



Author: [George McIntire](#)
Data Scientist



Author: [Brendan Martin](#)
Founder of LearnDataSci



Author: [Lauren Washington](#)
Lead Data Scientist & ML Developer

Python Pandas Tutorial: A Complete Introduction for Beginners

Learn some of the most important pandas features for exploring, cleaning, transforming, visualizing, and learning from data.

Contents Index

+

You should already know:

- Python fundamentals – learn interactively on [dataquest.io](#)

The *pandas* package is the most important tool at the disposal of Data Scientists and Analysts working in Python today. The powerful machine learning and glamorous visualization tools may get all the attention, but pandas is the backbone of most data projects.

include observational periods for the sc

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

If you're thinking about data science as a career, then it is imperative that one of the first things you do is learn pandas. In this post, we will go over the essential bits of information about pandas, including how to install it, its uses, and how it works with other common Python data analysis packages such as **matplotlib** and **scikit-learn**.

Article Resources

- iPython notebook and data [available on GitHub](#)

What's Pandas for?

Pandas has so many uses that it might make sense to list the things it can't do instead of what it can do.

This tool is essentially your data's home. Through pandas, you get acquainted with your data by cleaning, transforming, and analyzing it.

For example, say you want to explore a dataset stored in a CSV on your computer. Pandas will extract the data from that CSV into a DataFrame — a table, basically — then let you do things like:

- Calculate statistics and answer questions about the data, like
 - What's the average, median, max, or min of each column?
 - Does column A correlate with column B?
 - What does the distribution of data in column C look like?
- Clean the data by doing things like removing missing values and filtering rows or columns by some criteria
- Visualize the data with help from Matplotlib. Plot bars, lines, histograms, bubbles, and more.
- Store the cleaned, transformed data back into a CSV, other file or database

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

The rise in popularity of Pandas

1.0% of all question views on Stack Overflow*



ATLAS | Data: Stack Overflow | * World Bank high-income countries

How does pandas fit into the data science toolkit?

Not only is the pandas library a central component of the data science toolkit but it is used in conjunction with other libraries in that collection.

Pandas is built on top of the **NumPy** package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in **SciPy**, plotting functions from **Matplotlib**, and machine learning algorithms in **Scikit-learn**.

Jupyter Notebooks offer a good environment for using pandas to do data exploration and modeling, but pandas can also be used in text editors just as easily.

Jupyter Notebooks give us the ability to execute code in a particular cell as opposed to running the entire file. This saves a lot of time when working with large datasets and complex transformations. Notebooks also provide an easy way to visualize pandas'

When should you start learning pandas?

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

If you do not have any experience with Python, it's recommended you learn pandas until you do. You don't need to be a Python expert, but you should be adept at the basics, such as lists, tuples, dictionaries, functions, and iterations. Also, I'd also recommend familiarizing yourself with **NumPy** due to the similarities mentioned above.

If you're looking for a good place to learn Python, [Python for Everybody](#) on Coursera is great (and Free).

Moreover, for those of you looking to do a [data science bootcamp](#) or some other accelerated data science education program, it's highly recommended you start learning pandas on your own before you start the program.

Even though accelerated programs teach you pandas, better skills beforehand means you'll be able to maximize time for learning and mastering the more complicated material.

Pandas First Steps

Install and import

Pandas is an easy package to install. Open up your terminal program (for Mac users) or command line (for PC users) and install it using either of the following commands:

```
conda install pandas
```

OR

```
pip install pandas
```

Alternatively, if you're currently viewing this article in a Jupyter notebook you can run this cell:

```
!pip install pandas
```

Join over 3,500 data science enthusiasts.

```
import pandas as pd
```

Enter your email

Subscribe

Now to the basic components of pandas.

Core components of pandas: Series and DataFrames

The primary two components of pandas are the **Series** and **DataFrame**.

A **Series** is essentially a column, and a **DataFrame** is a multi-dimensional table made up of a collection of Series.

Series		Series		DataFrame	
	apples		oranges		apples
0	3	0	0	0	3
1	2	1	3	1	2
2	0	2	7	2	0
3	1	3	2	3	1

+ =

DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.

You'll see how these components work when we start working with data below.

Creating DataFrames from scratch

There are *many* ways to create a DataFrame from a simple `dict`.

Let's say we have a fruit stand that for each fruit and a row for each customer. With pandas we could do something like:

```
data = {  
    'apples': [3, 2, 0, 1],  
    'oranges': [0, 3, 7, 2]  
}
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

And then pass it to the pandas DataFrame constructor:

```
purchases = pd.DataFrame(data)
```

```
purchases
```

OUT:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

How did that work?

could also create our own when we want to.

Let's have customer names as our index:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
```

```
purchases
```

OUT:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

So now we could **locate** a customer's order by using their name:

```
purchases.loc['June']
```

OUT:

```
apples    3
oranges   0
Name: June, dtype: int64
```

There's more on locating and extracting data from the DataFrame later, but now you should be able to create a DataFrame with any random data to learn on.

Want to learn more?

See Best Data Science

[View](#)

Join over 3,500 data science enthusiasts.

Enter your email

[Subscribe](#)

How to read in data

It's quite simple to load data from various file formats into a DataFrame. In the following examples we'll keep using our apples and oranges data, but this time it's coming from various files.

Reading data from CSVs

With CSV files all you need is a single line to load in the data:

```
df = pd.read_csv('purchases.csv')
```

```
df
```

OUT:

	Unnamed: 0	apples	oranges
0	June	3	0
1	Robert	2	3
2	Lily	0	7
3	David	1	2

```
df = pd.read_csv('purchases.
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
df
```

OUT:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

Here we're setting the index to be column zero.

You'll find that most CSVs won't ever have an index column and so usually you don't have to worry about this step.

Reading data from JSON

If you have a JSON file — which is essentially a stored Python `dict` — pandas can read this just as easily:

```
df = pd.read_json('purchases.json')
```

```
df
```

Dav	Join over 3,500 data science enthusiasts.	
Jun	Enter your email	
Lily		
Robert	2	3

Notice this time our index came with us correctly since using JSON allowed indexes to work through nesting. Feel free to open `data_file.json` in a notepad so you can see how it works.

Pandas will try to figure out how to create a DataFrame by analyzing structure of your JSON, and sometimes it doesn't get it right. Often you'll need to set the `orient` keyword argument depending on the structure, so check out [read_json docs](#) about that argument to see which orientation you're using.

Reading data from a SQL database

If you're working with data from a SQL database you need to first establish a connection using an appropriate Python library, then pass a query to pandas. Here we'll use SQLite to demonstrate.

First, we need `pysqlite3` installed, so run this command in your terminal:

```
pip install pysqlite3
```

Or run this cell if you're in a notebook:

```
!pip install pysqlite3
```

`sqlite3` is used to create a connection to a database which we can then use to generate a DataFrame through a `SELECT` query.

So first we'll make a connection to a SQLite database file:

```
con = sqlite3.connect("datab
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

SQL TIP

If you have data in PostgreSQL, MySQL, or some other SQL server, you'll need to obtain the right Python library to make a connection. For example, [psycopg2](#) ([link](#)) is a commonly used library for making connections to PostgreSQL. Furthermore, you would make a connection to a database URI instead of a file like we did here with SQLite.

For a great course on SQL check out [The Complete SQL Bootcamp](#) on Udemy

In this SQLite database we have a table called *purchases*, and our index is in a column called "index".

By passing a SELECT query and our `con`, we can read from the *purchases* table:

```
df = pd.read_sql_query("SELECT * FROM purchases", con)
```

```
df
```

0		Join over 3,500 data science enthusiasts.		
1		Enter your email		Subscribe
2				
3	David	1	2	

Just like with CSVs, we could pass `index_col='index'`, but we can also set an index after-the-fact:

```
df = df.set_index('index')
```

```
df
```

OUT:

	apples	oranges
index		
June	3	0
Robert	2	3
Lily	0	7
David	1	2

In fact, we could use `set_index()` on *any* DataFrame using *any* column at *any* time. Indexing Series and DataFrames is a very common task, and the different ways of doing it is worth remembering.

Join over 3,500 data science enthusiasts.

```
df.to_csv('new_purchases.csv')
```

Enter your email

Subscribe

```
df.to_json('new_purchases.json')
```

```
df.to_sql('new_purchases', con)
```

When we save JSON and CSV files, all we have to input into those functions is our desired filename with the appropriate file extension. With SQL, we're not creating a new file but instead inserting a new table into the database using our `con` variable from before.

Let's move on to importing some real-world data and detailing a few of the operations you'll be using a lot.

Most important DataFrame operations

DataFrames possess hundreds of methods and other operations that are crucial to any analysis. As a beginner, you should know the operations that perform simple transformations of your data and those that provide fundamental statistical analysis.

Let's load in the IMDB movies dataset to begin:

```
movies_df = pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")
```

We're loading this dataset from a CSV and designating the movie titles to be our index.

Viewing your data

The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with `.head()`:



Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

Join over 3,500 data science enthusiasts.

Enter your email

[Subscribe](#)

Title						
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	- criminals are forced ...	James Gunn	Diese Coop	
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noon Logan Greene	
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	Jame Taylor Richards	
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matth McConaugh Witherspoon Mayer	
Suicide Squad	5	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith Leto, Viola Davis	

number as well: `movies_df.head(10)` v

To see the **last** five rows use `.tail()`
printing the bottom two rows.:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df.tail(2)
```

OUT:

	Rank	Genre	Description	Director	
Title					
Search Party	999	Adventure,Comedy	A pair of friends embark on a mission to reun...	Scot Armstrong	Adam Pally Miller, Tho Middleditc
Nine Lives	1000	Comedy,Family,Fantasy	A stuffy businessman finds himself trapped ins...	Barry Sonnenfeld	Kevin Spac Jennifer G Robbie Amell,Ch...

Typically when we load in a dataset, we like to view the first five or so rows to see what's under the hood. Here we can see the names of each column, the index, and examples of values in each row.

You'll notice that the index in our DataFrame is the *Title* column, which you can tell by how the word *Title* is slightly lower than the rest of the columns.

Getting info about your data

`movies_df.info()`

OUT:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
Rank           1000 non-null int64
Genre          1000 non-null object
Description    1000 non-null object
Director       1000 non-null object
Actors         1000 non-null object
Year           1000 non-null int64
Runtime (Minutes) 1000 non-null int64
Rating         1000 non-null float64
Votes          1000 non-null int64
Revenue (Millions) 872 non-null float64
Metascore      936 non-null float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

`.info()` provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

Notice in our movies dataset we have some obvious missing values in the `Revenue` and `Metascore` columns. We'll look at how to handle those in a bit.

Seeing the datatype quickly is actually quite useful. Imagine you just imported some JSON and the integers were recorded as strings. You go to do some arithmetic and find an "unsupported operand" Exception because you can't do math with strings. Calling `.info()`

Another fast and useful attribute is

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df.shape
```

OUT:

```
(1000, 11)
```

Note that `.shape` has no parentheses and is a simple tuple of format (rows, columns). So we have **1000 rows** and **11 columns** in our movies DataFrame.

You'll be going to `.shape` a lot when cleaning and transforming data. For example, you might filter some rows based on some criteria and then want to know quickly how many rows were removed.

Handling duplicates

This dataset does not have duplicate rows, but it is always important to verify you aren't aggregating duplicate rows.

To demonstrate, let's simply just double up our movies DataFrame by appending it to itself:

```
temp_df = movies_df.append(movies_df)
```

```
temp_df.shape
```

OUT:

```
(2000, 11)
```

Notice call `.shape` quickly proves c

Now we can try dropping duplicate

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
temp_df = temp_df.drop_duplicates()
```

```
temp_df.shape
```

OUT:

```
(1000, 11)
```

Just like `append()`, the `drop_duplicates()` method will also return a copy of your DataFrame, but this time with duplicates removed. Calling `.shape` confirms we're back to the 1000 rows of our original dataset.

It's a little verbose to keep assigning DataFrames to the same variable like in this example. For this reason, pandas has the `inplace` keyword argument on many of its methods. Using `inplace=True` will modify the DataFrame object in place:

```
temp_df.drop_duplicates(inplace=True)
```

Now our `temp_df` will have the transformed data automatically.

Another important argument for `drop_duplicates()` is `keep`, which has three possible options:

- `first`: (default) Drop duplicates except for the first occurrence.
- `last`: Drop duplicates except for the last occurrence.
- `False`: Drop all duplicates.

Since we didn't define the `keep` argument in the previous example it was defaulted to

be dropped. Watch what happens if we drop all the duplicates:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
temp_df = movies_df.append(movies_df)
```

```
temp_df.drop_duplicates(inplace=True, keep=False)
```

```
temp_df.shape
```

OUT:

```
(0, 11)
```

Since all rows were duplicates, `keep=False` dropped them all resulting in zero rows being left over. If you're wondering why you would want to do this, one reason is that it allows you to locate all duplicates in your dataset. When conditional selections are shown below you'll see how to do that.

Column cleanup

Many times datasets will have verbose column names with symbols, upper and lowercase words, spaces, and typos. To make selecting data by column name easier we can spend a little time cleaning up their names.

Here's how to print the column names of our dataset:

```
movies_df.columns
```

OUT:

```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',  
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
```

Not only does `.columns` come in handy for simple copy and paste, it's also useful for avoiding `Key Error` when selecting data by column name.

We can use the `.rename()` method to rename certain or all columns via a `dict`. We don't want parentheses, so let's rename those:

```
movies_df.rename(columns={  
    'Runtime (Minutes)': 'Runtime',  
    'Revenue (Millions)': 'Revenue_millions'  
}, inplace=True)  
  
movies_df.columns
```

OUT:

```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year', 'Runtime',  
       'Rating', 'Votes', 'Revenue_millions', 'Metascore'],  
      dtype='object')
```

Excellent. But what if we want to lowercase all names? Instead of using `.rename()` we could also set a list of names to the columns like so:

```
movies_df.columns = ['rank', 'genre', 'description', 'director', 'actors', 'year',  
                     'rating', 'votes', 'revenue_millions', 'metascore']
```

OUT:

```
Index(['rank', 'genre', 'des  
      'rating', 'votes', 'revenue_millions', 'metascore'],  
      dtype='object')
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

But that's too much work. Instead of just renaming each column manually we can do a list comprehension:

```
movies_df.columns = [col.lower() for col in movies_df]
```

```
movies_df.columns
```

OUT:

```
Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime  
      'rating', 'votes', 'revenue_millions', 'metascore'],  
      dtype='object')
```

`list` (and `dict`) comprehensions come in handy a lot when working with pandas and data in general.

It's a good idea to lowercase, remove special characters, and replace spaces with underscores if you'll be working with a dataset for some time.

How to work with missing values

When exploring data, you'll most likely encounter missing or null values, which are essentially placeholders for non-existent values. Most commonly you'll see Python's `None` or NumPy's `np.nan`, each of which are handled differently in some situations.

2. Replace nulls with non-null values

Let's calculate the total number of nulls in each column and check which cells in our DataFrame are null.

Join over 3,500 data science enthusiasts.

Enter your email

[Subscribe](#)

```
movies_df.isnull()
```

OUT:

	rank	genre	description	director	actors	year	runtime
Title							
Guardians							
of the	False	False	False	False	False	False	False
Galaxy							
Prometheus	False	False	False	False	False	False	False
Split	False	False	False	False	False	False	False
Sing	False	False	False	False	False	False	False
Suicide							
Squad	False	False	False	False	False	False	False

Notice `isnull()` returns a DataFrame where each cell is either True or False depending on that cell's null status.

To count the number of nulls in each column we use an aggregate function for summing:

```
movies_df.isnull().sum()
```

```
8 rows × 8 columns
description      0
director        0
actors          0
year            0
runtime         0
rating          0
votes           0
revenue_millions 128
metascore       64
dtype: int64
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

`.isnull()` just by itself isn't very useful, and is usually used in conjunction with other methods, like `sum()`.

We can see now that our data has **128** missing values for `revenue_millions` and **64** missing values for `metascore`.

Removing null values

Data Scientists and Analysts regularly face the dilemma of dropping or imputing null values, and is a decision that requires intimate knowledge of your data and its context. Overall, removing null data is only suggested if you have a small amount of missing data.

Remove nulls is pretty simple:

```
movies_df.dropna()
```

This operation will delete any **row** with at least a single null value, but it will return a new DataFrame without altering the original one. You could specify `inplace=True` in this method as well.

Waste since there's perfectly good data.

That's why we'll look at imputation.

Other than just dropping rows, you

`axis=1`:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df.dropna(axis=1)
```

In our dataset, this operation would drop the `revenue_millions` and `metascore` columns.

INTUITION

What's with this `axis=1` parameter?

It's not immediately obvious where `axis` comes from and why you need it to be 1 for it to affect columns. To see why, just look at the `.shape` output:

```
movies_df.shape
```

```
Out: (1000, 11)
```

As we learned above, this is a tuple that represents the shape of the DataFrame, i.e. 1000 rows and 11 columns. Note that the *rows* are at index zero of this tuple and *columns* are at **index one** of this tuple. This is why `axis=1` affects columns. This comes from NumPy, and is a great example of why learning NumPy is worth your time.

Imputation

Imputation is a conventional feature engineering technique used to keep valuable data that have null values.

There may be instances where dropping every row with a null value removes too big a

that column into its own variable:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
revenue = movies_df[ 'revenue'
```

Using square brackets is the general way we select columns in a DataFrame.

If you remember back to when we created DataFrames from scratch, the keys of the `dict` ended up as column names. Now when we select columns of a DataFrame, we use brackets just like if we were accessing a Python dictionary.

`revenue` now contains a Series:

```
revenue.head()
```

OUT:

```
Title
Guardians of the Galaxy    333.13
Prometheus                 126.46
Split                       138.12
Sing                         270.32
Suicide Squad                325.02
Name: revenue_millions, dtype: float64
```

Slightly different formatting than a DataFrame, but we still have our `Title` index.

We'll impute the missing values of revenue using the mean. Here's the mean value:

```
revenue_mean = revenue.mean()
```

OUT:

```
82.95637614678897
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

With the mean, let's fill the nulls using `fillna()`.

```
revenue.fillna(revenue_mean, inplace=True)
```

We have now replaced all nulls in `revenue` with the mean of the column. Notice that by using `inplace=True` we have actually affected the original `movies_df`:

```
movies_df.isnull().sum()
```

OUT:

```
rank          0
genre         0
description   0
director      0
actors        0
year          0
runtime       0
rating        0
votes          0
revenue_millions  0
metascore     64
dtype: int64
```

For example, you would find the mean for each column individually and impute the nulls in each column.

Let's now look at more ways to examine your variables.

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

Understanding your variables

Using `describe()` on an entire DataFrame we can get a summary of the distribution of continuous variables:

```
movies_df.describe()
```

OUT:

	rank	year	runtime	rating	
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000
mean	500.500000	2012.783000	113.172000	6.723200	1.698083
std	288.819436	3.205962	18.810908	0.945429	1.887626
min	1.000000	2006.000000	66.000000	1.900000	6.100000
25%	250.750000	2010.000000	100.000000	6.200000	3.630900
50%	500.500000	2014.000000	111.000000	6.800000	1.107990
75%	750.250000	2016.000000	123.000000	7.400000	2.399090
max	1000.000000	2016.000000	191.000000	9.000000	1.791916e+000

Understanding which numbers are continuous also comes in handy when thinking about the type of plot to use to represent your data visually.

```
movies_df['genre'].describe()
```

OUT:

```
count          1000
unique         207
top    Action, Adventure, Sci-Fi
freq            50
Name: genre, dtype: object
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

This tells us that the genre column has 207 unique values, the top value is Action/Adventure/Sci-Fi, which shows up 50 times (freq).

`.value_counts()` can tell us the frequency of all values in a column:

```
movies_df['genre'].value_counts().head(10)
```

OUT:

```
Action, Adventure, Sci-Fi      50
Drama                          48
Comedy, Drama, Romance          35
Comedy                         32
Drama, Romance                  31
Action, Adventure, Fantasy     27
Comedy, Drama                   27
Animation, Adventure, Comedy   27
Comedy, Romance                 26
Crime, Drama, Thriller         24
Name: genre, dtype: int64
```

By using the correlation method `corr()` on a continuous variable:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df.corr()
```

	rank	year	runtime	rating	votes	r
rank	1.000000	-0.261605	-0.221739	-0.219555	-0.283876	
year	-0.261605	1.000000	-0.164900	-0.211219	-0.411904	
runtime	-0.221739	-0.164900	1.000000	0.392214	0.407062	
rating	-0.219555	-0.211219	0.392214	1.000000	0.511537	
votes	-0.283876	-0.411904	0.407062	0.511537	1.000000	
revenue_millions	-0.252996	-0.117562	0.247834	0.189527	0.607941	
metascore	-0.191869	-0.079305	0.211978	0.631897	0.325684	

Correlation tables are a numerical representation of the bivariate relationships in the dataset.

Positive numbers indicate a positive correlation — one goes up the other goes up — and negative numbers represent an inverse correlation — one goes up the other goes down. 1.0 indicates a perfect correlation.

So looking in the first row, first column we see `rank` has a perfect correlation with itself, which is obvious. On the other hand, the correlation between `votes` and `revenue_millions` is 0.6. A little more interesting.

Examining bivariate relationships comes in handy when you have an outcome or dependent variable in mind and would like to see the features most correlated to the increase or decrease of the outcome. You can visually represent bivariate relationships

Let's now look more at manipulation

Join over 3,500 data science enthusiasts.

DataFrame slicing, selecting

Enter your email

Subscribe

Up until now we've focused on some basic summaries of our data. We've learned about simple column extraction using single brackets, and we imputed null values in a column using `fillna()`. Below are the other methods of slicing, selecting, and extracting you'll need to use constantly.

It's important to note that, although many methods are the same, `DataFrames` and `Series` have different attributes, so you'll need be sure to know which type you are working with or else you will receive attribute errors.

Let's look at working with columns first.

By column

You already saw how to extract a column using square brackets like this:

```
genre_col = movies_df['genre']

type(genre_col)
```

OUT:

```
pandas.core.series.Series
```

This will return a `Series`. To extract a column as a `DataFrame`, you need to pass a list of column names. In our case that's just a single column:

```
genre_col = movies_df[['genre']]

type(genre_col)
```

pandas.core.frame.DataFrame

Since it's just a list, adding another

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
subset = movies_df[['genre', 'rating']]
```

```
subset.head()
```

OUT:

	genre	rating
Title		
Guardians of the Galaxy	Action,Adventure,Sci-Fi	8.1
Prometheus	Adventure,Mystery,Sci-Fi	7.0
Split	Horror,Thriller	7.3
Sing	Animation,Comedy,Family	7.2
Suicide Squad	Action,Adventure,Fantasy	6.2

Now we'll look at getting data by rows.

By rows

For rows, we have two options:

- `.loc` - locates by name
- `.iloc` - locates by numerical index

```
prom = movies_df.loc["Promet
```

```
prom
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

OUT:

```
rank                                2
genre                               Adventure,Mystery,Sci-Fi
description             Following clues to the origin of mankind, a te...
director                            Ridley Scott
actors                             Noomi Rapace, Logan Marshall-Green, Michael Fa...
year                                2012
runtime                             124
rating                               7
votes                                485820
revenue_millions                     126.46
metascore                           65
Name: Prometheus, dtype: object
```

On the other hand, with `iloc` we give it the numerical index of Prometheus:

```
prom = movies_df.iloc[1]
```

`loc` and `iloc` can be thought of as similar to Python `list` slicing. To show this even further, let's select multiple rows.

How would you do it with a list? In Python, just slice with brackets like `example_list[1:4]`. It's works the same way in pandas:

```
movie_subset = movies_df.iloc
```

Join over 3,500 data science enthusiasts.

Enter your email

[Subscribe](#)

```
movie_subset
```

OUT:

	rank	genre	description	director	
Title					
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Logan Green,
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James Taylor- Richard...
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaugh Witherspoon Maggie... Ma...

One important distinction between using `.loc` and `.iloc` to select multiple rows is that

end is not included.

Join over 3,500 data science enthusiasts.

Conditional selections

Enter your email

Subscribe

We've gone over how to select columns and rows, but what about conditional selection?

For example, what if we want to filter our movies DataFrame to show only films directed by Ridley Scott or films with a rating greater than or equal to 8.0?

To do that, we take a column from the DataFrame and apply a Boolean condition to it. Here's an example of a Boolean condition:

```
condition = (movies_df['director'] == "Ridley Scott")  
  
condition.head()
```

OUT:

```
Title  
Guardians of the Galaxy    False  
Prometheus                  True  
Split                        False  
Sing                          False  
Suicide Squad                False  
  
Name: director, dtype: bool
```

Similar to `isnull()`, this returns a Series of True and False values: True for films directed by Ridley Scott and False for ones not directed by him.

We want to filter out all movies not directed by Ridley Scott, in other words, we don't want the False films. To return the rows where that condition is True we have to pass this operation into the DataFrame:

OUT:

Join over 3,500 data science enthusiasts.

rank

Enter your email

Subscribe

Title						
	rank		genre	description bring d...	director	actors
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael F...	
The Martian	103	Adventure,Drama,Sci-Fi	An astronaut becomes stranded on Mars after hi...	Ridley Scott	Matt Damon, Jessica Chastain, Kristen Wiig, Ka...	
Robin Hood	388	Action,Adventure,Drama	In 12th century England, Robin and his band of...	Ridley Scott	Russell Crowe, Cate Blanchett, Matthew Macfady...	
American Gangster	471	Biography,Crime,Drama	In 1970s America, a detective works to bring d...	Ridley Scott	Denzel Washington, Russell Crowe, Chiwetel Ej...	

Gods and
Kings

517

Action,A

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

You can get used to looking at these conditionals by reading it like:

Select `movies_df` where `movies_df[director]` equals Ridley Scott.

Let's look at conditional selections using numerical values by filtering the DataFrame by ratings:

```
movies_df[movies_df['rating'] >= 8.6].head(3)
```

Join over 3,500 data science enthusiasts.					
			Enter your email	Subscribe	
Interstellar	37	Adventure, Fi	through a wormhole ...	Nolan	Anne Hathaway, Jessica Ch...
The Dark Knight	55	Action,Crime,Drama	When the menace known as the Joker wreaks havo...	Christopher Nolan	Christian Bal Heath Ledger Aaron Eckhart,Mi...
Inception	81	Action,Adventure,Sci- Fi	A thief, who steals corporate secrets through ...	Christopher Nolan	Leonardo DiCaprio, Joseph Gordon-Levi Ellen...

We can make some richer conditionals by using logical operators `|` for "or" and `&` for "and".

Let's filter the DataFrame to show only movies by Christopher Nolan OR Ridley Scott:

```
movies_df[(movies_df['director'] == 'Christopher Nolan') | (movies_df['director'] == 'Ridley Scott')]
```

Join over 3,500 data science enthusiasts.					
		Enter your email		Subscribe	
Prometheus	2	Adventure,Sci-Fi	mankind, a team...	Scott	Green, Michael
Interstellar	37	Adventure,Drama,Sci-Fi	A team of explorers travel through a wormhole ...	Christopher Nolan	Matthew McCona Anne Hathawa Jessica C
The Dark Knight	55	Action,Crime,Drama	When the menace known as the Joker wreaks havoc...	Christopher Nolan	Christian Heath Le Aaron Eckhart,N
The Prestige	65	Drama,Mystery,Sci-Fi	Two stage magicians engage in competitive one-...	Christopher Nolan	Christian Hugh Jackmar Scarlett Johanss.
Inception	81	Action,Adventure,Sci-Fi	A thief, who steals corporate secrets through ...	Christopher Nolan	Leonardo DiCaprio Joseph Gordon-Ellen...



evaluate the conditional.

Using the `isin()` method we could

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df[movies_df['director'].isin(['Christopher Nolan', 'Ridley Scott'])].|
```

Join over 3,500 data science enthusiasts.					
			Enter your email	Subscribe	
Prometheus	2	Adventure,Sci-Fi	mankind, a team...	Scott	Green, Michael
Interstellar	37	Adventure,Drama,Sci-Fi	A team of explorers travel through a wormhole ...	Christopher Nolan	Matthew McCona Anne Hathawa Jessica C
The Dark Knight	55	Action,Crime,Drama	When the menace known as the Joker wreaks havoc...	Christopher Nolan	Christian Heath Le Aaron Eckhart,N
The Prestige	65	Drama,Mystery,Sci-Fi	Two stage magicians engage in competitive one-...	Christopher Nolan	Christian Hugh Jackmar Scarlett Johanss.
Inception	81	Action,Adventure,Sci-Fi	A thief, who steals corporate secrets through ...	Christopher Nolan	Leonardo DiCaprio Joseph Gordon-Ellen...

above 8.0, but made below the 25t

Here's how we could do all of that:

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
movies_df[  
  ((movies_df['year'] >= 2005) & (movies_df['year'] <= 2010))  
  & (movies_df['rating'] > 8.0)  
  & (movies_df['revenue_millions'] < movies_df['revenue_millions'].quantile  
]
```

Title

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

3 Idiots	431	Comedy,Drama	searching for their long lost ...	Rajkumar Hirani	Madhavan, I Singh, Sharr Joshi
The Lives of Others	477	Drama,Thriller	In 1984 East Berlin, an agent of the secret po...	Florian Henckel von Donnersmarck	Ulrich Mühe Martina Gedeck,Seb Koch, Ul...
Incendies	714	Drama,Mystery,War	Twins journey to the Middle East to discover t...	Denis Villeneuve	Lubna Azab Mélissa Désormeaux Poulin, Maxi
Taare Zameen Par	992	Drama,Family,Music	An eight- year-old boy is thought to be a lazy ...	Aamir Khan	Darsheel Sa Aamir Khan, Tanay Chhe Sac...



So here we have only four movies!

Join over 3,500 data science enthusiasts.

Applying functions

Enter your email

Subscribe

It is possible to iterate over a DataFrame or Series as you would with a list, but doing so — especially on large datasets — is very slow.

An efficient alternative is to `apply()` a function to the dataset. For example, we could use a function to convert movies with an 8.0 or greater to a string value of "good" and the rest to "bad" and use this transformed values to create a new column.

First we would create a function that, when given a rating, determines if it's good or bad:

```
def rating_function(x):
    if x >= 8.0:
        return "good"
    else:
        return "bad"
```

Now we want to send the entire rating column through this function, which is what `apply()` does:

```
movies_df["rating_category"] = movies_df["rating"].apply(rating_function)
```

```
movies_df.head(2)
```

Join over 3,500 data science enthusiasts.						
			Enter your email	Subscribe		
Guardians of the Galaxy	1	Action,Ac Fi	criminals are forced ...	Gunn	Bradley Cooper, Zoe S...	
Prometheus	2	Adventure,Mystery,Sci- Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall- Green, Michael Fa...	

The `.apply()` method passes every value in the `rating` column through the `rating_function` and then returns a new Series. This Series is then assigned to a new column called `rating_category`.

You can also use anonymous functions as well. This lambda function achieves the same result as `rating_function`:

```
movies_df["rating_category"] = movies_df["rating"].apply(lambda x: 'good' if x >= 7.5 else 'bad')

movies_df.head(2)
```

Join over 3,500 data science enthusiasts.						
			Enter your email	Subscribe		
Guardians of the Galaxy	1	Action,Ac Fi	criminals are forced ...	Gunn	Bradley Cooper, Zoe S...	
Prometheus	2	Adventure,Mystery,Sci- Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall- Green, Michael Fa...	

Overall, using `apply()` will be much faster than iterating manually over rows because pandas is utilizing vectorization.

Vectorization: a style of computer programming where operations are applied to whole arrays instead of individual elements —[Wikipedia](#)

A good example of high usage of `apply()` is during natural language processing (NLP) work. You'll need to apply all sorts of text cleaning functions to strings to prepare for machine learning.

Another great thing about pandas is that you can plot directly off DataFrames and run (`pip install matplotlib`):

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

```
import matplotlib.pyplot as plt  
plt.rcParams.update({'font.size': 20, 'figure.figsize': (10, 8)}) # set font size
```

Now we can begin. There won't be a lot of coverage on plotting, but it should be enough to explore your data easily.

PLOTTING TIP

For categorical variables utilize Bar Charts* and Boxplots.

For continuous variables utilize Histograms, Scatterplots, Line graphs, and Boxplots.

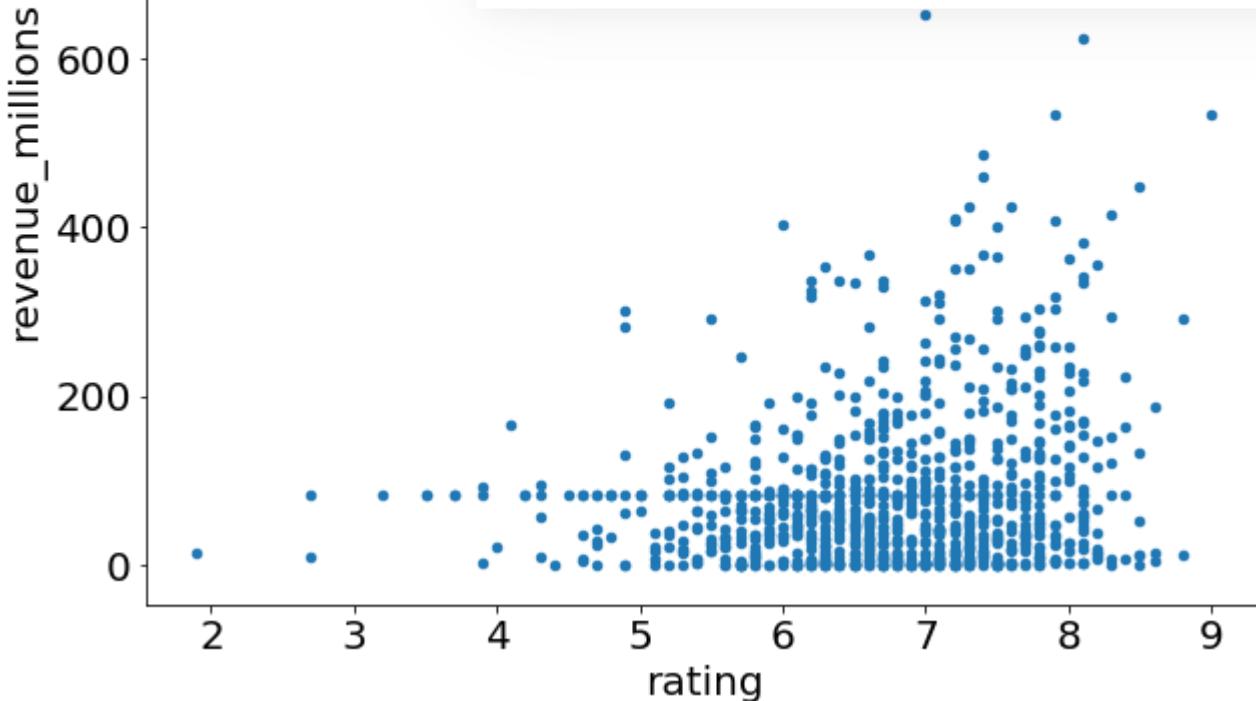
Let's plot the relationship between ratings and revenue. All we need to do is call `.plot()` on `movies_df` with some info about how to construct the plot:

```
movies_df.plot(kind='scatter', x='rating', y='revenue_millions', title='Revenue vs Rating')
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe



What's with the semicolon? It's not a syntax error, just a way to hide the
`<matplotlib.axes._subplots.AxesSubplot at 0x26613b5cc18>` output when plotting in Jupyter notebooks.

If we want to plot a simple Histogram based on a single column, we can call `plot` on a column:

```
movies_df['rating'].plot(kind='hist', title='Rating');
```

Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

Frequency

250

200

150

100

50

0

2

3

4

5

6

7

8

9

Do you remember the `.describe()` example at the beginning of this tutorial? Well, there's a graphical representation of the interquartile range, called the Boxplot. Let's recall what `describe()` gives us on the ratings column:

```
movies_df['rating'].describe()
```

OUT:

count	1000.000000
mean	6.723200
std	0.945429
min	1.900000
25%	6.200000

Name: rating, dtype: float64

Join over 3,500 data science enthusiasts.

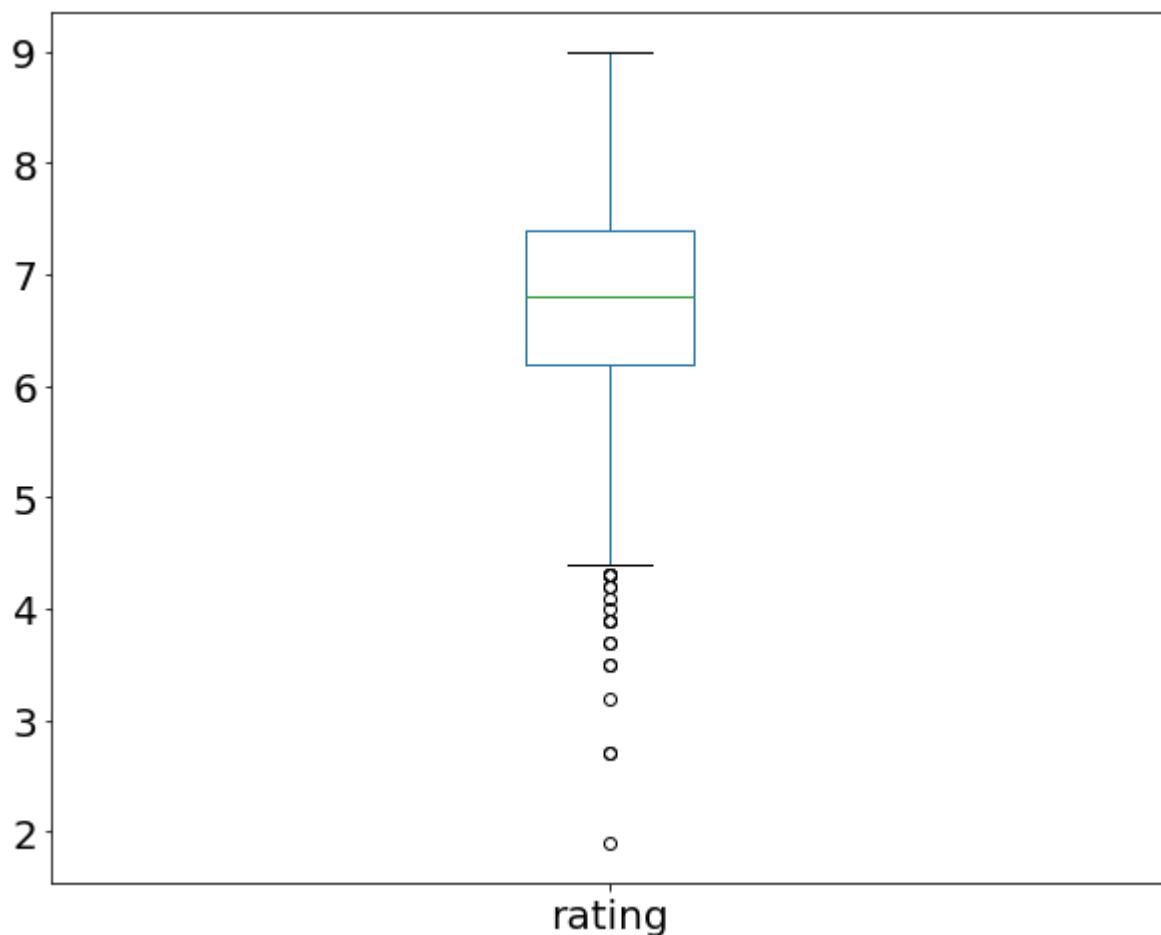
Enter your email

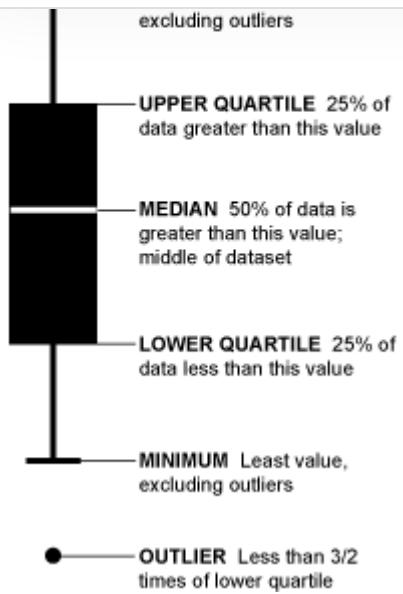
Subscribe

Using a Boxplot we can visualize th

```
movies_df['rating'].plot(kind="box");
```

RESULT:





Source: *Flowing Data*

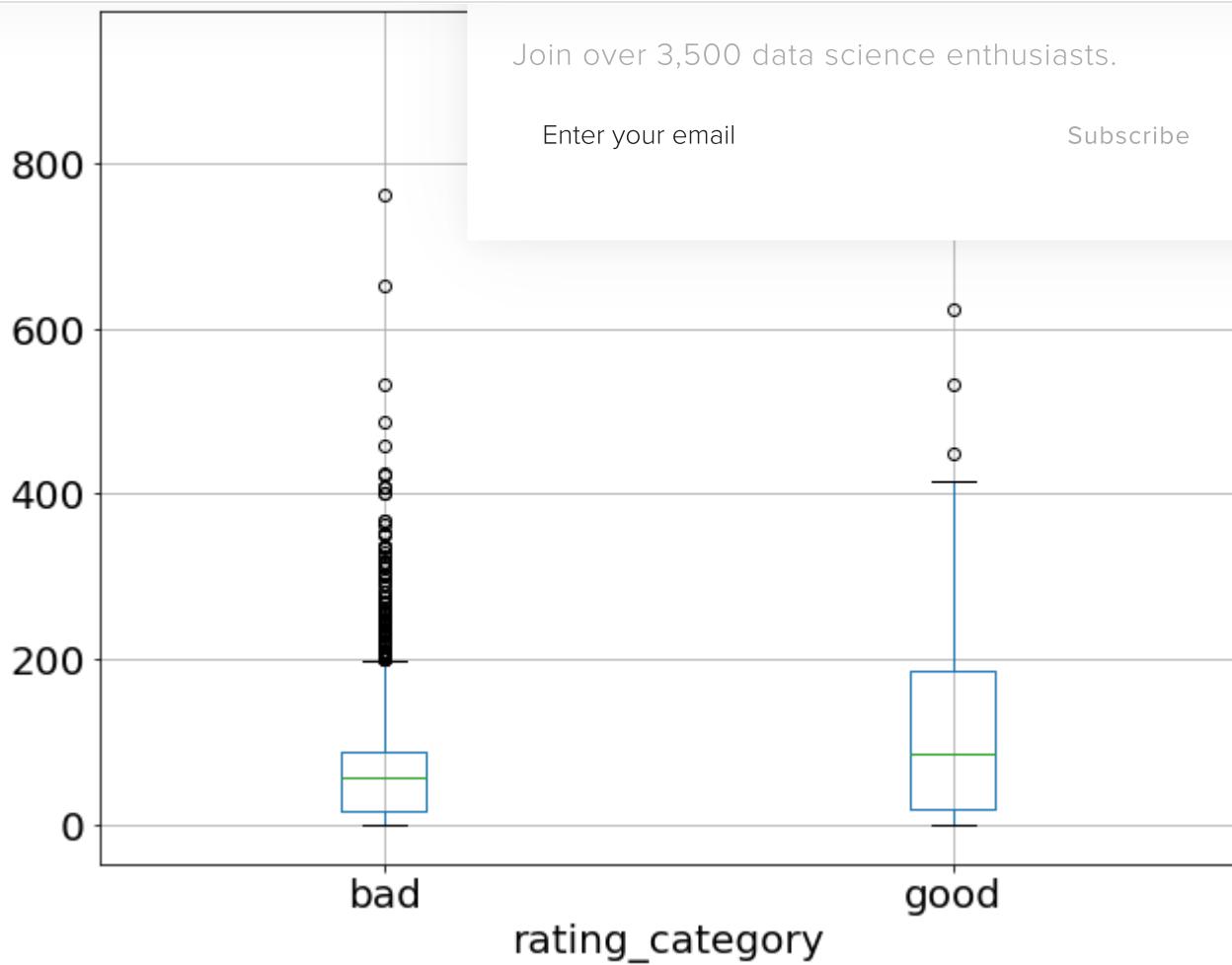
Join over 3,500 data science enthusiasts.

Enter your email

Subscribe

By combining categorical and continuous data, we can create a Boxplot of revenue that is grouped by the Rating Category we created above:

```
movies_df.boxplot(column='revenue_millions', by='rating_category');
```



That's the general idea of plotting with pandas. There's too many plots to mention, so definitely take a look at the [plot\(\)](#) [docs here](#) for more information on what it can do.

Wrapping up

Exploring, cleaning, transforming, and visualization data with pandas in Python is an essential skill in data science. Just cleaning wrangling data is 80% of your job as a Data Scientist. After a few projects and some practice, you should be very comfortable with most of the basics.

To keep improving, view the [extensive tutorials](#) offered by the official pandas docs, follow along with a few [Kaggle kernels](#), and keep working on your own projects!