8 DECEMBER 2017 / PANDAS

Using Excel with pandas

Excel is one of the most popular and widely-used data tools; it's hard to find an organization that doesn't work with it in some way. From analysts, to sales VPs, to CEOs, various professionals use Excel for both quick stats and serious data crunching.

With Excel being so pervasive, data professionals must be familiar with it. You'll also want a tool that can easily read and write Excel files — pandas is perfect for this.

Pandas has excellent methods for reading all kinds of data from Excel files. You can also export your results from pandas back to Excel, if that's preferred by your intended audience. Pandas is great for other routine data analysis tasks, such as:

- quick Exploratory Data Analysis (EDA)
- drawing attractive plots
- feeding data into machine learning tools like scikit-learn
- building machine learning models on your data
- taking cleaned and processed data to any number of data tools

Pandas is better at automating data processing tasks than Excel,

In this tutorial, we are going to show you how to work with Excel files in pandas. We will cover the following concepts.

- setting up your computer with the necessary software
- reading in data from Excel files into pandas
- data exploration in pandas
- visualizing data in pandas using the matplotlib visualization library
- manipulating and reshaping data in pandas
- moving data from pandas into Excel

Note that this tutorial does not provide a deep dive into pandas. To explore pandas more, check out our course.

System prerequisites

We will use Python 3 and Jupyter Notebook to demonstrate the code in this tutorial.
In addition to Python and Jupyter Notebook, you will need the following Python modules:

- matplotlib - data visualization
- NumPy - numerical data functionality
- OpenPyXL - read/write Excel 2010 xlsx/xlsm files
- pandas - data import, clean-up, exploration, and analysis
- xlrd - read Excel data
- xlwt - write to Excel
- XlsxWriter - write to Excel (xlsx) files

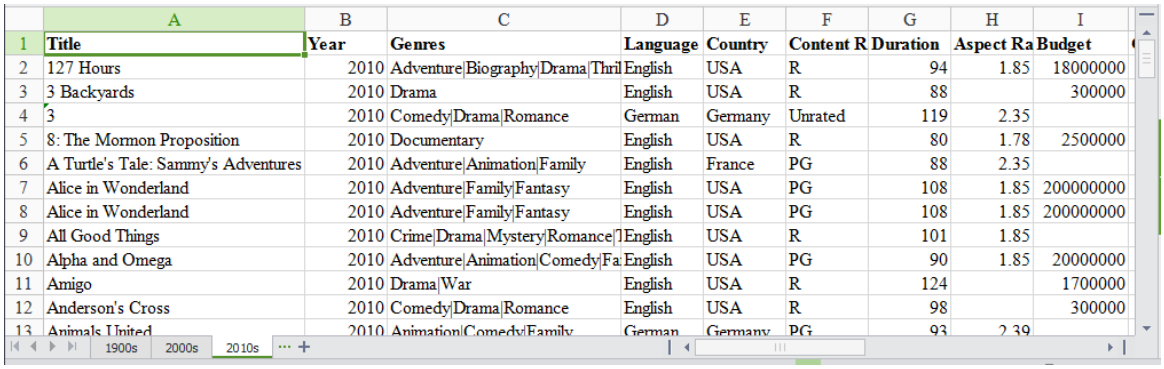
There are multiple ways to get set up with all the modules. We cover three of the most common scenarios below.

- If you have Python installed via Anaconda package manager, you can install the required modules using the command `conda install`. For example, to install pandas, you would execute the command - `conda install pandas`.
- If you already have a regular, non-Anaconda Python installed on the computer, you can install the required modules using `pip`. Open your command line program and execute command `pip install <module name>` to install a module. You should replace `<module name>` with the actual name of the module you are trying to install. For example, to install pandas, you would execute

- If you don't have Python already installed, you should get it through the Anaconda package manager. Anaconda provides installers for Windows, Mac, and Linux Computers. If you choose the full installer, you will get all the modules you need, along with Python and pandas within a single package. This is the easiest and fastest way to get started.

The data set

In this tutorial, we will use a multi-sheet Excel file we created from Kaggle's IMDB Scores data. You can download the file here.



	A	B	C	D	E	F	G	H	I
1	Title	Year	Genres	Language	Country	Content R	Duration	Aspect Ra	Budget
2	127 Hours	2010	Adventure|Biography|Drama|Thril	English	USA	R	94	1.85	18000000
3	3 Backyards	2010	Drama	English	USA	R	88		300000
4	3	2010	Comedy|Drama|Romance	German	Germany	Unrated	119	2.35	
5	8: The Mormon Proposition	2010	Documentary	English	USA	R	80	1.78	2500000
6	A Turtle's Tale: Sammy's Adventures	2010	Adventure|Animation|Family	English	France	PG	88	2.35	
7	Alice in Wonderland	2010	Adventure|Family|Fantasy	English	USA	PG	108	1.85	200000000
8	Alice in Wonderland	2010	Adventure|Family|Fantasy	English	USA	PG	108	1.85	200000000
9	All Good Things	2010	Crime|Drama|Mystery|Romance|	English	USA	R	101	1.85	
10	Alpha and Omega	2010	Adventure|Animation|Comedy|Fa	English	USA	PG	90	1.85	20000000
11	Amigo	2010	Drama|War	English	USA	R	124		1700000
12	Anderson's Cross	2010	Comedy|Drama|Romance	English	USA	R	98		300000
13	Animals United	2010	Animation|Comedy|Family	German	Germany	PG	93	2.39	

1900s 2000s 2010s

Our Excel file has three sheets: '1900s,' '2000s,' and '2010s.' Each sheet has data for movies from those years.

We will use this data set to find the ratings distribution for the movies, visualize movies with highest ratings and net earnings and calculate statistical information about the movies. We will

demonstrating pandas capabilities to work with Excel data.

Read data from the Excel file

We need to first import the data from the Excel file into pandas. To do that, we start by importing the pandas module.

```
import pandas as pd
```

We then use the pandas' read_excel method to read in data from the Excel file. The easiest way to call this method is to pass the file name. If no sheet name is specified then it will read the first sheet in the index (as shown below).

```
excel_file = 'movies.xls'
movies = pd.read_excel(excel_file)
```

Here, the `read_excel` method read the data from the Excel file into a pandas DataFrame object. Pandas defaults to storing data in DataFrames. We then stored this DataFrame into a variable called `movies`.

Pandas has a built-in `DataFrame.head()` method that we can use to easily display the first few rows of our DataFrame. If no argument is passed, it will display first five rows. If a number is passed, it will display the equal number of rows from the top.

```
movies.head()
```

	Title	Year	Genres
0	Intolerance: Love's Struggle Throughout the Ages	1916	Drama|Hist
1	Over the Hill to the Poorhouse	1920	Crime|Dram
2	The Big Parade	1925	Drama|Rom
3	Metropolis	1927	Drama|Sci-F
4	Pandora's Box	1929	Crime|Dram

5 rows × 25 columns

Excel files quite often have multiple sheets and the ability to read a specific sheet or all of them is very important. To make this easy, the pandas `read_excel` method takes an argument called `sheetname` that tells pandas which sheet to read in the data from. For this, you can either use the sheet name or the sheet number. Sheet numbers start with zero. If the `sheetname` argument is not given, it defaults to zero and pandas will import the first sheet.

By default, pandas will automatically assign a numeric index or row label starting with zero. You may want to leave the default index as such if your data doesn't have a column with unique values that can serve as a better index. In case there is a column that you feel would serve as a better index, you can override the default behavior by setting `index_col` property to a column. It

list of numeric values for creating a multi-index.

In the below code, we are choosing the first column, 'Title', as index (index=0) by passing zero to the `index_col` argument.

```
movies_sheet1 = pd.read_excel(excel_file, sheetname=0, index_col=0
movies_sheet1.head()
```

	Year	Genres	I
Title			
Intolerance: Love's Struggle Throughout the Ages	1916	Drama|History|War	I
Over the Hill to the Poorhouse	1920	Crime|Drama	I
The Big Parade	1925	Drama|Romance|War	I
Metropolis	1927	Drama|Sci-Fi	(
Pandora's Box	1929	Crime|Drama|Romance	(

5 rows × 24 columns

already read the first sheet in a DataFrame above. Now, using the same syntax, we will read in rest of the two sheets too.

```
movies_sheet2 = pd.read_excel(excel_file, sheetname=1, index_col=0
movies_sheet2.head()
```

	Year	Genres	Language	Country
Title				
102 Dalmatians	2000	Adventure|Comedy|Family	English	USA
28 Days	2000	Comedy|Drama	English	USA
3 Strikes	2000	Comedy	English	USA
Aberdeen	2000	Drama	English	UK
All the Pretty Horses	2000	Drama|Romance|Western	English	USA

5 rows × 24 columns

```
movies_sheet3 = pd.read_excel(excel_file, sheetname=2, index_col=0
movies_sheet3.head()
```

	Year	Genres
Title		
127 Hours	2010.0	Adventure|Biography|Drama|Thrille
3 Backyards	2010.0	Drama
3	2010.0	Comedy|Drama|Romance
8: The Mormon Proposition	2010.0	Documentary
A Turtle's Tale: Sammy's Adventures	2010.0	Adventure|Animation|Family

5 rows × 24 columns

Since all the three sheets have similar data but for different records\movies, we will create a single DataFrame from all the three DataFrames we created above. We will use the pandas `concat` method for this and pass in the names of the three DataFrames we just created and assign the results to a new DataFrame object, `movies`. By keeping the DataFrame name same as before, we are over-writing the previously created DataFrame.

```
movies = pd.concat([movies_sheet1, movies_sheet2, movies_sheet3])
```

rows in the combined DataFrame by calling the method `shape` on it that will give us the number of rows and columns.

```
movies.shape
```

```
(5042, 24)
```

Using the ExcelFile class to read multiple sheets

We can also use the ExcelFile class to work with multiple sheets from the same Excel file. We first wrap the Excel file using `ExcelFile` and then pass it to `read_excel` method.

```
xlsx = pd.ExcelFile(excel_file)
xlsx = pd.ExcelFile(excel_file)
movies_sheets = []
for sheet in xlsx.sheet_names:
    movies_sheets.append(xlsx.parse(sheet))
movies = pd.concat(movies_sheets)
```

If you are reading an Excel file with a lot of sheets and are creating a lot of DataFrames, `ExcelFile` is more convenient and efficient in comparison to `read_excel` . With ExcelFile, you only need to pass the Excel file once, and then you can use it to get the DataFrames. When using `read_excel` , you pass the Excel file every time and hence the file is loaded again for every sheet.

sheets with a large number of rows.

Exploring the data

Now that we have read in the movies data set from our Excel file, we can start exploring it using pandas. A pandas DataFrame stores the data in a tabular format, just like the way Excel displays the data in a sheet. Pandas has a lot of built-in methods to explore the DataFrame we created from the Excel file we just read in.

We already introduced the method `head` in the previous section that displays few rows from the top from the DataFrame. Let's look at few more methods that come in handy while exploring the data set.

We can use the `shape` method to find out the number of rows and columns for the DataFrame.

```
movies.shape
```

```
(5042, 25)
```

This tells us our Excel file has 5042 records and 25 columns or observations. This can be useful in reporting the number of records and columns and comparing that with the source data set.

We can use the `tail` method to view the bottom rows. If no parameter is passed, only the bottom five rows are returned.

```
movies.tail()
```

	Title	Year	Genres
1599	War & Peace	NaN	Drama|History|Romance|War
1600	Wings	NaN	Comedy|Drama
1601	Wolf Creek	NaN	Drama|Horror|Thriller
1602	Wuthering Heights	NaN	Drama|Romance
1603	Yu-Gi-Oh! Duel Monsters	NaN	Action|Adventure|Animation|Fami

5 rows × 25 columns

In Excel, you're able to sort a sheet based on the values in one or more columns. In pandas, you can do the same thing with the `sort_values` method. For example, let's sort our movies DataFrame based on the Gross Earnings column.

```
sorted_by_gross = movies.sort_values(['Gross Earnings'], ascending
```

Since we have the data sorted by values in a column, we can do few interesting things with it. For example, we can display the top 10 movies by Gross Earnings.

```
sorted_by_gross["Gross Earnings"].head(10)
```

```
1867    760505847.0
1027    658672302.0

1263    652177271.0
610     623279547.0
611     623279547.0
1774    533316061.0
1281    474544677.0
226     460935665.0
1183    458991599.0
618     448130642.0
Name: Gross Earnings, dtype: float64
```

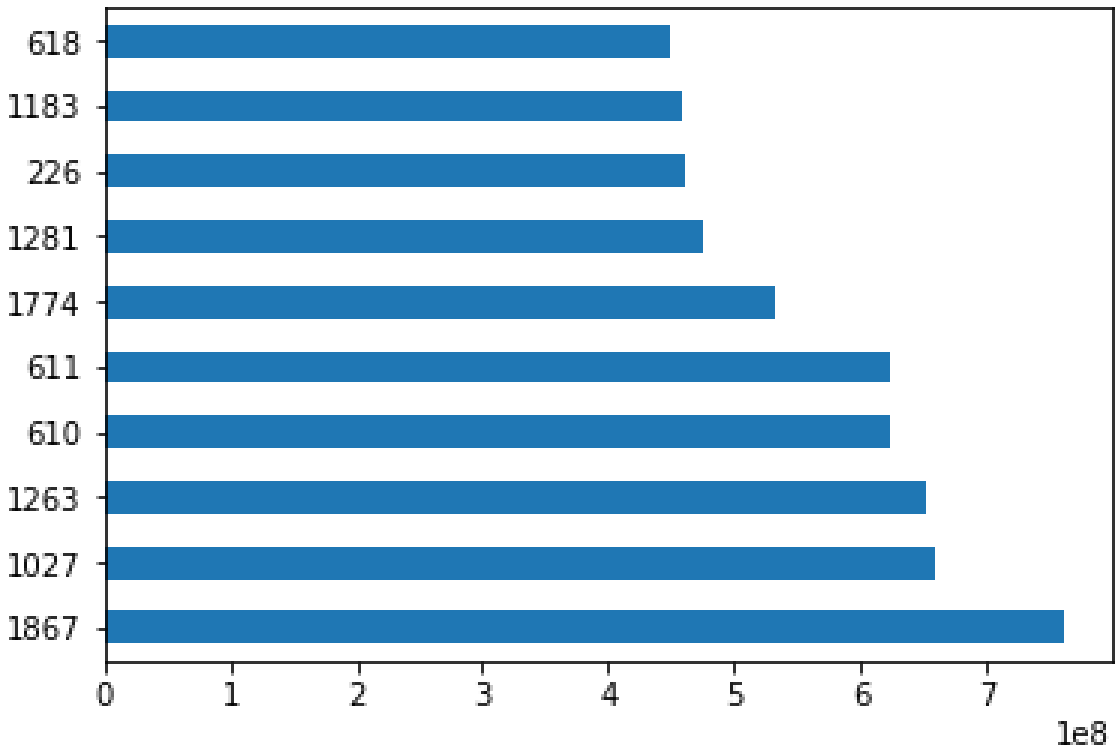
We can also create a plot for the top 10 movies by Gross Earnings. Pandas makes it easy to visualize your data with plots and charts through matplotlib, a popular data visualization library. With a couple lines of code, you can start plotting. Moreover, matplotlib plots work well inside Jupyter Notebooks since you can displace the plots right under the code.

First, we import the matplotlib module and set matplotlib to

```
import matplotlib.pyplot as plt
%matplotlib inline
```

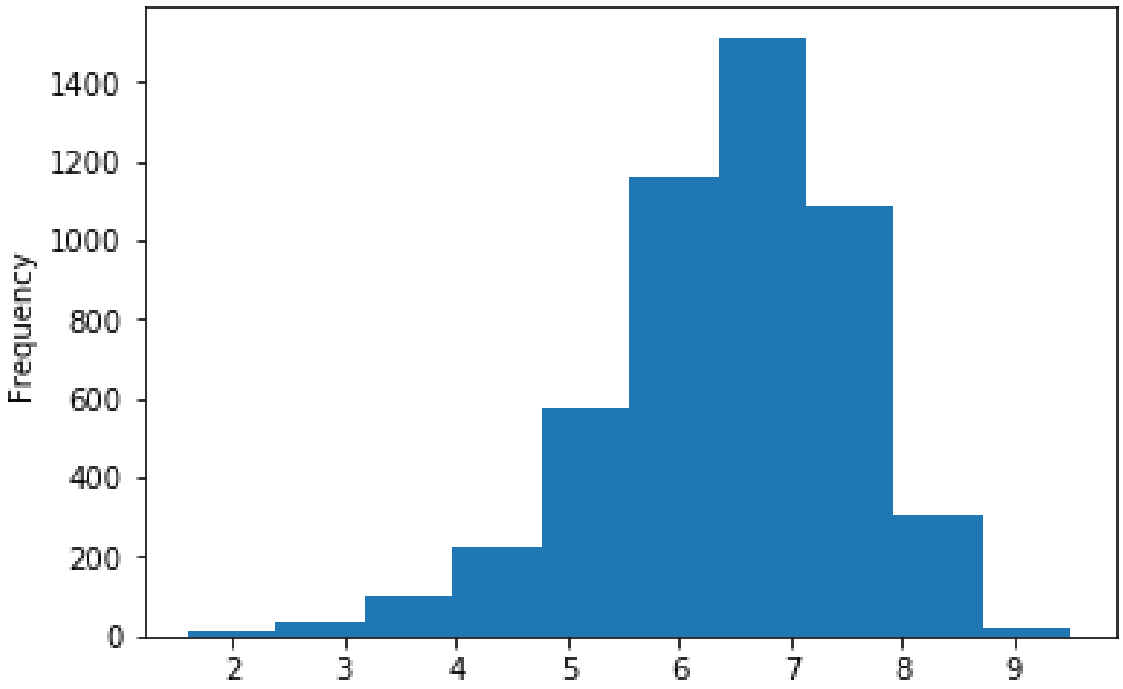
We will draw a bar plot where each bar will represent one of the top 10 movies. We can do this by calling the `plot` method and setting the argument `kind` to `barh`. This tells `matplotlib` to draw a horizontal bar plot.

```
sorted_by_gross['Gross Earnings'].head(10).plot(kind="barh")
plt.show()
```



of IMDB Scores across all movies. Histograms are a good way to visualize the distribution of a data set. We use the `plot` method on the IMDB Scores series from our movies DataFrame and pass it the argument `kind="hist"`.

```
movies['IMDB Score'].plot(kind="hist")
plt.show()
```



This data visualization suggests that most of the IMDB Scores fall between six and eight.

Getting statistical information about the data

Pandas has some very handy methods to look at the statistical data about our data set. For example, we can use the `describe` method to get a statistical summary of the data set.

```
movies.describe()
```

	Year	Duration	Aspect Ratio	Budget	Gro
count	4935.000000	5028.000000	4714.000000	4.551000e+03	4.1
mean	2002.470517	107.201074	2.220403	3.975262e+07	4.8
std	12.474599	25.197441	1.385113	2.061149e+08	6.8
min	1916.000000	7.000000	1.180000	2.180000e+02	1.6
25%	1999.000000	93.000000	1.850000	6.000000e+06	5.3
50%	2005.000000	103.000000	2.350000	2.000000e+07	2.5
75%	2011.000000	118.000000	2.350000	4.500000e+07	6.2
max	2016.000000	511.000000	16.000000	1.221550e+10	7.6

The `describe` method displays below information for each of the columns.

- the count or number of values
- mean
- standard deviation
- minimum, maximum
- 25%, 50%, and 75% quantile

Please note that this information will be calculated only for the numeric values.

We can also use the corresponding method to access this information one at a time. For example, to get the mean of a particular column, you can use the `mean` method on that column.

```
movies["Gross Earnings"].mean()
```

```
48468407.526809327
```

Just like mean, there are methods available for each of the statistical information we want to access. You can read about these methods in our free pandas cheat sheet.

Reading files with no header and skipping records

Earlier in this tutorial, we saw some ways to read a particular kind of Excel file that had headers and no rows that needed skipping. Sometimes, the Excel sheet doesn't have any header row. For such instances, you can tell pandas not to consider the first row as header or columns names. And If the Excel sheet's first few rows contain data that should not be read in, you can ask the `read_excel` method to skip a certain number of rows, starting from the top.

For example, look at the top few rows of this Excel file.

	A	B	C	D	E	F	G	H
1								
2	**IMDB Ratings data**							
3	**Available from Kaggle.com**							
4								
5	Metropolis	1927	Drama|Sci-Fi	German	Germany	Not Rated	145	1.33
6	Pandora's Box	1929	Crime|Drama|Romance	German	Germany	Not Rated	110	1.33
7	The Broadway Melody	1929	Musical|Romance	English	USA	Passed	100	1.37
8	Hell's Angels	1930	Drama|War	English	USA	Passed	96	1.2
9	A Farewell to Arms	1932	Drama|Romance|War	English	USA	Unrated	79	1.37
10	42nd Street	1933	Comedy|Musical|Romance	English	USA	Unrated	89	1.37
11	She Done Him Wrong	1933	Comedy|Drama|History|Musical|Romance	English	USA	Approved	66	1.37
12	It Happened One Night	1934	Comedy|Romance	English	USA	Unrated	65	1.37
13	Top Hat	1935	Comedy|Musical|Romance	English	USA	Approved	81	1.37
14	Modern Times	1936	Comedy|Drama|Family	English	USA	G	87	1.37

This file obviously has no header and first four rows are not actual records and hence should not be read in. We can tell read_excel there is no header by setting argument `header` to `None` and we can skip first four rows by setting argument `skiprows` to four.

```
movies_skip_rows = pd.read_excel("movies-no-header-skip-rows.xls",
movies_skip_rows.head(5)
```

	0	1	2	3	4
0	Metropolis	1927	Drama|Sci-Fi	German	C
1	Pandora's Box	1929	Crime|Drama|Romance	German	C
2	The Broadway Melody	1929	Musical|Romance	English	U
3	Hell's Angels	1930	Drama|War	English	U
4	A Farewell to Arms	1932	Drama|Romance|War	English	U

5 rows × 25 columns

We skipped four rows from the sheet and used none of the rows as the header. Also, notice that one can combine different options in a single read statement. To skip rows at the bottom of the sheet, you can use option `skip_footer`, which works just like `skiprows`, the only difference being the rows are counted from the bottom upwards.

The column names in the previous DataFrame are numeric and were allotted as default by the pandas. We can rename the column names to descriptive ones by calling the method `columns` on the DataFrame and passing the column names as a list.

```
movies_skip_rows.columns = ['Title', 'Year', 'Genres', 'Language',
       'Duration', 'Aspect Ratio', 'Budget', 'Gross Earnings', 'Di
       'Actor 1', 'Actor 2', 'Actor 3', 'Facebook Likes - Director
       'Facebook Likes - Actor 1', 'Facebook Likes - Actor 2',
       'Facebook Likes - Actor 3', 'Facebook Likes - cast Total',
       'Facebook likes - Movie', 'Facenumber in posters', 'User Vo
```

	Title	Year	Genres	Language	C
0	Metropolis	1927	Drama|Sci-Fi	German	C
1	Pandora's Box	1929	Crime|Drama|Romance	German	C
2	The Broadway Melody	1929	Musical|Romance	English	U
3	Hell's Angels	1930	Drama|War	English	U
4	A Farewell to Arms	1932	Drama|Romance|War	English	U

5 rows × 25 columns

Now that we have seen how to read a subset of rows from the Excel file, we can learn how to read a subset of columns.

Reading a subset of columns

Although read_excel defaults to reading and importing all columns, you can choose to import only certain columns. By passing parse_cols=6, we are telling the `read_excel` method to

read only the first columns till index six or first seven columns (the first column being indexed zero).

```
movies_subset_columns = pd.read_excel(excel_file, parse_cols=6)
movies_subset_columns.head()
```

	Title	Year	Genres
0	Intolerance: Love's Struggle Throughout the Ages	1916	Drama|Hist
1	Over the Hill to the Poorhouse	1920	Crime|Dram
2	The Big Parade	1925	Drama|Rom
3	Metropolis	1927	Drama|Sci-F
4	Pandora's Box	1929	Crime|Dram

Alternatively, you can pass in a list of numbers, which will let you import columns at particular indexes.

Applying formulas on the columns

One of the much-used features of Excel is to apply formulas to create new columns from existing column values. In our Excel file, we have Gross Earnings and Budget columns. We can get Net earnings by subtracting Budget from Gross earnings. We could then apply this formula in the Excel file to all the rows. We can do this in pandas also as shown below.

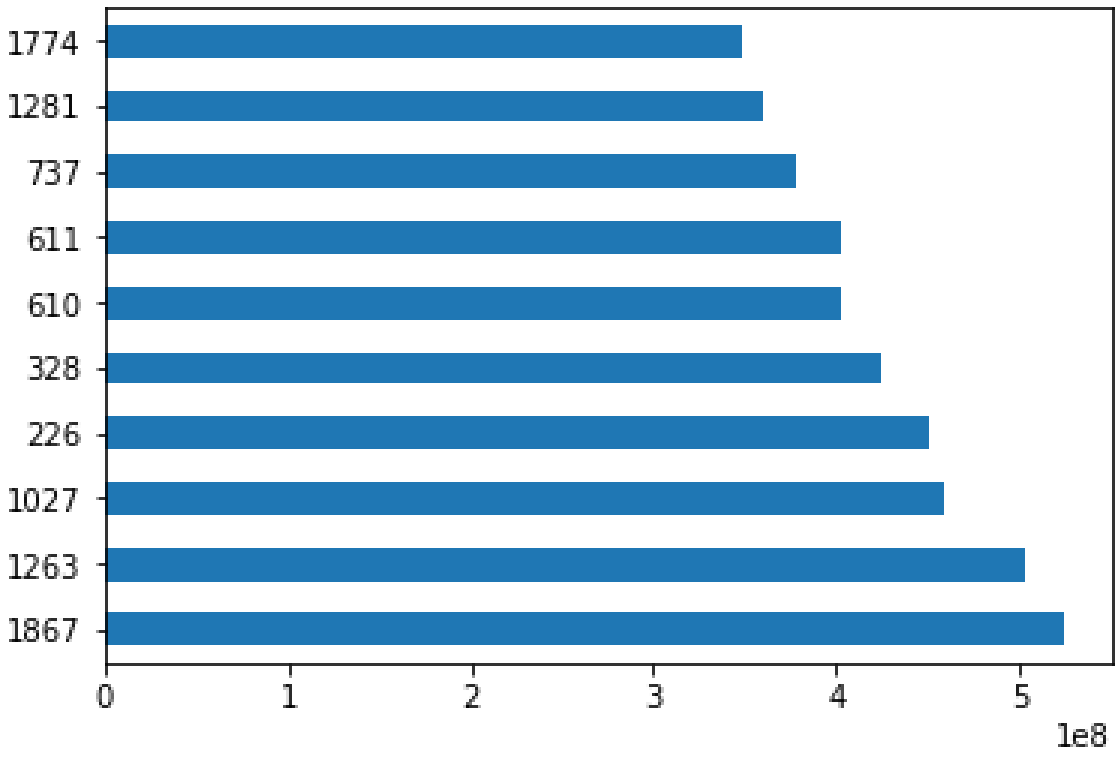
```
movies["Net Earnings"] = movies["Gross Earnings"] - movies["Budget
```

Above, we used pandas to create a new column called Net Earnings, and populated it with the difference of Gross Earnings and Budget. It's worth noting the difference here in how

Python, the calculations happen and the values are stored - if Gross Earnings for one movie was manually changed, Net Earnings won't be updated.

Let's use the `sot_values` method to sort the data by the new column we created and visualize the top 10 movies by Net Earnings.

```
sorted_movies = movies[['Net Earnings']].sort_values(['Net Earning
sorted_movies.head(10)['Net Earnings'].plot.barh()
plt.show()
```



Pivot Table in pandas

Advanced Excel users also often use pivot tables. A pivot table summarizes the data of another table by grouping the data on an index and applying operations such as sorting, summing, or averaging. You can use this feature in pandas too.

We need to first identify the column or columns that will serve as the index, and the column(s) on which the summarizing formula will be applied. Let's start small, by choosing Year as the index column and Gross Earnings as the summarization column and creating a separate DataFrame from this data.

```
movies_subset = movies[['Year', 'Gross Earnings']]
movies_subset.head()
```

	Year	Gross Earnings
0	1916.0	NaN
1	1920.0	3000000.0
2	1925.0	NaN
3	1927.0	26435.0
4	1929.0	9950.0

We now call `pivot_table` on this subset of data. The method `pivot_table` takes a parameter `index`. As mentioned, we want to use Year as the index.

```
earnings_by_year = movies_subset.pivot_table(index=['Year'])
earnings_by_year.head()
```

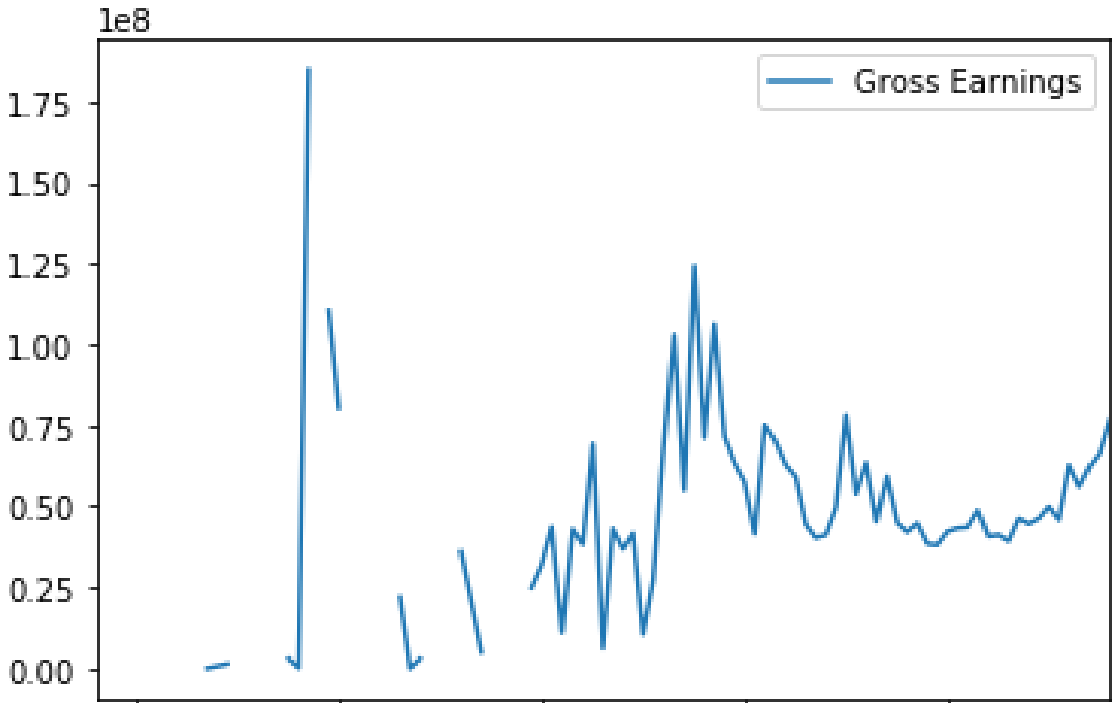
	Gross Earnings
Year	
1916.0	NaN
1920.0	3000000.0
1925.0	NaN
1927.0	26435.0
1929.0	1408975.0

This gave us a pivot table with grouping on Year and summarization on the sum of Gross Earnings. Notice, we didn't need to specify Gross Earnings column explicitly as pandas

should be applied.

We can use this pivot table to create some data visualizations. We can call the `plot` method on the DataFrame to create a line plot and call the `show` method to display the plot in the notebook.

```
earnings_by_year.plot()
plt.show()
```



We saw how to pivot with a single column as the index. Things will get more interesting if we can use multiple columns. Let's create another DataFrame subset but this time we will choose the columns, Country, Language and Gross Earnings.

```
movies_subset = movies[['Country', 'Language', 'Gross Earnings']]
movies_subset.head()
```

	Country	Language	Gross Earnings
0	USA	NaN	NaN
1	USA	NaN	3000000.0
2	USA	NaN	NaN
3	Germany	German	26435.0
4	Germany	German	9950.0

We will use columns Country and Language as the index for the pivot table. We will use Gross Earnings as summarization table,

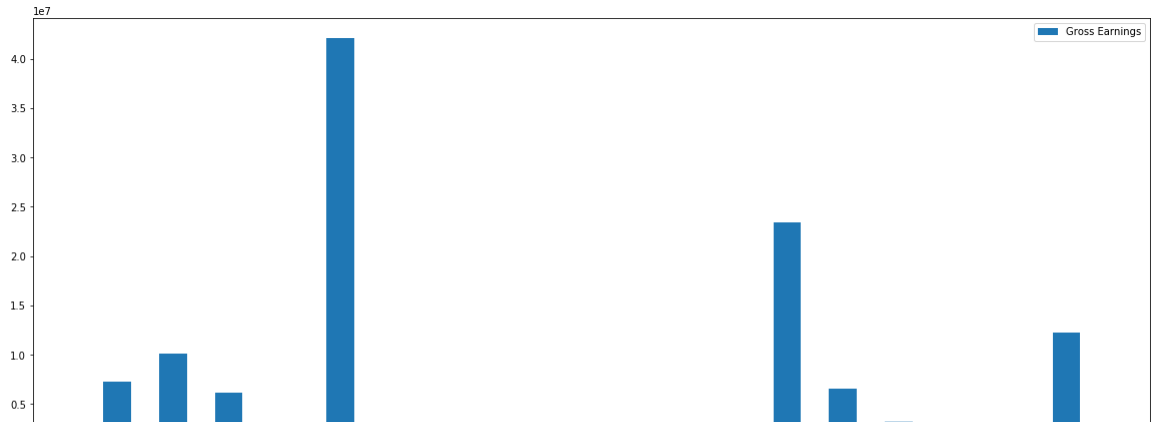
however, we do not need to specify this explicitly as we saw earlier.

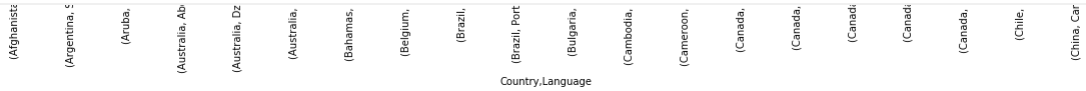
```
earnings_by_co_lang = movies_subset.pivot_table(index=['Country',
earnings_by_co_lang.head()
```

		Gross Earnings
Country	**Language**	
Afghanistan	**Dari**	1.127331e+06
Argentina	**Spanish**	7.230936e+06
Aruba	**English**	1.007614e+07
Australia	**Aboriginal**	6.165429e+06
	Dzongkha	5.052950e+05

Let's visualize this pivot table with a bar plot. Since there are still few hundred records in this pivot table, we will plot just a few of them.

```
earnings_by_co_lang.head(20).plot(kind='bar', figsize=(20,8))
plt.show()
```





Exporting the results to Excel

If you're going to be working with colleagues who use Excel, saving Excel files out of pandas is important. You can export or write a pandas DataFrame to an Excel file using pandas `to_excel` method. Pandas uses the `xlwt` Python module internally for writing to Excel files. The `to_excel` method is called on the DataFrame we want to export.We also need to pass a filename to which this DataFrame will be written.

```
movies.to_excel('output.xlsx')
```

By default, the index is also saved to the output file. However, sometimes the index doesn't provide any useful information. For example, the `movies` DataFrame has a numeric auto-increment index, that was not part of the original Excel data.

```
movies.head()
```

	Title	Year	Genres
0	Intolerance: Love's Struggle Throughout the Ages	1916.0	Drama|His
1	Over the Hill to the Poorhouse	1920.0	Crime|Dra

2	The Big Parade	1925.0	Drama|Ro
3	Metropolis	1927.0	Drama|Sc
4	Pandora's Box	1929.0	Crime|Dra

5 rows × 26 columns

You can choose to skip the index by passing along index-False.

```
movies.to_excel('output.xlsx', index=False)
```

We need to be able to make our output files look nice before we can send it out to our co-workers. We can use pandas `ExcelWriter` class along with the `XlsxWriter` Python module to apply the formatting.

We can do use these advanced output options by creating a `ExcelWriter` object and use this object to write to the EXcel file.

```
writer = pd.ExcelWriter('output.xlsx', engine='xlsxwriter')

movies.to_excel(writer, index=False, sheet_name='report')

workbook = writer.book

worksheet = writer.sheets['report']
```

We can apply customizations by calling `add_format` on the

as bold.

```
header_fmt = workbook.add_format({'bold': True})
worksheet.set_row(0, None, header_fmt)
```

Finally, we save the output file by calling the method `save` on the writer object.

```
writer.save()
```

As an example, we saved the data with column headers set as bold. And the saved file looks like the image below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Title	Year	Genres	Language	Country	ntent Rati	Duration	spect Rati	Budget	oss Earnin	Director	Actor 1	Actor 2	Actor 3
2	Intoleranc	1916	Drama|History|War		USA	Not Rated	123	1.33	385907		D.W. Griff	Lillian Gis	Mae Mars	Walter Lo
3	Over the H	1920	Crime|Drama		USA		110	1.33	100000	3000000	Harry F. M	Stephen C	Johnnie W	Mary Carr
4	The Big Pa	1925	Drama|Romance|W		USA	Not Rated	151	1.33	245000		King Vidor	John Gilbe	Renée Ad	Claire Ada
5	Metropoli	1927	Drama|Sc	German	Germany	Not Rated	145	1.33	6000000	26435	Fritz Lang	Brigitte He	Gustav Fr	Rudolf Kle
6	Pandora's	1929	Crime|Dra	German	Germany	Not Rated	110	1.33		9950	Georg Wil	Louise Bro	Francis Le	Fritz Kortn
7	The Broad	1929	Musical|R	English	USA	Passed	100	1.37	379000	2808000	Harry Bea	Anita Page	Bessie Lov	Charles Ki
8	Hell's Ang	1930	Drama|W	English	USA	Passed	96	1.2	3950000		Howard H	Jean Harlo	Marian Ma	James Hal
9	A Farewel	1932	Drama|Ro	English	USA	Unrated	79	1.37	800000		Frank Bor	Gary Coop	Helen Hay	Adolphe M
10	42nd Stree	1933	Comedy|M	English	USA	Unrated	89	1.37	439000	2300000	Lloyd Bac	Ginger Ro	Dick Powe	George Br
11	She Done	1933	Comedy|M	English	USA	Approved	66	1.37	200000		Lowell Sh	Mae West	Gilbert Ro	Louise Be

Like this, one can use `XlsxWriter` to apply various formatting to the output Excel file.

Conclusion

place in the data analysis workflow and can be very great companion tools. As we demonstrated, pandas can do a lot of complex data analysis and manipulations, which depending on your need and expertise, can go beyond what you can achieve if you are just using Excel. One of the major benefits of using Python and pandas over Excel is that it helps you automate Excel file processing by writing scripts and integrating with your automated data workflow. Pandas also has excellent methods for reading all kinds of data from Excel files. You can export your results from pandas back to Excel too if that's preferred by your intended audience.

On the other hand, Excel is a such a widely used data tool, it's not a wise to ignore it. Acquiring expertise in both pandas and Excel and making them work together gives you skills that can help you stand out in your organization.

Harish Garg

Entrepreneur, Technical Trainer, and Lead Software Developer with extensive experience in Data Science, Python, Web, and Mobile Development. Passionate about Data Science and Artificial Intelligence.

Read More

— Dataquest —
Pandas

Adding Axis Labels to Plots With pandas

Pandas Concatenation Tutorial

Regular Expressions for Data Scientists

See all 23 posts →

Pandas Concatenation Tutorial

In this tutorial, we walk through several methods of combining data using pandas.


SUNISHCHAL DEV

PYTHON

Regular Expressions for Data Scientists

In this tutorial, learn how to use regular expressions and the pandas library to manage large data sets.


ALEX YANG