To explore what kind of data science books are available out in Amazon:

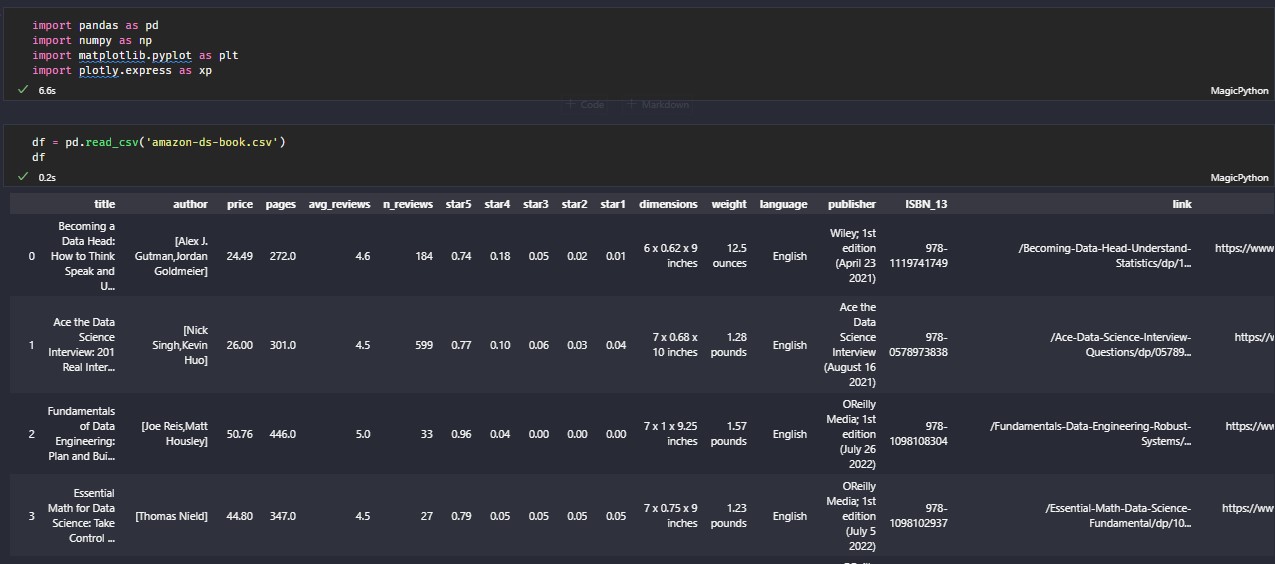
I found a dataset on Kaggle that contains nearly 1000 data science books on Amazon including the title, author, ratings, number of pages and other features.

We’ll be doing three levels of data analysis, Beginner, intermediate and advanced.

In level 1, we’ll do simple Exploratory Data Analysis to answer questions such as;

1. More expensive books have better reviews?
2. Is it always true that longer pages are more expensive or what are the best python related books and machine learning books based on review stars?

In level 2, we’ll do Cluster analysis on book titles.  And we want to find out what are the main types or main categories of data science books out there based on book titles. We’ll be using K-means for clustering the book titles and in order to do that for text data we use the NLP technique called TF-IDF [Term frequency-inverse document frequency] to convert the text into numeric features.

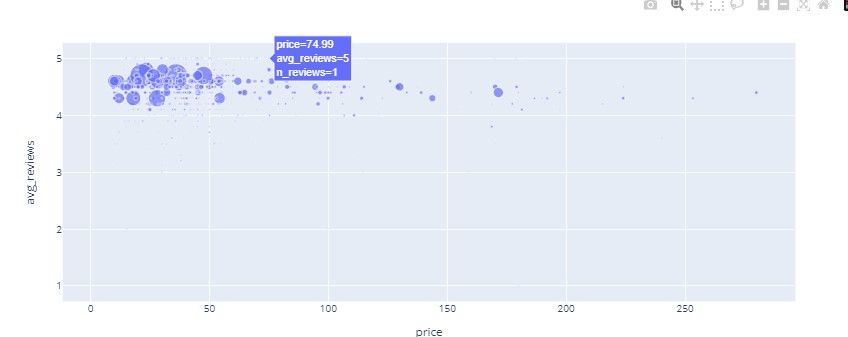


After importing relevant libraries and loading the data out the Data Frame here we can see a beautiful overview of the data including the distribution of the data in each column and also the percentages of missing values in each column which is quite handy. We have 946 book titles containing many data science and machine learning books. As we can see, we have title, author, price, pages etc.

Level 1: Exploratory data analysis to answer a question like do more expensive books have better reviews.

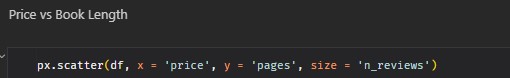
For that what we can do is to create a scatter plot that plots the book price against the reviews. In such a case, we’re using Plotly Express to quickly create the scatter plot with the x-axis being the price and the y-axis being the reviews and adding the size of the number of reviews.

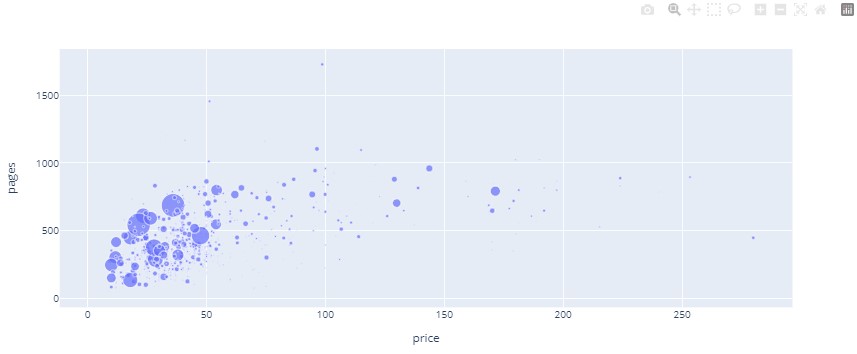




After doing data visualization with scatter plot we can see that there’s no clear relationship between price and average reviews. So, good books can also be very cheap or they can be very expensive.

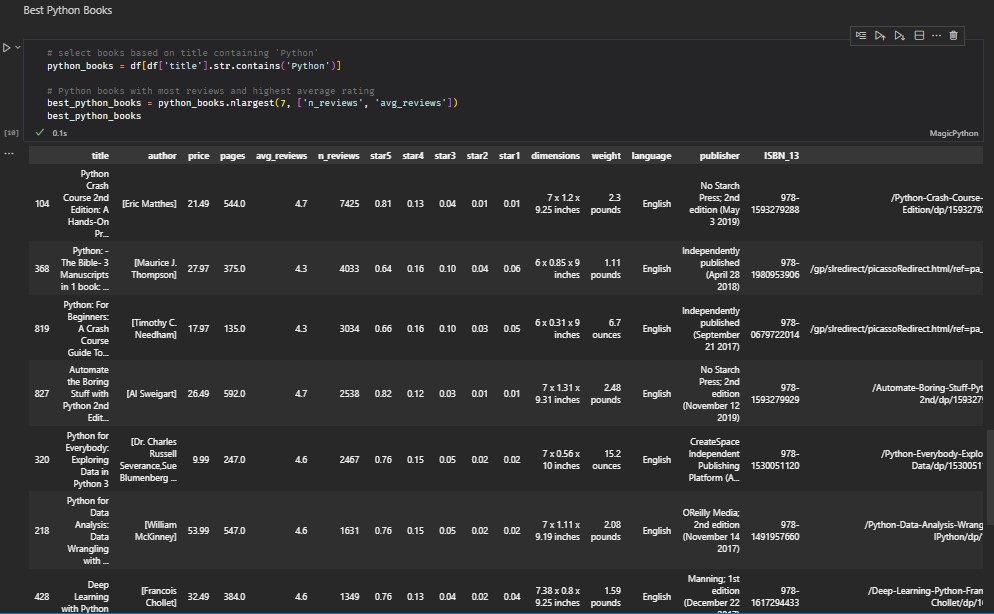
Similarly, if we do the same exercise but plotting the price against the number of pages. We can see that there’s some positive correlation. Longer books tend to be a bit more expensive which makes total sense because it costs more time and effort to write those books.



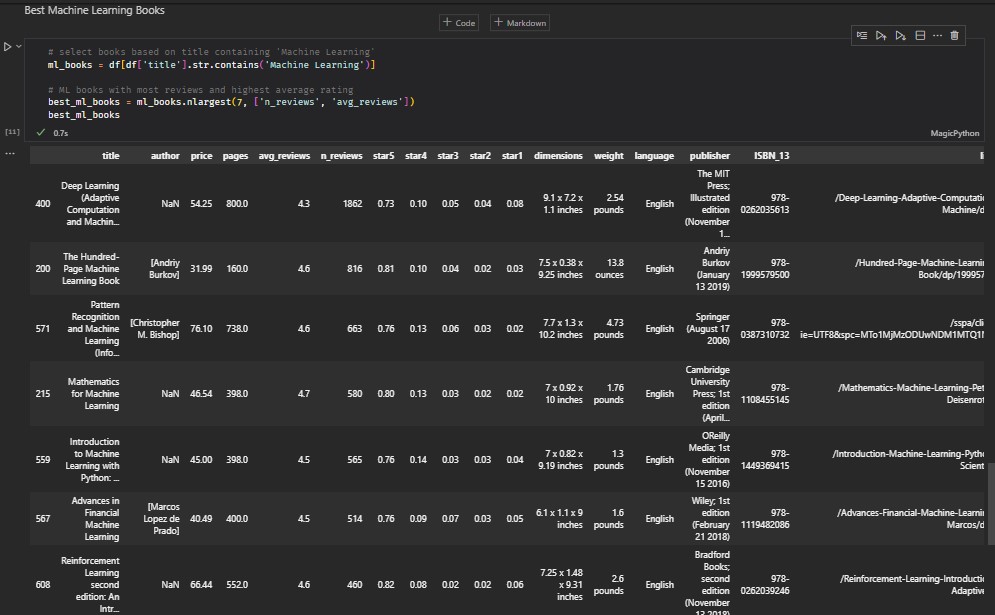


Now sort out which are the best Python books and Machine Learning books.

* We can do this by a simple rule-based method so select all the books with the titles containing Python.
* And now select the Python books with the most reviews and have the highest average rating. We can do this by using n largest function (nlargest ()). For instance, if we want to only select the top seven books and here, we can select the columns that we want to use to sort the data. The reason why we want to sort data on both number of reviews and average reviews is because some books might be not very popular and they have very few reviews but they are very good reviews so these cases might be more prone to bias and in these 7 best Python books we have:



Best Machine Learning books:



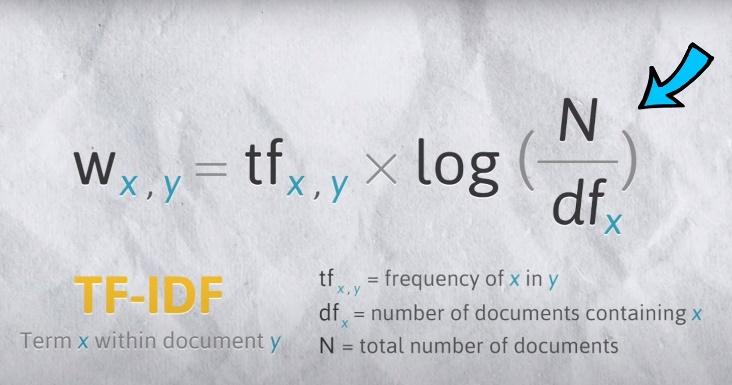
Level 2: Cluster analysis on book titles

On the 2nd level, we are doing text clustering with TF-IDF and K-means clustering to answer the question: what are the main categories of data science books in this dataset? And with this approach we can automatically classify the books into different categories without having to go through the 1000 book titles by ourselves and come up with the classification roles. As we know that clustering is an unsupervised ML technique, it is a task of dividing the population or data points into a number of groups such that the data points are more similar to other data points in the same group and thus similar to the data points in other groups.

This method is very useful for understanding the structure of the data and there are many clustering algorithms out there. However, for this project we use one of the most simple and popular clustering analysis K-means. K-means is an iterative algorithm. We just randomly initialize the centroids or the centers for our clusters and then we simply assign each data point to the nearest centroids. We calculate the mean of the location of all data points that belong to each cluster and then move the location of the centroids to that average location. This is the first iteration where data points are not adjusted center. So, after several iterations the centroids will naturally be shifted towards the true clusters data and at some point the position of the centroids will stop changing.

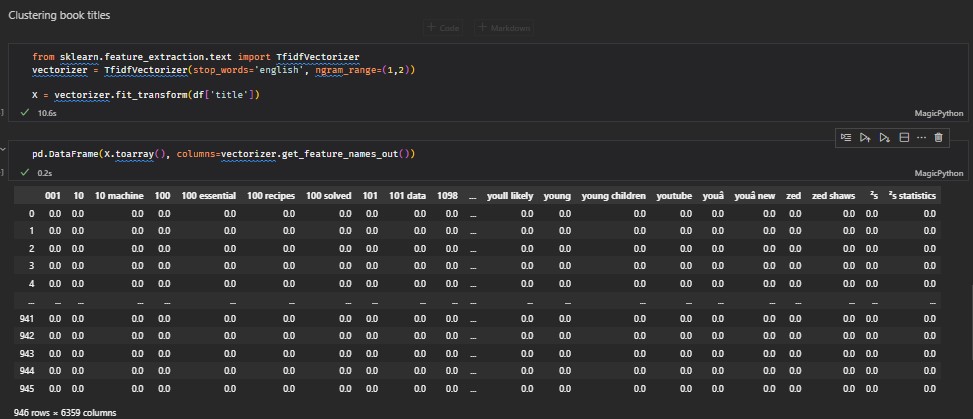
Before applying this technique, we have a small problem. In our case, we have clustering text data which is a bit trickier than the usual numeric data. so, first we need to convert our text data to numeric data. and that can be used and understood by machine. In a fancy term, this is called text vectorization.

We use TF-IDF which is a popular frequency-based vectorization method that is simple but far better than the simple counts vectorizer. Our aim is to convert text and, in this case, book titles into an array of numbers and would contain the value for each word that ever appears in one of the book titles. Zero means the word doesn’t occur at all in this current title, low value means the word exists in the title but is not very important for the overall meaning of the title and the higher value means the word is more important for the overall meaning. How does IDF determine those values, it does this by multiplying two factors together term frequency and inverse document frequency.



The idea is simple, the first parameter is local. We calculate the frequency of the word within the documents in this case within the book title. In the book title the best python book for python lovers the word python occurs twice while other words occur only once. So, python has a term frequency of two and others have one. Another parameter is global. How rare the word is across all the documents. We do this by taking the inverse document frequency of the words in all the documents. For example, if we have another book title named the best r book, we can say that the word best is quite popular. It appears in both book titles so we can assume that words like this are probably less important. So, multiply these two terms [tf and log(n/df)] would give a quite good measure for the importance of the words and it assigns higher value to more descriptive and informative words and lower value to the unimportant but maybe more commonly occurring words in the English Language.

Let’s import the TF-IDF vectorizer from scikit-learn. And then we can initiate the vectorizer object. Here the stop words using the English language stop words like you, me, at, the etc. These are the words that are generally not interesting for Nob tasks and let’s also specify the ngram range here from one to two because in our book titles we usually see words like data analysis or machine learning or data science etc those are the words that have two words so the n-gram here basically says how many we want to put together at time and then we just use the vectorizer to fit and transform the title into vector X.



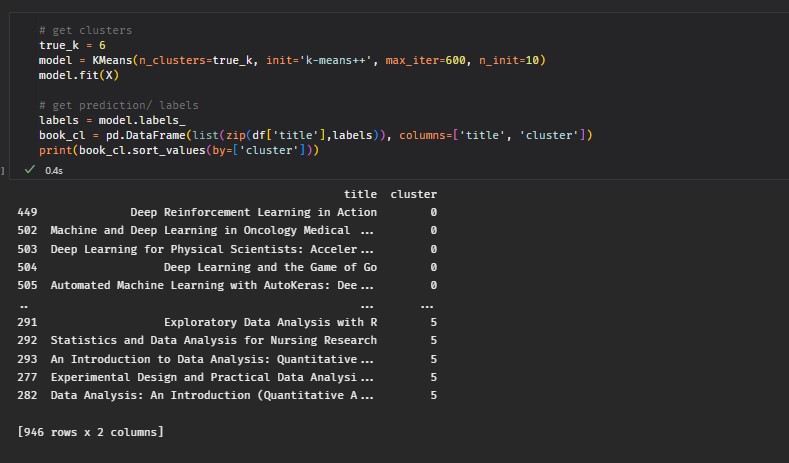
Here is just basically a huge array that represents each book title.

Now to implement K-means we also import K-means from scikit-learn and a small challenge with K-means is that we have to specify the number of clusters that we want to create and we don’t know in advance what is the optimal number of clusters so we have to find it out ourselves so the strategy is that we’ll just assume a minimum number of clusters let’s say two clusters and the maximum number of clusters let’s say 10 and for each value in those potential numbers of clusters will perform K-means cross string and then we’ll calculate the sum of squared distance or inertia as how it’s called in the scikit-learn library. It is the sum of squared distance of the data points to their closest cluster center and we’ll append those sums of squared distance into an array.



After we’ve performed all the clustering for each possible number of clusters. We plot out the sum of squared distance. We can see that the more clusters we have the lower the sum of squared distance is. It might make sense because in extreme cases if we make one cluster for each individual data point then the sum squared distance would be technically zero.

With the elbow method, we want to find the optimal number of clusters that have the lowest sum of squared distance. We can see that around cluster 6, the sum of squared distance starts declining more slowly so we can expect that 6 is the optimal number of clusters. However, K-means is not deterministic so if we run the code, we might see slightly different graphs. For now, we chose 6 is the optimal number of clusters and then we’ll pass that into the K-means model to get the prediction labels it’s quite simple we just get the labels out of the model and then zip the labels with the title in the data and here we can see that which cluster the books belong to.



These are the top terms per cluster.