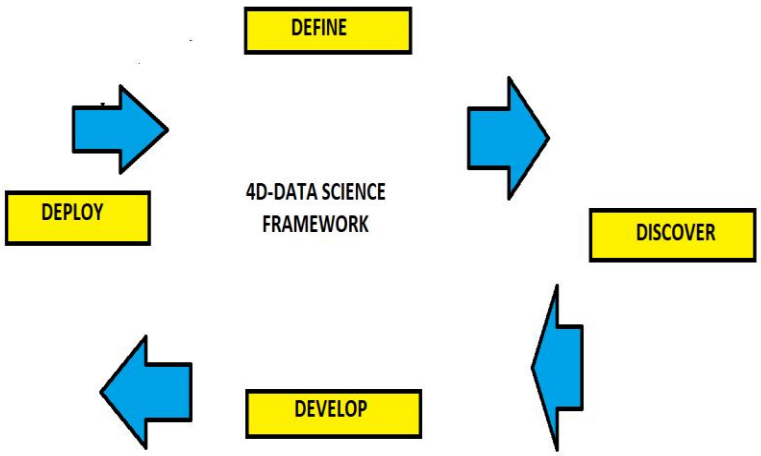
BOOK RECOMMENDATION SYSTEM

Any Data Science problem to be solved,requires a frame work.Here we have adopted the 4-D Data Science Framework.Its has 4 steps:

1) Define
2) Discover
3) Develop
4) Deploy



Step1: DEFINE:

Creating a Book Recommendation system based on the content and title of the books.

Step2: DISCOVER:

Obtaining data, cleaning data, exploring data, establish baseline outcomes and hypothesing the solutions.

In [13]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import RegexpTokenizer
import re
import string
import random
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
%matplotlib inline
```

In [14]:
```
# Reading the file
df = pd.read_csv("goodreads.csv", encoding = "ISO-8859-1", engine='python')
```

We have imported the required libraries and as the first step of “DISCOVER” we have obtained the data using panda’s read func().

```
In [15]: #Reading the first five records
         df.head()
```

Out[15]:

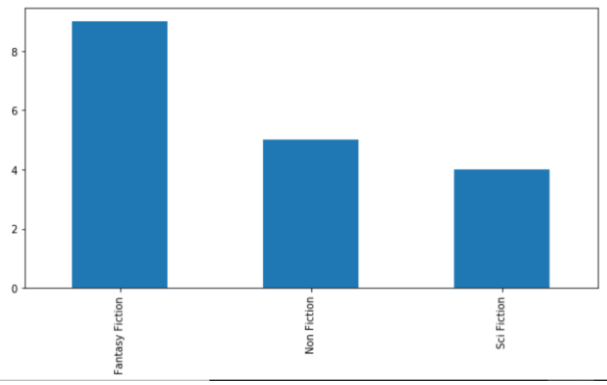
	Id	title	genre	Unnamed: 3	authors	Rating	Desc	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	1	Harry Potter and the Half-Blood Prince (Harry ...	Fantasy Fiction	NaN	J.K. Rowling	4.57	The war against Voldemort is not going well; e...	NaN	NaN	NaN
1	2	Harry Potter and the Order of the Phoenix (Har...	Fantasy Fiction	NaN	J.K. Rowling	4.50	There is a door at the end of a silent corrido...	NaN	NaN	NaN
2	3	Harry Potter and the Sorcerer's Stone (Harry P...	Fantasy Fiction	NaN	J.K. Rowling	4.47	Harry□s perfectly normal life at number 4 priv...	NaN	NaN	NaN
3	4	Harry Potter and the Chamber of Secrets (Harry...	Fantasy Fiction	NaN	J.K. Rowling	4.42	Harry Potter is about to start his second year...	NaN	NaN	NaN
4	5	Harry Potter and the Prisoner of Azkaban (Harr...	Fantasy Fiction	NaN	J.K. Rowling	4.57	Harry Potter and the Prisoner of Azkaban is th...	NaN	NaN	NaN

```
In [16]: #Checking the shape of the file
         df.shape
Out[16]: (18, 10)
```

Due to the processing restrictions , we have used only 18 rows of data, which would be sufficient for a prototype data model. The cleaning of the data for such small dataset was done by hardwiring.

The next step is of Exploring the Data

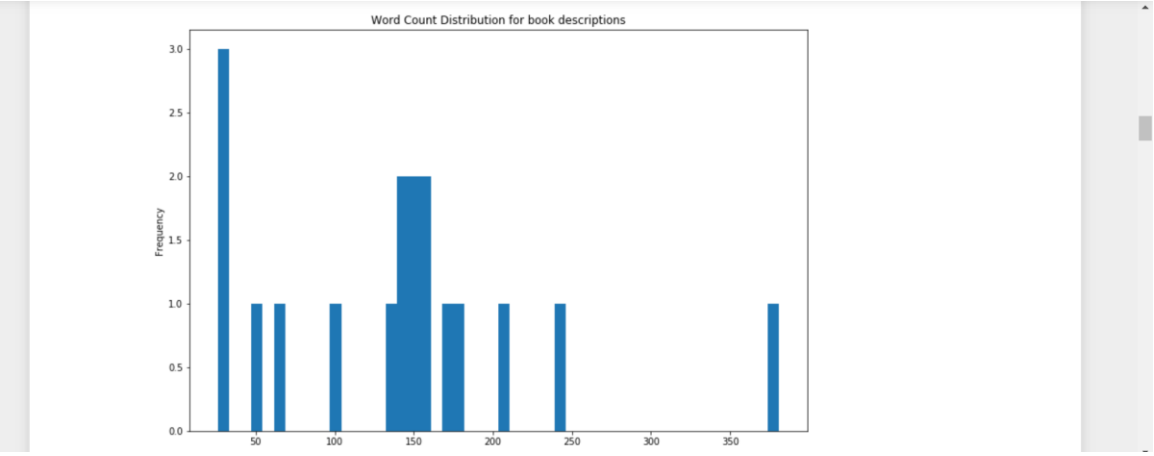
```
In [17]: #Exploratory Data Analysis

In [18]: # Genre distribution
         df['genre'].value_counts().plot(x = 'genre', y ='count', kind = 'bar', figsize = (10,5)  )
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x228da9ce488>
```



We first check what are the genre’s included in the dataset, based on the above graph we can say that there are 3 genres: Fantasy Fiction, Non-Fiction and Sci Fiction

```
In [64]: #Book description – Word count distribution

In [107]: # Calculating the word count for book description
          df['word_count'] = df['Desc'].apply(lambda x: len(str(x).split()))
          # Plotting the word count
          df['word_count'].plot(
              kind='hist',
              bins = 50,
              figsize = (12,8),title='Word Count Distribution for book descriptions')
Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0x228e20bf608>
```

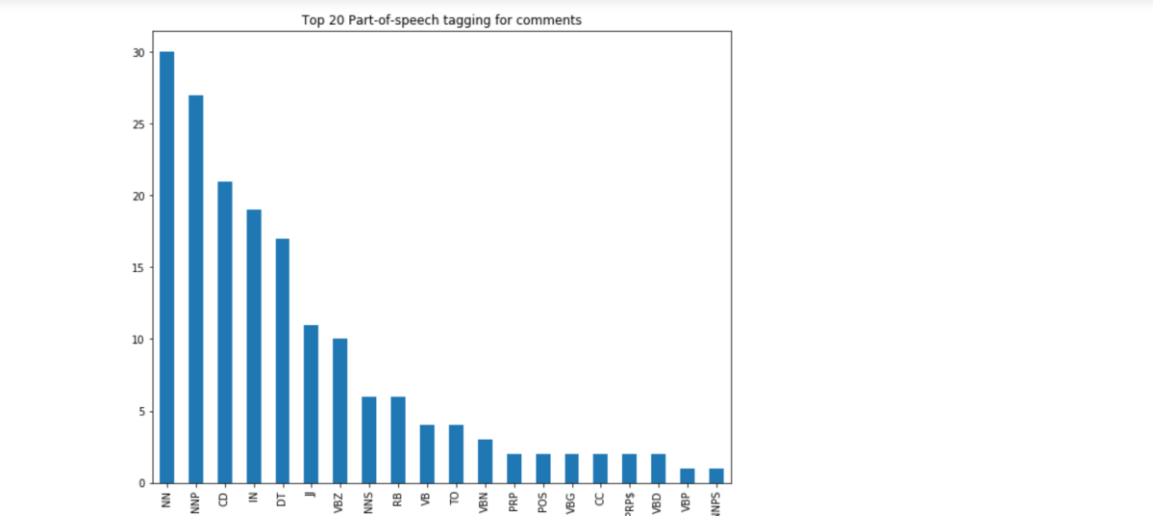
Next to check the words on description. We need to know how long the content is base on the word count we plot a graph.



Now as we know, how long is the word count for the description is , we download nltk package and begin the match case of words with the corpus library in the package we downloaded.

Using the function of TextBlob, we find the “Top 20 Speech tags” which are present in the description and put in a graph to see, how many times each tag is repeated.

```
In [66]: #import nltk
         #nltk.download()
         #It will open a page for corpus download.Download all the packages

In [67]: #The distribution of top part-of-speech tags in the book descriptions
         from textblob import TextBlob
         blob = TextBlob(str(df['Desc']))
         pos_df = pd.DataFrame(blob.tags, columns = ['word' , 'pos'])
         pos_df = pos_df.pos.value_counts()[:20]
         pos_df.plot(kind = 'bar', figsize=(10, 8), title = "Top 20 Part-of-speech tagging for comments")

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x228e15f6ac8>
```



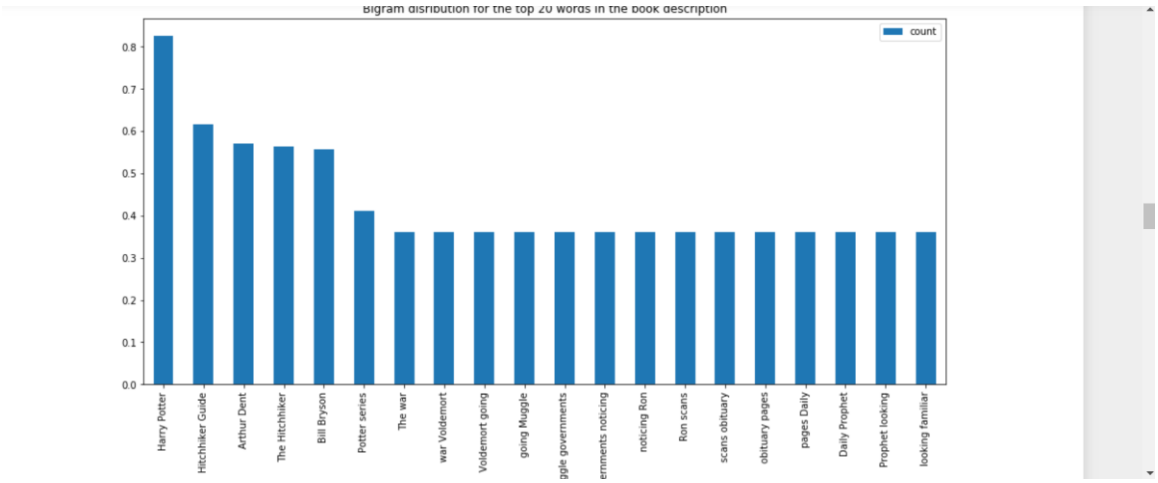
Since, we have established baseline outcomes, its time to hypothise the solution.

As, we can see through the exploratory data analysis, that desc can provide the idea of content of book, we can use it for recommending books with similar content.

Step3:DEVELOP

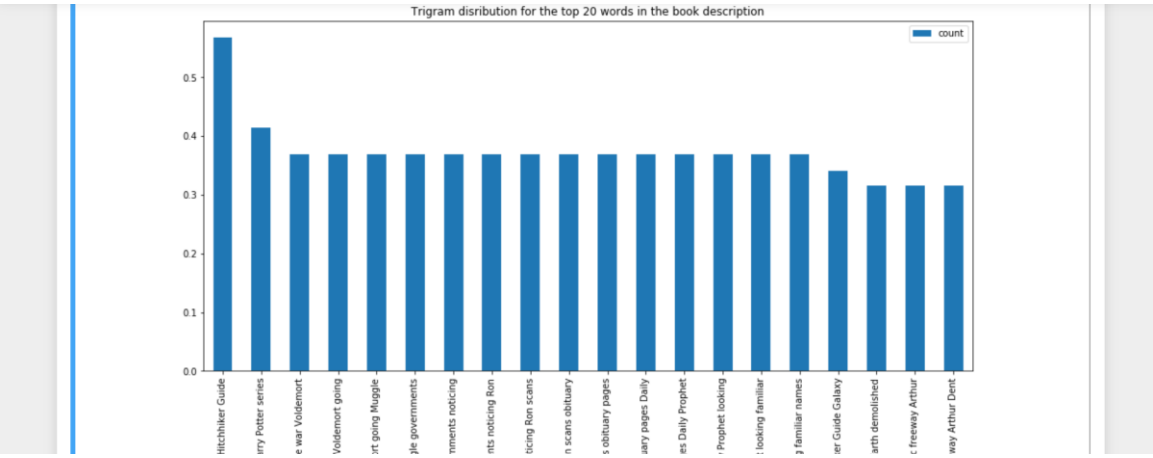
Based on the discoveries we made, we know we have to use content and title of books for recommendation, thus we vectorize the desc using n- gram formula, we check both bi-gram and tri-gram using TF-IDF.

```
In [68]: #Converting text descriptions into vectors using TF-IDF using Bigram
         tf = TfidfVectorizer(ngram_range=(2, 2), stop_words='english', lowercase = False)
         tfidf_matrix = tf.fit_transform(df['Desc'])
         total_words = tfidf_matrix.sum(axis=0)
         #Finding the word frequency
         freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
         freq =sorted(freq, key = lambda x: x[1], reverse=True)
         #converting into dataframe
         bigram = pd.DataFrame(freq)
         bigram.rename(columns = {0:'bigram', 1: 'count'}, inplace = True)
         #Taking first 20 records
         bigram = bigram.head(20)

In [69]: #Plotting the bigram distribution
         bigram.plot(x ='bigram', y='count', kind = 'bar', title = "Bigram disribution for the top 20 words in the book description", figs

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x228e1c601c8>
```



Based on the Bi-Gram , we see the couple of words with highest frequency in the whole dataset within this graph.

```
In [70]: #Converting text descriptions into vectors using TF-IDF using Trigram
         tf = TfidfVectorizer(ngram_range=(3, 3), stop_words='english', lowercase = False)
         tfidf_matrix = tf.fit_transform(df['Desc'])
         total_words = tfidf_matrix.sum(axis=0)
         #Finding the word frequency
         freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
         freq =sorted(freq, key = lambda x: x[1], reverse=True)
         #converting into dataframe
         trigram = pd.DataFrame(freq)
         trigram.rename(columns = {0:'trigram', 1: 'count'}, inplace = True)
         #Taking first 20 records
         trigram = trigram.head(20)
         #Plotting the trigramn distribution
         trigram.plot(x ='trigram', y='count', kind = 'bar', title = "Trigram disribution for the top 20 words in the book description", f

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x228e1c86648>
```



Based on the Tri-Gram , we see triplets words with highest frequency in the whole dataset within this graph.

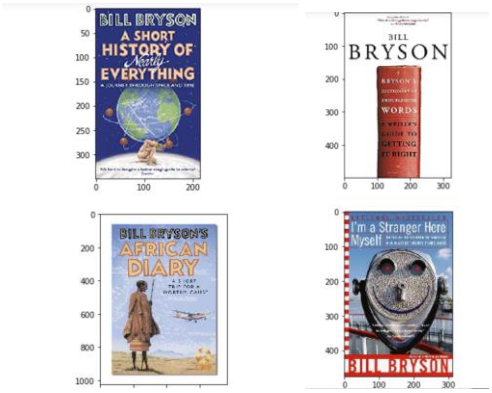
```
In [71]: #Text Preprocessing
#Cleaning the book description.
# Function for removing NonAscii characters
def _removeNonAscii(s):
    return "".join(i for i in s if  ord(i)<128)
# Function for converting into lower case
def make_lower_case(text):
    return text.lower()
# Function for removing stop words
def remove_stop_words(text):
    text = text.split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)
    return text
# Function for removing punctuation
def remove_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    text = tokenizer.tokenize(text)
    text = " ".join(text)
    return text
#Function for removing the html tags
def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)
# Applying all the functions in description and storing as a cleaned_desc
df['cleaned_desc'] = df['Desc'].apply(_removeNonAscii)
df['cleaned_desc'] = df.cleaned_desc.apply(func = make_lower_case)
df['cleaned_desc'] = df.cleaned_desc.apply(func = remove_stop_words)
```

Since, the description, is written by different people, before we apply it to the algorithm it needs to be cleaned, thus we remove, all upper case, punctuation marks, stop words and html from the description.

Now applying the cleaned data to below model where we used vectors and apply cosine similarity function to find the similarities, between two titles and two descriptions.

```
In [72]: from sklearn.metrics.pairwise import cosine_similarity

In [73]: #Recommendation based on book title
         # Function for recommending books based on Book title. It takes book title and genre as an input.
         def recommend(title, genre):

             # Matching the genre with the dataset and reset the index
             data = df.loc[df['genre'] == genre]
             data.reset_index(level = 0, inplace = True)

             # Convert the index into series
             indices = pd.Series(data.index, index = data['title'])

             #Converting the book title into vectors and used bigram
             tf = TfidfVectorizer(analyzer='word', ngram_range=(2, 2), min_df = 1, stop_words='english')
             tfidf_matrix = tf.fit_transform(data['title'])

             # Calculating the similarity measures based on Cosine Similarity
             sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

             # Get the index corresponding to original_title

             idx = indices[title]
         # Get the pairwsie similarity scores
             sig = list(enumerate(sg[idx]))
         # Sort the books
             sig = sorted(sig, key=lambda x: x[1], reverse=True)
         # Scores of the 5 most similar books
             sig = sig[1:6]
         # Book indicies
             movie_indices = [i[0] for i in sig]

             # Top 5 book recommendation
             rec = data[['title', 'url']].iloc[movie_indices]

             # It reads the top 5 recommend book url and print the images

             for i in rec['url']:
                 response = requests.get(i)
                 img = Image.open(BytesIO(response.content))
                 plt.figure()
                 print(plt.imshow(img))

In [80]: recommend("In a Sunburned Country", "Non Fiction")
```

Here we recommend, book by title, thus we gave input of “In a sunburned country” which is a non - fiction and authored by “Bill Bryson” and as in output we can see that it recommended 4 other book which is a non-fiction and written by the same author.



Step4: DEPLOY:

As the part discovering we discovered that this system can work on big data if it is implemented on cloud. Due to processing restrictions we were only able to make a Prototype based model and thus, cannot be deployed at this moment.