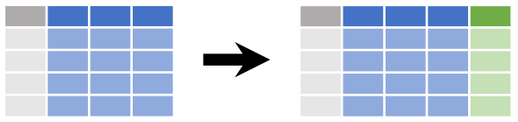
`cumprod()`
Cumulative product.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

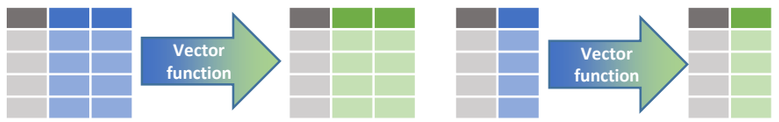
Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

`abs()`
Absolute value.

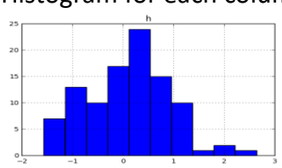
Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

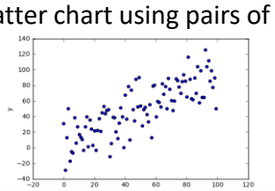
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Python For Data Science *Cheat Sheet*

NumPy Basics

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

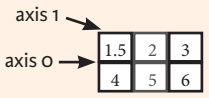
Use the following import convention:

```
>>> import numpy as np
```

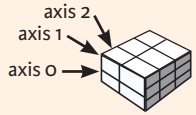
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

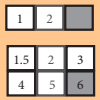
```
>>> h = a.view()                     Create a view of the array with the same data
>>> np.copy(a)                       Create a copy of the array
>>> h = a.copy()                     Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()                         Sort an array
>>> c.sort(axis=0)                   Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists



Slicing

```
>>> a[0:2]                           Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                         Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                            Select all items at row 0
  array([[1.5, 2., 3.]])                (equivalent to b[0:1, :])
>>> c[1,...]                         Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                        Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                           Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]    Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]    Select a subset of the matrix's rows
  array([[ 4. , 5. , 6. , 4. ],        and columns
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)              Permute array dimensions
>>> i.T                              Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                        Flatten the array
>>> g.reshape(3,-2)                  Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                  Return a new array with shape (2,6)
>>> np.append(h,g)                   Append items to an array
>>> np.insert(a, 1, 5)               Insert items in an array
>>> np.delete(a,[1])                 Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]     index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

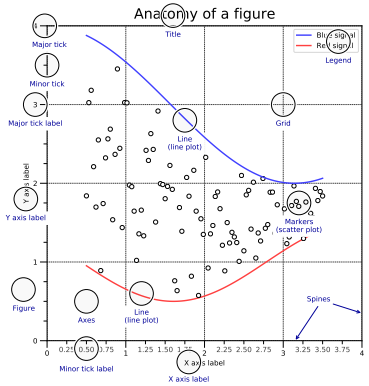
matplotlib Cheat sheet

Version 3.2

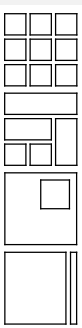
Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X,Y,color='C1')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a ﬞ e



Subplots layout



```
subplot[s](cols,rows,…)
fig, axs = plt.subplots(3,3)

G = gridspec(cols,rows,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax=d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)`
X, Y, fmt, color, marker, linestyle

`scatter(X,Y,…)`
X, Y, [s]izes, [c]olors, markers, cmap

`bar[h](x,height,…)`
x, height, width, bottom, align, color

`imshow(Z,[cmap],…)`
Z, cmap, interpolation, extent, origin

`contour[f]([X],[Y],Z,…)`
X, Y, Z, levels, colors, extent, origin

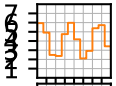
`quiver([X],[Y],U,V,…)`
X, Y, U, V, C, units, angles

`pie(X,[explode],…)`
Z, explode, labels, colors, radius

`text(x,y,text,…)`
x, y, text, va, ha, size, weight, transform

`fill[_between][x](…)`
X, Y1, Y2, color, where

Advanced plots



`step(X,Y,[fmt],…)`
X, Y, fmt, color, marker, where

`boxplot(X,…)`
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr,…)`
X, Y, xerr, yerr, fmt

`hist(X, bins, …)`
X, bins, range, density, weights

`violinplot(D,…)`
D, positions, widths, vert

`barbs([X],[Y], U, V, …)`
X, Y, U, V, C, length, pivot, sizes

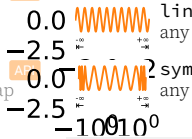
`eventplot(positions,…)`
positions, orientation, lineoffsets

`hexbin(X,Y,C,…)`
X, Y, C, gridsize, bins

`xcorr(X,Y,…)`
X, Y, normed, detrend

Scales

```
ax.set_[xy]scale(scale,…)
```



Projections

```
subplot(…,projection=p)
```

p='polar'

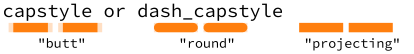
p='3d'

p=Orthographic()
from cartopy.crs import Cartographic

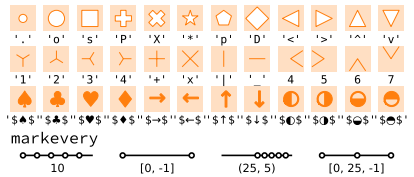
Lines

linestyle or ls

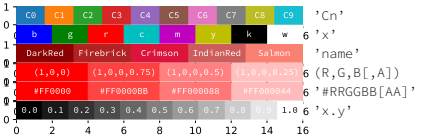
capstyle or dash_capstyle



Markers

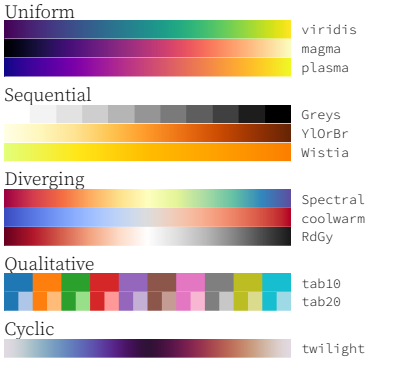


Colors



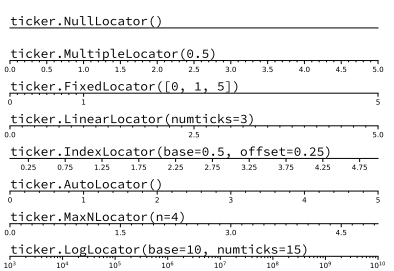
Colormaps

```
plt.get_cmap(name)
```



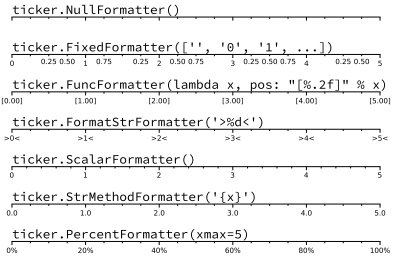
Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



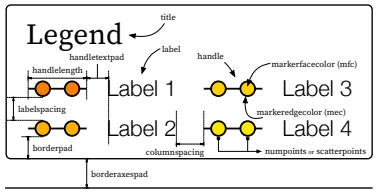
Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

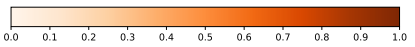


Ornaments

`ax.legend(…)`
handles, labels, loc, title, frameon



`ax.colorbar(…)`
mappable, ax, cax, orientation



`ax.annotate(…)`
text, xy, xytext, xycoords, textcoords, arrowprops

Event handling

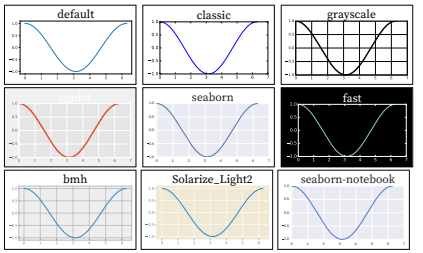
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0,2*np.pi,100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
  line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
  plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

ax.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

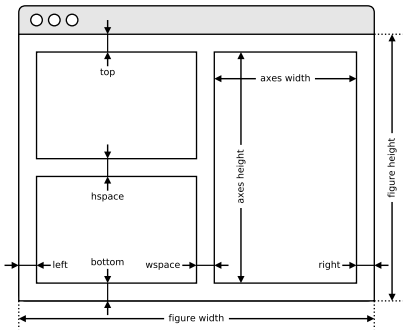
Ten Simple Rules

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

Axes adjustements

API

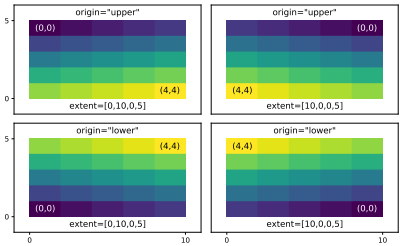
plt.**subplot_adjust**(...)



Extent & origin

API

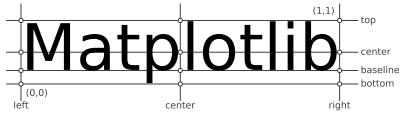
ax.**imshow**(extent=..., origin=...)



Text alignments

API

ax.**text**(..., ha=... , va=..., ...)



Text parameters

API

ax.**text**(..., family=... , size=..., weight = ...)
ax.**text**(..., fontproperties = ...)

xx-large (1.73)
x-large (1.44)
large (1.20)
medium (1.00)
small (0.83)
x-small (0.69)
xx-small (0.58)

black (900)
bold (700)
semibold (600)
normal (400)
ultralight (100)

monospace
serif
sans
cursive
italic
normal
small-caps
normal

Uniform colormaps

viridis
plasma
inferno
magma
cividis

Sequential colormaps

Greys
Purples
Blues
Greens
Oranges
Reds
YlOrBr
YlOrRd
OrRd
PuRd
RdPu
BuPu
GnBu
PuBu
YlGnBu
PuBuGn
BuGn
YlGn

Diverging colormaps

PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic

Qualitative colormaps

Pastel1
Pastel2
Paired
Accent
Dark2
Set1
Set2
Set3
tab10
tab20
tab20b
tab20c

Miscellaneous colormaps

terrain
ocean
cubehelix
rainbow
twilight

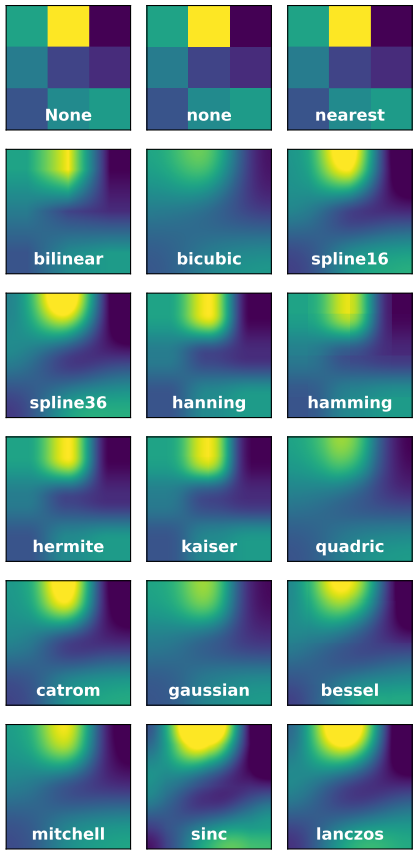
Color names

API

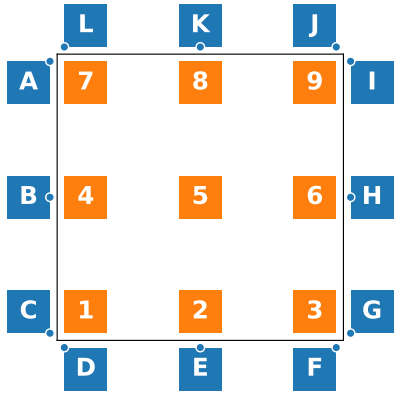
black
k
dimgray
dimgrey
gray
grey
darkgray
darkgrey
silver
lightgray
lightgrey
gainsboro
whitesmoke
w
white
snow
rosybrown
lightcoral
indianred
brown
firebrick
maroon
darkred
r
red
mistyrose
salmon
tomato
darksalmon
coral
orangered
lightsalmon
sienna
seashell
chocolate
saddlebrown
sandybrown
peachpuff
peru
linen
bisque
darkorange
burlywood
antiquewhite
tan
navajowhite
blanchedalmond
papayawhip
moccasin
orange
wheat
oldlace
floralwhite
darkgoldenrod
goldenrod
cornsilk
gold
lemonchiffon
khaki
palegoldenrod
darkkhaki
ivory
beige
lightyellow
lightgoldenrodyellow
olive
y
yellow
olivedrab
yellowgreen
darkolivegreen
greenyellow
chartreuse
lawngreen
honeydew
darkseagreen
palegreen
lightgreen
forestgreen
limegreen
darkgreen
g
green
lime
seagreen
mediumseagreen
springgreen
mintcream
mediumspringgreen
mediumaquamarine
aquamarine
turquoise
lightseagreen
mediumturquoise
azure
lightcyan
paleturquoise
darkslategray
darkslategrey
teal
darkcyan
c
aqua
cyan
darkturquoise
cadetblue
powderblue
lightblue
deepskyblue
skyblue
lightskyblue
steelblue
aliceblue
dodgerblue
lightslategray
lightslategrey
slategray
slategrey
lightsteelblue
cornflowerblue
royalblue
ghostwhite
lavender
midnightblue
navy
darkblue
mediumblue
b
blue
slateblue
darkslateblue
mediumslateblue
mediumpurple
rebeccapurple
blueviolet
indigo
darkorchid
darkviolet
mediumorchid
thistle
plum
violet
purple
darkmagenta
m
fuchsia
magenta
orchid
mediumvioletred
deeppink
hotpink
lavenderblush
palevioletred
crimson
pink
lightpink

Image interpolation

API



Legend placement



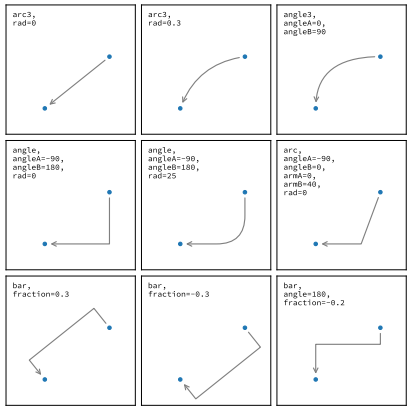
ax.**legend**(loc="string", bbox_to_anchor=(x,y))

1: lower left
2: lower center
3: lower right
4: left
5: center
6: right
7: upper left
8: upper center
9: upper right

A: upper right / (-.1,.9)
B: right / (-.1,.5)
C: lower right / (-.1,.1)
D: upper left / (-.1,-.1)
E: upper center / (.5,-.1)
F: upper right / (.9,-.1)
G: lower left / (1.1,.1)
H: left / (1.1,.5)
I: upper left / (1.1,.9)
J: lower right / (.9,1.1)
K: lower center / (.5,1.1)
L: lower left / (.1,1.1)

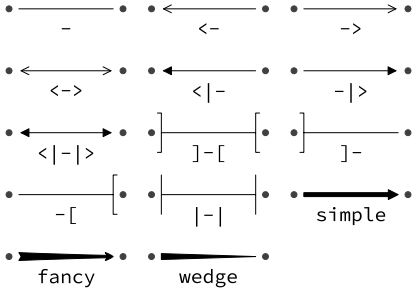
Annotation connection styles

API



Annotation arrow styles

API



How do I ...

... resize a figure?
→ fig.set_size_inches(w,h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure?
→ ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_xticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.set_[xy]ticks(rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0),1,1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R+0.5870*G+0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(…), canvas.draw()        slow
im.set_data(…), canvas.draw()          fast
```

Beyond Matplotlib

Seaborn: Statistical Data Visualization
Cartopy: Geospatial Data Processing
yt: Volumetric data Visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A Grammar of Graphics for Python

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

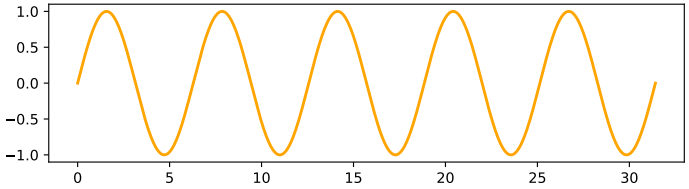
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

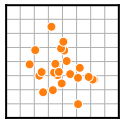
4 Observe



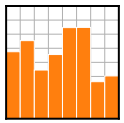
Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```

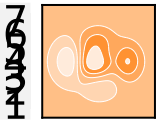


```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

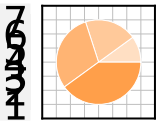


```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

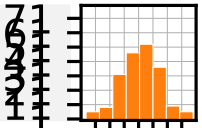
```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```



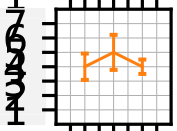
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```



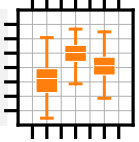
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0,1,5)
ax.errorbar(X, Y, Y/4)
```



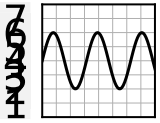
```
Z = np.random.normal(0,1,(100,3))

ax.boxplot(Z)
```



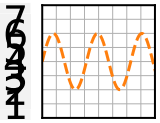
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

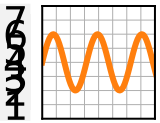
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



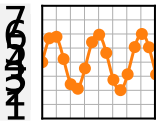
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



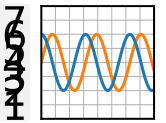
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



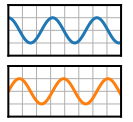
Organize

You can plot several data on the the same figure but you can also split a figure in several subplots (named *Axes*):

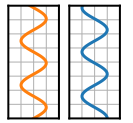
```
X = np.linspace(0,10,100)
Y1, Y1 = np.sin(X), np.cos(X)
ax.plot(X, Y1, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

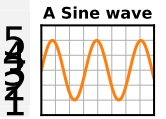


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

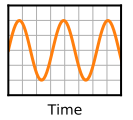


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

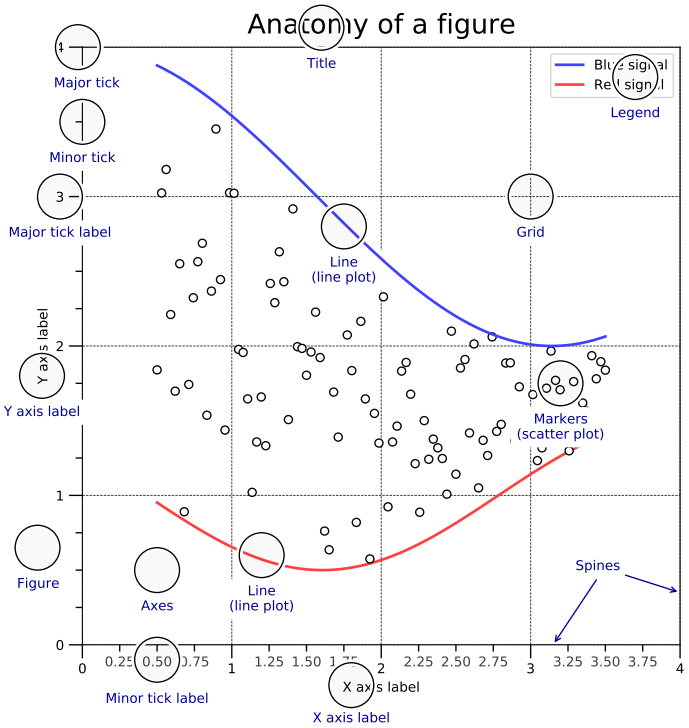
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

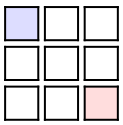
Matplotlib for intermediate users

A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.

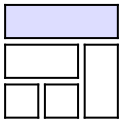


Figure, axes & spines

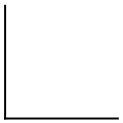
```
fig, axs = plt.subplots((3,3))
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```



```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```

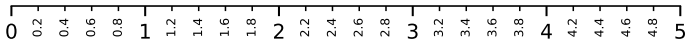


```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```



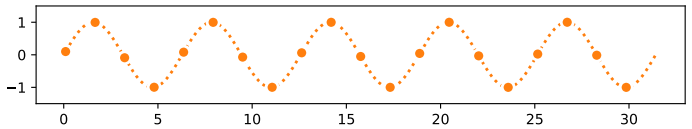
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



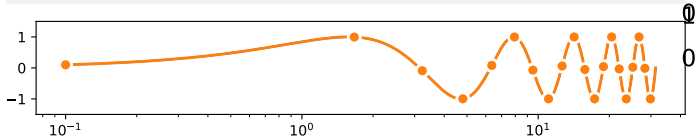
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



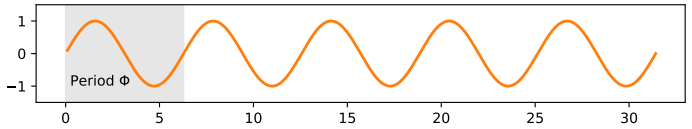
Scales & Projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



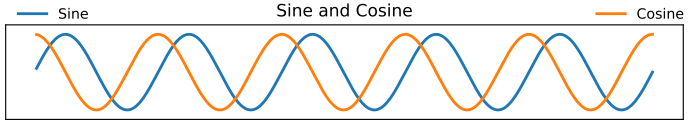
Text & Ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



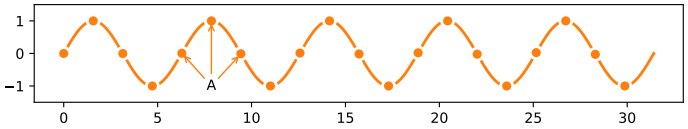
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



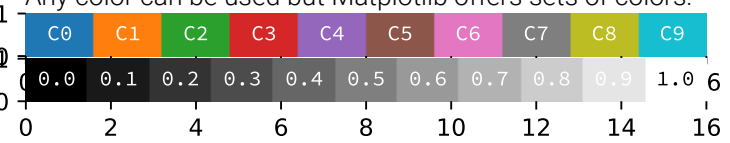
Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

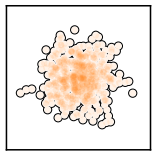
```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density and multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1,1,500)
Y = np.random.normal(-1,1,500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1",  lw=0, alpha=0.1)
```



Rasterization

If your figure is made of a lot graphical elements such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

???e rendering

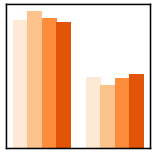
Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw som stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Blues")
colors = [cmap(i) for in in [.2,.4,.6,.8]]

ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
  fx.Stroke(linewidth=3, foreground='1.0'),
  fx.Normal()])
```

Multiline plot

You can plot several lines at once using None as separator.

```
X,Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
  X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```

Dotted lines

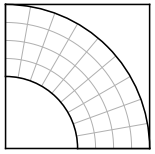
To have rounded dotted lines, use a custom `linestyle` and modify `dash_capstyle`.

```
ax.plot([0,1], [0,0], "C1",
        linestyle = (0, (0.01, 1)), dash_capstyle="round")
ax.plot([0,1], [1,1], "C1",
        linestyle = (0, (0.01, 2)), dash_capstyle="round")
```

Combining axes

You can use overlaid axes with different projections.

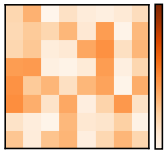
```
ax1 = fig.add_axes([0,0,1,1],
                   label="cartesian")
ax2 = fig.add_axes([0,0,1,1],
                   label="polar",
                   projection="polar")
```



Colorbar adjustment

You can adjust colorbar aspect when adding it.

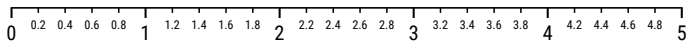
```
im = ax.imshow(Z)

cb = plt.colorbar(im,
        fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed face such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



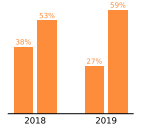
Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/" )
```



Read the documentation

Matplotlib comes with an extensive documenation explaining every detals of each command and is generally accompanied by examples with. Together with the huge online gallery, this documenation is a gold-mine.