Pandas Tutorial: Analyzing Video Game Data with Python and Pandas

August 26, 2019

Python's pandas library is one of the things that makes Python a great programming language for data analysis. Pandas makes importing, analyzing, and visualizing data much easier. It builds on packages like NumPy and matplotlib to give you a single, convenient, place to do most of your data analysis and visualization work.

In this Python data science tutorial, we'll use Pandas to analyze video game reviews from IGN, a popular video game review site, using data scraped by Eric Grinstein. Which console is winning the "console wars" (in the sense of having better-reviewed games)? This data set will help us find out.

As we analyze the video game reviews, we'll learn about key pandas concepts like indexing. You can follow this up and learn more about Python and pandas in one of our many other Python tutorials, or by enrolling in our Python Pandas course. Many of our other data science courses also use pandas.

Just as a note, this tutorial was written using Python 3.5 and built using Jupyter Notebook. You're likely using a more updated versions of Python, pandas, and Jupyter, but your results should be essentially the same.

Importing Data with Pandas

If you're working along with this tutorial, you'll need to download the data set, which you can do here.

The first step we'll take is to read the data in. The data is stored as a comma-separated values, or csv, file, where each row is separated by a new line, and each column by a comma (`,`). Here are the first few rows of the `ign.csv` file:

```
,score_     ,title,url,platform,score,genre,editors_choice,release_year,release_month,release_day
0,Amazi     tleBigPlanet PS Vita,/games/littlebigplanet-vita/vita-98907,PlayStation Vita,9.0,Platformer,Y,2012,9,12
1,Amazi     tleBigPlanet PS Vita -- Marvel Super Hero Edition,/games/littlebigplanet-ps-vita-marvel-super-hero-edition/vita-
2002705     Station Vita,9.0,Platformer,Y,2012,9,12
2,Great,Splice: Tree of Life,/games/splice/ipad-141070,iPad,8.5,Puzzle,N,2012,9,12
3,Great,NHL 13,/games/nhl-13/xbox-360-128182,Xbox 360,8.5,Sports,N,2012,9,11
```

As you can see above, each row in the file represents a single game that was reviewed by IGN. The columns contain information about that game:

- `score_phrase` — how IGN described the game in one word. This is linked to the score it received.
- `title` — the name of the game.
- `url` — the URL where you can see the full review.
- `platform` — the platform the game was reviewed on (PC, PS4, etc).
- `score` — the score for the game, from `1.0` to `10.0`.
- `genre` — the genre of the game.
- `editors_choice` — `N` if the game wasn't an editor's choice, `Y` if it was. This is tied to score.
- `release_year` — the year the game was released.
- `release_month` — the month the game was released.
- `release_day` — the day the game was released.

There's also a leading column that contains row index values. We can safely ignore this column, but we'll dive into what index values are later on.

To work data effectively in Python and pandas, we'll need to read the csv file into a Pandas DataFrame. A DataFrame is a way to represent and work with tabular data — data that's in table form, like a spreadsheet. Tabular data has rows and columns, just like our csv file, but it'll be easier for us to read and sort through if we can view it as a table.

In order to read in the data, we'll need to use the pandas.read_csv function. This function will take in a csv file and return a DataFrame. The below code will:

- Import the `pandas` library. We rename it to `pd` so it's faster to type out. This is a standard convention in data analysis and data science, and you will often see pandas imported as `pd` in other people's code.
- Read `ign.csv` into a DataFrame, and assign the result to a new variable called `reviews` so that we can use `reviews` to refer to our data.

```
import p        as pd
reviews          ead_csv("ign.csv")
```

Once we re   a DataFrame, it's helpful to take a look at what we've got in a more visual way. Conveniently, Pandas gives us two methods that make   to print out the data a table. These functions are:

- DataFrame.head() — prints the first N rows of a DataFrame, where N is a number you pass as an argument to the function, i.e. `DataFrame.head(7)`. If you don't pass any argument, the default is 5.
- DataFrame.tail() — prints the last N rows of a DataFrame. Again, the default is `5`.

We'll use the `head` method to see what's in `reviews`:

```
reviews.head()
```

	Unnamed: 0	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	releas
0	0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y	2012	9	12
1	1	Amazing	LittleBigPlanet PS Vita — Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y	2012	9	12
2	2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N	2012	9	12
3	3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N	2012	9	11
4	4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N	2012	9	11

We can also access the pandas.DataFrame.shape property to see row many rows and columns are in `reviews`:

```
reviews.shape
```

```
(18625, 11)
```

As we can see, everything has been read in properly — we have `18,625` rows and `11` columns.

One of the lvantages of using Pandas over a similar Python package like NumPy is that Pandas allows us to have columns with different d. es. In our data set, `reviews`, we have columns that store float values like `score`, string values like `score_phrase`, and integers lik ease_year`, so using NumPy here would be difficult, but Pandas and Python handle it well.

Now that w ead the data in properly, let's work on indexing `reviews` to get the rows and columns that we want.

Indexing DataFrames with Pandas

Earlier, we used the `head` method to print the first `5` rows of `reviews`. We could accomplish the same thing using the pandas.DataFrame.iloc method. The `iloc` method allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The below code will replicate the results of our `reviews.head()` by selecting rows zero to five, and all of the columns in our data set:

```
reviews.iloc[0:5,:]
```

	Unnamed: 0	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	releas
0	0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y	2012	9	12
1	1	Amazing	LittleBigPlanet PS Vita — Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y	2012	9	12
2	2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N	2012	9	12
3	3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N	2012	9	11
4	4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N	2012	9	11

Let's dig in a little deeper into our code: we specified that we wanted rows `0:5`. This means that we wanted the rows from position `0` up to, but not including, position `5`.

The first row is considered to be in position `0`, so selecting rows `0:5` gives us the rows at positions `0`, `1`, `2`, `3`, and `4`. We wanted all of the columns, too, and we used a shortcut to select them. It works like this: if we leave off the first position value, like `:5`, it's assumed we mean `0`. If we leave off the last position value, like `0:`, it's assumed we mean the last row or column in the DataFrame. We wanted all of the columr - -- we specified just a colon (`:`), without any positions. This gave us the columns from `0` to the last column. Here are some indexing ex es, along with the results:

- `reviews [:5,:]` — the first `5` rows, and all of the columns for those rows.
- `reviews :[:,:]` — the entire DataFrame.
- `reviews :[5:,5:]` — rows from position `5` onwards, and columns from position `5` onwards.
- `reviews.iloc[:,0]` — the first column, and all of the rows for the column.
- `reviews.iloc[9,:]` — the 10th row, and all of the columns for that row.

Indexing by position is very similar to NumPy indexing. If you want to learn more, you can read our NumPy tutorial. Now that we know how to index by position, let's remove the first column, which doesn't have any useful information:

```
reviews = reviews.iloc[:,1:]
reviews.head()
```

	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day
0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y	2012	9	12
1	Amazing	LittleBigPlanet PS Vita — Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y	2012	9	12
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N	2012	9	12
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N	2012	9	11
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N	2012	9	11

Indexing Using Labels in Python with Pandas

Now that we know how to retrieve rows and columns by position, it's worth looking into the other major way to work with DataFrames, which is to retrieve rows and columns by label. A major advantage of Pandas over NumPy is that each of the columns and rows has a label. Working with column positions is possible, but it can be hard to keep track of which number corresponds to which column.

We can work with labels using the pandas.DataFrame.loc method, which allows us to index using labels instead of positions. We can display the first five rows of `reviews` using the `loc` method like this:

```
reviews.loc[0:5,:]
```

	score_ e	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day
0	Amazin	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y	2012	9	12
1	Amazing	LittleBigPlanet PS Vita — Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y	2012	9	12
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N	2012	9	12
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N	2012	9	11
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N	2012	9	11
5	Good	Total War Battles: Shogun	/games/total-war-battles-shogun/mac-142565	Macintosh	7.0	Strategy	N	2012	9	11

The above doesn't actually look much different from `reviews.iloc[0:5,:]`. This is because while row labels can take on any values, our row labels match the positions exactly. You can see the row labels on the very left of the table above (they're in bold). You can also see them by accessing the index property of a DataFrame. We'll display the row indexes for `reviews`:

```
reviews.index
```

```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
```

Indexes don't always have to match up with positions, though. In the below code cell, we'll:

- Get row `10` to row `20` of `reviews`, and assign the result to `some_reviews`.
- Display the first `5` rows of `some_reviews`.

```
some_reviews = reviews.iloc[10:20,]
some_reviews.head()
```

	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day

	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day
10	G	Tekken Tag Tournament 2	/games/tekken-tag-tournament-2/ps3-124584	PlayStation 3	7.5	Fighting	N	2012	9	11
11	Good	Tekken Tag Tournament 2	/games/tekken-tag-tournament-2/xbox-360-124581	Xbox 360	7.5	Fighting	N	2012	9	11
12	Good	Wild Blood	/games/wild-blood/iphone-139363	iPhone	7.0	NaN	N	2012	9	10
13	Amazing	Mark of the Ninja	/games/mark-of-the-ninja-135615/xbox-360-129276	Xbox 360	9.0	Action, Adventure	Y	2012	9	7
14	Amazing	Mark of the Ninja	/games/mark-of-the-ninja-135615/pc-143761	PC	9.0	Action, Adventure	Y	2012	9	7

As you can see above, in `some_reviews`, the row indexes start at `10` and end at `20`. Thus, trying `loc` along with numbers lower than `10` or higher than `20` will result in an error:

```
some_reviews.loc[9:21,:]
```

```
---------------------------------------------------------------------------
KeyError Traceback (most recent call last)
<ipython -input-76-5378b774c9a7> in <module>()
----> 1 some_reviews.loc[9:21,:]
/Users/vik/python_envs/dsserver/lib/python3.4/site-packages/pandas/core/indexing.py in __getitem__(self, key)
1198 def __getitem__(self, key):
1199 if type(key) is tuple:
-> 1200 return self._getitem_tuple(key)
1201 else:
1202 return self._getitem_axis(key, axis=0)
/Users/vik/python_envs/dsserver/lib/python3.4/site-packages/pandas/core/indexing.py in _getitem_tuple(self, tup)
702
703 # no multi-index, so validate all of the indexers
--> 704 self._has_valid_tuple(tup)
```

```
705
706 # ug      k for GH #836
/Users/v      thon_envs/dsserver/lib/python3.4/site-packages/pandas/core/indexing.py in _has_valid_tuple(self, key)
129 if i      elf.obj.ndim:
130 rais      exingError('Too many indexers')
--> 131       t self._has_valid_type(k, i):
132 rais_ .__ueError("Location based indexing can only have [%s] "
133 "types" % self._valid_types)
/Users/vik/python_envs/dsserver/lib/python3.4/site-packages/pandas/core/indexing.py in _has_valid_type(self, key, axis)
1258 raise KeyError(
1259 "start bound [%s] is not the [%s]" %
-> 1260 (key.start, self.obj._get_axis_name(axis))
1261 )
1262 if key.stop is not None:
KeyError: 'start bound [9] is not the [index]'
</module></ipython>
```

As we mentioned earlier, column labels can make life much easier when you're working with data. We can specify column labels in the `loc` method to retrieve columns by label instead of by position.

```
reviews.loc
[:5,"score"]
```

```
0 9.0
1 9.0
2 8.5
3 8.5
4 8.5
5 7.0
Name: score, dtype: float64
```

We can also specify more than one column at a time by passing in a list:

```
reviews.loc[:5,["score", "release_year"]]
```

	score	release_year
0	9.0	2012
1	9.0	2012

score	release_year
8.5	2012
8.5	2012
8.5	2012
7.0	2012

Pandas Series Objects

We can retrieve an individual column in Pandas a few different ways. So far, we've seen two types of syntax for this:

- `reviews.iloc[:,1]` — will retrieve the second column.
- `reviews.loc[:,"score_phrase"]` — will also retrieve the second column.

There's a third, even easier, way to retrieve a whole column. We can just specify the column name in square brackets, like with a dictionary:

```
reviews["score"]
```

```
0 9.0
1 9.0
2 8.5
3 8.5
4 8.5
5 7.0
6 3.0
7 9.0
8 3.0
9 7.0
10 7.5
11 7.5
12 7.0
13 9.0
14 9.0
...
18610 6.0
18611 5.8
18612 7.8
18613 8.0
18614 9.2
18615 9.2
18616 7.5
```

```
18617 8.
18618 9.
18619 7.
18620 7.
18621 9.
18622 5.
18623 10.
18624 10.0
Name: score, Length: 18625, dtype: float64
```

We can also use lists of columns with this method:

```
reviews[["score", "release_year"]]
```

	score	release_year
0	9.0	2012
1	9.0	2012
2	8.5	2012
3	8.5	2012
4	8.5	2012
5	7.0	2012
6	3.0	2012
7	9.0	2012
8	3.0	2012
9	7.0	2012
10	7.5	2012
11	7.5	2012
12	7.0	2012
13	9.0	2012
14	9.0	2012
15	6.5	2012
16	6.5	2012
17	8.0	2012
18	5.5	2012
19	7.0	2012
20	7.0	2012
21	7.5	2012
22	7.5	2012

	score	release_year
23	7.5	2012
24	9.0	2012
25	7.0	2012
26	9.0	2012
27	7.5	2012
28	8.0	2012
29	6.5	2012
...	...	...
18595	4.4	2016
18596	6.5	2016
18597	4.9	2016
18598	6.8	2016
18599	7.0	2016

When we retrieve a single column, we're actually retrieving a Pandas Series object. A DataFrame stores tabular data, but a Series stores a single column or row of data.

We can verify that a single column is a Series:

```
type(reviews["score"])
```

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

We can create a Series manually to better understand how it works. To create a Series, we pass a list or NumPy array into the Series object when we instantiate it:

```
s1 = pd.Series([1,2])
s1
```

```
0 1
1 2
dtype: int64
```

A Series can contain any type of data, including mixed types. Here, we create a Series that contains string objects:

```
s2 = pd.Series(["Boris Yeltsin", "Mikhail Gorbachev"])
s2
```

```
0 Boris      n
1 Mikhai     achev
dtype: o
```

Creating A DataFrame in Pandas

We can create a DataFrame by passing multiple Series into the DataFrame class. Here, we pass in the two Series objects we just created,

`s1` as the first row, and `s2` as the second row:

```
pd.DataFrame([s1,s2])
```

	0	1
0	1	2
1	Boris Yeltsin	Mikhail Gorbachev

We can also accomplish the same thing with a list of lists. Each inner list is treated as a row in the resulting DataFrame:

```
pd.DataFrame(
    [
    [1,2],
    ["Boris Yeltsin", "Mikhail Gorbachev"]
    ]
)
```

	0	1
0	1	2
1	Boris Yeltsin	Mikhail Gorbachev

We can specify the column labels when we create a DataFrame:

```
pd.DataFrame(
    [
    [1,2],
    ["Boris Yeltsin", "Mikhail Gorbachev"]
    ],
    colu 'column1", "column2"]
)
```

	column1	column2
0	1	2
1	Boris Yeltsin	Mikhail Gorbachev

As well as the row labels (the index):

```
frame = pd.DataFrame(
    [
    [1,2],
    ["Boris Yeltsin", "Mikhail Gorbachev"]
    ],
    index=["row1", "row2"],
    columns=["column1", "column2"]
)
frame
```

	column1	column2
row1	1	2
row2	Boris Yeltsin	Mikhail Gorbachev

Note also that the indentation and separate lines are not required. We've written the code this way to make it a bit easier to parse, but you'll often encounter it all written as one line. For example, the following code will produce the exact same results as what we see in the table above this paragraph:

```
frame = pd.DataFrame([[1,2],["Boris Yeltsin", "Mikhail Gorbachev"]],index=["row1", "row2"],columns=["column1", "column2"])

frame
```

Anyway, once we've added labels, we're then able index the DataFrame using them:

```
frame.loc["row1":"row2", "column1"]
```

```
row1 1
row2 Bor      tsin
Name: co    , dtype: object
```

We can ski| :ifying the `columns` keyword argument if we pass a dictionary into the `DataFrame` constructor. This will automatically set up column names:

```
frame = pd.DataFrame(
    {
    "column1": [1, "Boris Yeltsin"],
    "column2": [2, "Mikhail Gorbachev"]
    }
)
frame
```

	column1	column2
0	1	2
1	Boris Yeltsin	Mikhail Gorbachev

Pandas DataFrame Methods

As we mentioned earlier, each column in a pandas DataFrame is a Series object:

```
type(reviews["title"])
```

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

We can call most of the same methods on a Series object that we can on a DataFrame, including `head`:

```
reviews["title"].head()
```

```
0 LittleBigPlanet PS Vita
1 LittleBigPlanet PS Vita -- Marvel Super Hero E...
2 Splice: Tree of Life
3 NHL 13
```

```
4 NHL 13
Name: ti dtype: object
```

Pandas Ser d DataFrames also have other methods that make calculations simpler. For example, we can use the pandas.Series.mean method to ne mean of a Series:

```
reviews["score"].mean()
```

```
6.950459060402685
```

We can also call the similar pandas.DataFrame.mean method, which will find the mean of each numerical column in a DataFrame by default:

```
reviews.mean()
```

```
score 6.950459
release_year 2006.515329
release_month 7.138470
release_day 15.603866
dtype: float64
```

We can modify the `axis` keyword argument to `mean` in order to compute the mean of each row or of each column. By default, `axis` is equal to `0`, and will compute the mean of each column. We can also set it to `1` to compute the mean of each row. Note that this will only compute the mean of the numerical values in each row:

```
reviews.mean(axis=1)
```

```
0 510.500
1 510.500
2 510.375
3 510.125
4 510.125
5 509.750
6 508.750
7 510.250
8 508.750
9 509.750
10 509.875
```

```
11 509.875
12 509.5
13 509.2
14 509.2
...
18610 51
18611 50
18612 509.200
18613 508.000
18614 515.050
18615 515.050
18616 508.375
18617 508.600
18618 515.025
18619 514.725
18620 514.650
18621 515.000
18622 513.950
18623 515.000
18624 515.000
Length: 18625, dtype: float64
```

There are quite a few methods on Series and DataFrames that behave like `mean`. Here are some handy ones:

- pandas.DataFrame.corr — finds the correlation between columns in a DataFrame.
- pandas.DataFrame.count — counts the number of non-null values in each DataFrame column.
- pandas.DataFrame.max — finds the highest value in each column.
- pandas.DataFrame.min — finds the lowest value in each column.
- pandas.DataFrame.median — finds the median of each column.
- pandas.DataFrame.std — finds the standard deviation of each column.

For example, we can use the `corr` method to see if any columns correlate with `score`. This would tell us if games released more recently have been getting higher reviews (`release_year`), or if games released towards the end of the year score better (`release_month`):

```
reviews.corr()
```

	score	release_year	release_month	release_day
score	1.000000	0.062716	0.007632	0.020079
release_year	0.062716	1.000000	-0.115515	0.016867

	score	release_year	release_month	release_day
release_month	0.007632	-0.115515	1.000000	-0.067964
release_day	0.020079	0.016867	-0.067964	1.000000

As we can s ove, none of our numeric columns correlate with `score`, so we know that release timing doesn't linearly relate to review score.

DataFrame Math with Pandas

We can also perform math operations on Series or DataFrame objects in Python with pandas. For example, we can divide every value in the `score` column by `2` to switch the scale from `0-10` to `0-5`:

```
reviews["score"] / 2
```

```
0 4.50
1 4.50
2 4.25
3 4.25
4 4.25
5 3.50
6 1.50
7 4.50
8 1.50
9 3.50
10 3.75
11 3.75
12 3.50
13 4.50
14 4.50
...
18610 3.00
18611 2.90
18612 3.90
18613 4.00
18614 4.60
18615 4.60
18616 3.75
18617 4.20
18618 4.55
18619 3.95
18620 3.80
```

```
18621 4.50
18622 2.
18623 5.
18624 5.
Name: sc    Length: 18625, dtype: float64
```

All the common mathematical operators that work in Python, like `+`, `-`, `*`, `/`, and `^` will work in pandas on Series or DataFrames, and will apply to each element in a DataFrame or a Series.

Boolean Indexing in Pandas

Now that we've gotten a bit of the pandas basics down, let's move towards the meat of our analysis. We saw earlier that the mean of all the values in the `score` column of `reviews` is around `7`. What if we wanted to find all the games that got an above-average score?

We could start by doing a comparison. The comparison compares each value in a Series to a specified value, then generate a Series full of Boolean values indicating the status of the comparison. For example, we can see which of the rows have a `score` value higher than `7`:

```
score_filter = reviews["score"] > 7
score_filter
```

```
0 True
1 True
2 True
3 True
4 True
5 False
6 False
7 True
8 False
9  False
10 True
11 True
12 False
13 True
14 True...
18610 False
18611 False
18612 True
18613 True
18614 True
```

```
18615 True
18616 Tr
18617 Tr
18618 Tr
18619 Tr
18620 Tr
18621 Tr
18622 False
18623 True
18624 True
Name: score, Length: 18625, dtype: bool
```

Once we have a Boolean Series, we can use it to select only rows in a DataFrame where the Series contains the value `True`. So, we could only select rows in `reviews` where `score` is greater than `7`:

```
filtered_reviews = reviews[score_filter]
filtered_reviews.head()
```

	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day
0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y	2012	9	12
1	Amazing	LittleBigPlanet PS Vita — Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y	2012	9	12
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N	2012	9	12
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N	2012	9	11
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N	2012	9	11

It's possible to use multiple conditions for filtering. Let's say we want to find games released for the `Xbox One` that have a score of more than `7`. In the below code, we:

- Setup a filter with two conditions:
  - Check if `score` is greater than `7`.

  - Chec' 'f atform equals `Xbox One`
- Apply tl r to `reviews` to get only the rows we want.
- Use the method to print the first `5` rows of `filtered_reviews`.

```
xbox_one er = (reviews["score"] > 7) & (reviews["platform"] == "Xbox One")
filtered ews = reviews[xbox_one_filter]
filtered_reviews.head()
```

	score_phrase	title	url	platform	score	genre	editors_choice	release_year	release_month	release_day
17137	Amazing	Gone Home	/games/gone-home/xbox-one-20014361	Xbox One	9.5	Simulation	Y	2013	8	15
17197	Amazing	Rayman Legends	/games/rayman-legends/xbox-one-20008449	Xbox One	9.5	Platformer	Y	2013	8	26
17295	Amazing	LEGO Marvel Super Heroes	/games/lego-marvel-super-heroes/xbox-one-20000826	Xbox One	9.0	Action	Y	2013	10	22
17313	Great	Dead Rising 3	/games/dead-rising-3/xbox-one-124306	Xbox One	8.3	Action	N	2013	11	18
17317	Great	Killer Instinct	/games/killer-instinct-2013/xbox-one-20000538	Xbox One	8.4	Fighting	N	2013	11	18

When filtering with multiple conditions, it's important to put each condition in parentheses, and separate them with a single ampersand (`&`).

Pandas Plotting

Now that we know how to filter, we can create plots to observe the review distribution for the `Xbox One` vs the review distribution for the `PlayStation 4`. This will help us figure out which console has better games.

We can do this via a histogram, which will plot the frequencies for different score ranges. We can make a histogram for each console using the pandas.DataFrame.plot method. This method utilizes matplotlib, the popular Python plotting library, under the hood, to generate good-looking plots.

The `plot` d defaults to drawing a line graph. We'll need to pass in the keyword argument `kind="hist"` to draw a histogram instead. In low code, we:

- Call `%ma` `lib inline` to set up plotting inside a Jupyter notebook.
- Filter `re` to only have data about the `Xbox One`.
- Plot the column.

```
reviews[reviews["platform"] == "Xbox One"]["score"].plot(kind="hist")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c9c5438>
```

We can also do the same for the `PS4`:

```
reviews[reviews["platform"] == "PlayStation 4"]["score"].plot(kind="hist")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c9e0e80>
```

It appears from our histogram that the `PlayStation 4` has many more highly rated games than the `Xbox One`.

Obviously, this is just the tip of the iceberg when it comes to the potential directions we could take analyzing this data set, but we're already off to a great start: we've imported a data set using Python and pandas, learned to select the data points we want using a variety of different indexing methods, and done some quick exploratory data analysis to answer the question we started with.

Free Pandas Cheat Sheet

If you're int ed in learning more about pandas, check out our interactive course on NumPy and pandas. You can register and do the first missio ree. You also might like to take your pandas skills to the next level with our free pandas cheat sheet!

Furthe ading

Hopefully, this andas tutorial helped you to read, explore, analyze, and visualize data using Pandas and Python. In the next post, we cover grouping data and doing more advanced computations. You can find it here. If you want to read more about Pandas, check out these resources:

- Dataquest Pandas Course
- 10 minutes to Pandas
- Intro to Pandas data structures

This article was last updated in September 2019.

Get Free Data Science Resources

Sign up for free to get our weekly newsletter with data science, **Python**, **R**, and **SQL** resource links. Plus, you get access to our free, interactive online course content!

SIGN UP

Vik Paruchuri

Vik is the CEO and Founder of Dataquest.

TAGS beginner, games, gaming, Learn Python, Pandas, python, tutorial, Tutorials, video games

You may also like

SQL Cheat Sheet — SQL Reference Guide for Data Analysis

READ MORE

Do You Need a SQL Certification to Get a Data Job in 2021?

READ MORE

45 Fun (and Unique) Python Project Ideas for Easy Learning

READ MORE

SQL Tutorial: Selecting Ungrouped Columns Without Aggregate Functions

READ MORE

Learn by ~~watching videos~~ coding!

Try it now >>

Search

Categories

Building A Data Science Portfolio

Cheat Sheets

Data Science Career Tips

Data Science Projects

Data Science Tutorials

Top Picks

JULY 2, 2019

Why Jorge Prefers Dataquest Over DataCamp for Learning Data Analysis

JULY 21, 2020

Tutorial: Better Blog Post Analysis with googleAnalyticsR

AUGUST 1, 2020

How to Learn Python (Step-by-Step) in 2020

MAY 4, 2020

How to Learn Data Science (Step-By-Step) in 2020

JULY 6, 2020

Data Science Certificates in 2020 (Are They Worth It?)

Get started with Dataquest today - for free!

Sign up now

Or, visit our pricing page to learn about our Basic and Premium plans.

Data Science Courses
For Business
For Academia
About Dataquest
Success Stories
Careers

Pricing
Contact Us
Community
Privacy Policy
Terms of Use

Blog
Facebook
Twitter
LinkedIn
Resource List