**Content-Based Recommender**

 We will build a system that recommends properties that are similar to a particular property. To achieve this, we will compute pairwise cosine similarity scores for all properties based on their attributes and recommend property based on that similarity score threshold.

There are several similarity metrics that we can use for this, such as the manhattan, euclidean, the Pearson, and the cosine similarity scores. Again, there is no right answer to which score is the best. Different scores work well in different scenarios, and it is often a good idea to experiment with different metrics and observe the results.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two properties. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate Mathematically, it is defined as follows:

A picture containing chart

Description automatically generated

Calculating the dot product between each vector will directly give you the cosine similarity score. Therefore, you will use sklearn's linear\_kernel() instead of cosine\_similarities() since it is faster. Code will look something like this

Text

Description automatically generated

Then we are going to define a function that takes in a property as an input and outputs a list of the 10 most similar properties. These are the steps that we need to follow:

* Get the zipid of the property.
* Get the list of cosine similarity scores for that particular property with all properties. Convert it into a list of tuples where the first element is its position, and the second is the similarity score.
* Sort the aforementioned list of tuples based on the similarity scores; that is, the second element.
* Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
* Return the titles corresponding to the indices of the top elements.

# API Contracts

1. UI to Backend
   1. POST view listing
      1. Every time user click on listing it is stored in the backend
   2. GET recommended listings
      1. UI calls GET recommended listings, then backend gives recommended listing to be populated in the UI
2. Backend to the spark job
   1. Submit spark job
      1. Writing to KAFKA
   2. Listener in the backend
      1. Receives messages from the SPARK job