**Prudent AI TECHNICAL ASSESSMENT**

**TASK 1**

1. To perform EDA for the given Books dataset.
2. To build a system to recommend more books to the reader based on a book already selected.

**Import required libraries and load the dataset for performing Exploratory Data Analysis**

import pandas as pd

import numpy as np

import seaborn as sns

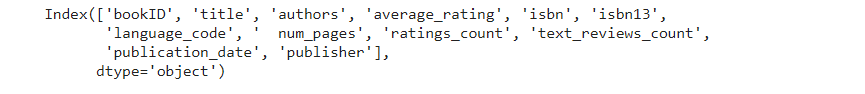
import matplotlib.pyplot as plt

%matplotlib inline

df = pd.read\_csv('books.csv', error\_bad\_lines=False)

df.head()

df.columns



**Data pre-processing**

Since the column ‘ num\_pages’ has extra space in starting, so we rename the column.

df.rename(columns={'  num\_pages':'num\_pages'}, inplace=True)

print("Dataset contains {} rows and {} columns".format(df.shape[0], df.shape[1]))

Dataset contains 11123 rows and 12 columns

Changing the author name by J.K, Rowling, for the sake of simplicity.

df.replace(to\_replace='J.K. Rowling-Mary GrandPré', value = 'J.K. Rowling', inplace=True)

df.replace(to\_replace='J.K. Rowling/Mary GrandPré', value = 'J.K. Rowling', inplace=True)

**Check whether the dataset has any null or missing or duplicate values**

df.isnull().sum()

df.isna().sum()

print('Number of duplicate rows: ',df.duplicated().sum())



Since there are no missing or null or duplicate rows, we don't need to drop any of the columns from the dataset. So let’s find whether there are any outliers present in the data.

**Finding Outliers**

plt.figure(figsize=(30,5))

sns.boxplot(x=df['average\_rating'],palette = 'colorblind')

plt.figure(figsize=(30,10))

sns.boxplot(x=df['ratings\_count'],palette = 'colorblind')





So, we can see that there are no outliers in average\_rating and ratings\_count. So let’s now explore with the data visually.

**Finding the books with most occurances in the list.**

sns.set\_context('poster')

plt.figure(figsize=(20,15))

book = df['title'].value\_counts()[:20]

rating = df.average\_rating[:20]

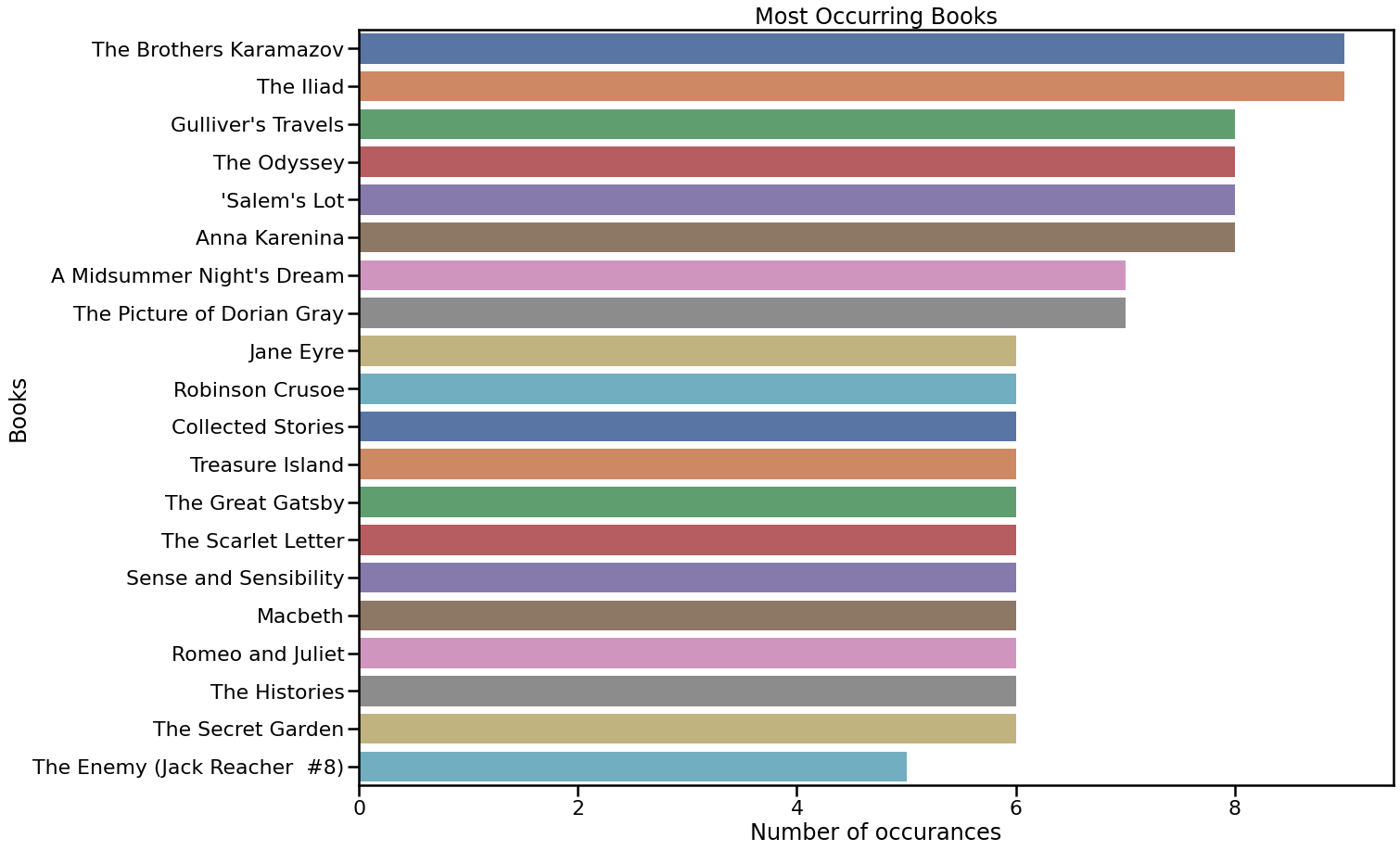
sns.barplot(x = book, y = book.index, palette='deep')

plt.title("Most Occurring Books")

plt.xlabel("Number of occurances")

plt.ylabel("Books")

plt.show()



We can see that,The lliad and The brothers karamazov are the books with most occurances. This shows that the books have aged well.

**Finding the most frequent language used**

sns.set\_context('paper')

plt.figure(figsize=(15,10))

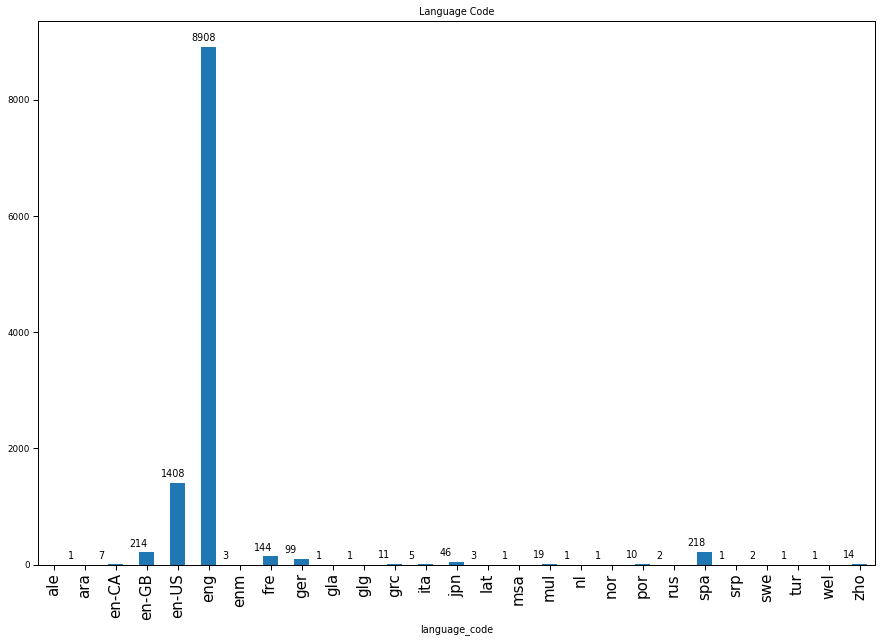
ax = df.groupby('language\_code')['title'].count().plot.bar()

plt.title('Language Code')

plt.xticks(fontsize = 15)

for p in ax.patches:

    ax.annotate(str(p.get\_height()), (p.get\_x()-0.3, p.get\_height()+100))



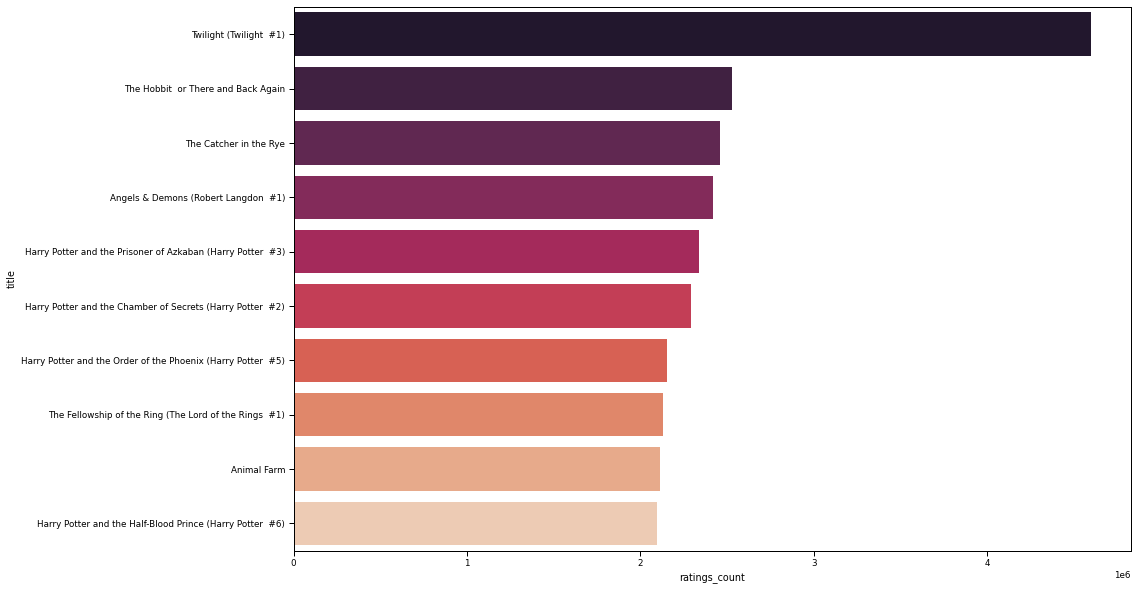
Clearly we can see that english is the most frequent language.

**Finding book that has got the most number of ratings**

most\_rated = df.sort\_values('ratings\_count', ascending = False).head(10).set\_index('title')

plt.figure(figsize=(15,10))

sns.barplot(most\_rated['ratings\_count'], most\_rated.index, palette='rocket')



Twilight(Twilight #1) is the most rated book. But no other book from twilight series can be seen in the list.

**Finding the authors with maximum number of books**

sns.set\_context('talk')

most\_books = df.groupby('authors')['title'].count().reset\_index().sort\_values('title', ascending=False).head(10).set\_index('authors')

plt.figure(figsize=(15,10))

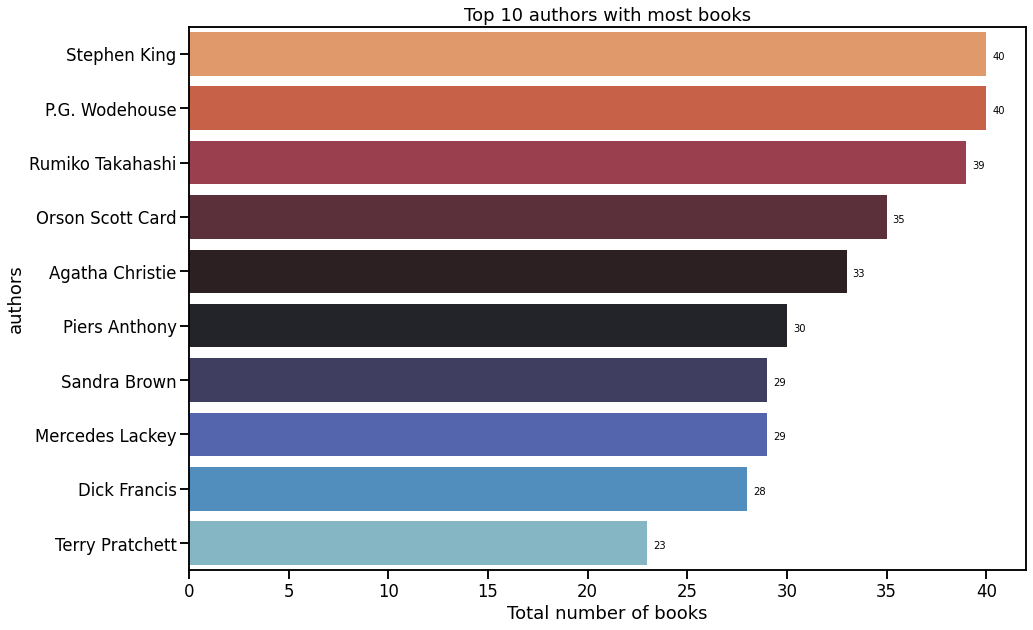
ax = sns.barplot(most\_books['title'], most\_books.index, palette='icefire\_r')

ax.set\_title("Top 10 authors with most books")

ax.set\_xlabel("Total number of books")

for i in ax.patches:

    ax.text(i.get\_width()+.3, i.get\_y()+0.5, str(round(i.get\_width())), fontsize = 10, color = 'k')



**Finding the most frequent publishing house**

sns.set\_context('talk')

most\_books = df.groupby('publisher')['title'].count().reset\_index().sort\_values('title', ascending=False).head(10).set\_index('publisher')

plt.figure(figsize=(10,5))

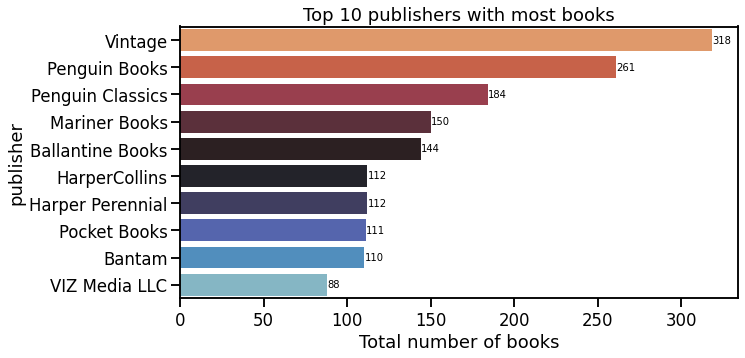
ax = sns.barplot(most\_books['title'], most\_books.index, palette='icefire\_r')

ax.set\_title("Top 10 publishers with most books")

ax.set\_xlabel("Total number of books")

for i in ax.patches:

    ax.text(i.get\_width()+.3, i.get\_y()+0.5, str(round(i.get\_width())), fontsize = 10, color = 'k')



df['Ratings\_Dist'] = segregation(df)

ratings\_pie = df['Ratings\_Dist'].value\_counts().reset\_index()

labels = ratings\_pie['index']

colors = ['lightblue','darkmagenta','coral','bisque', 'black']

percent = 100.\*ratings\_pie['Ratings\_Dist']/ratings\_pie['Ratings\_Dist'].sum()

fig, ax1 = plt.subplots()

ax1.pie(ratings\_pie['Ratings\_Dist'],colors = colors,

        pctdistance=0.85, startangle=90, explode=(0.05, 0.05, 0.05, 0.05, 0.05))

#Draw a circle now:

centre\_circle = plt.Circle((0,0), 0.70, fc ='white')

fig1 = plt.gcf()

fig1.gca().add\_artist(centre\_circle)

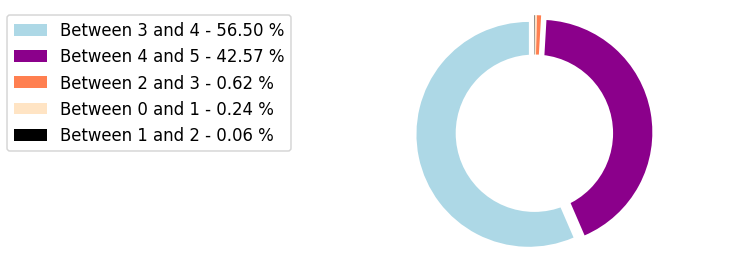
#Equal Aspect ratio ensures that pie is drawn as a circle

plt.axis('equal')

plt.tight\_layout()

labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(labels, percent)]

plt.legend( labels, loc = 'best',bbox\_to\_anchor=(-0.1, 1.),)



**The recommender system**

Here, we used KMeans Clustering because it is a type of unsupervised learning which groups unlabelled data.

**Elbow plot**

First we would find the K value using Elbow curve method (for finding the number of clusters for the data).

X = data

distortions = []

for k in range(2,30):

    k\_means = KMeans(n\_clusters = k)

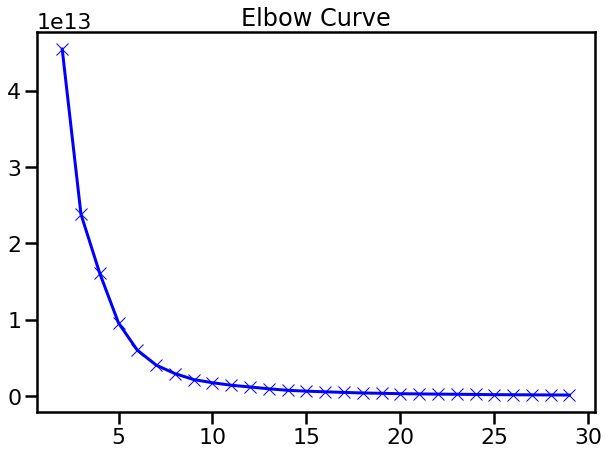
    k\_means.fit(X)

    distortions.append(k\_means.inertia\_)

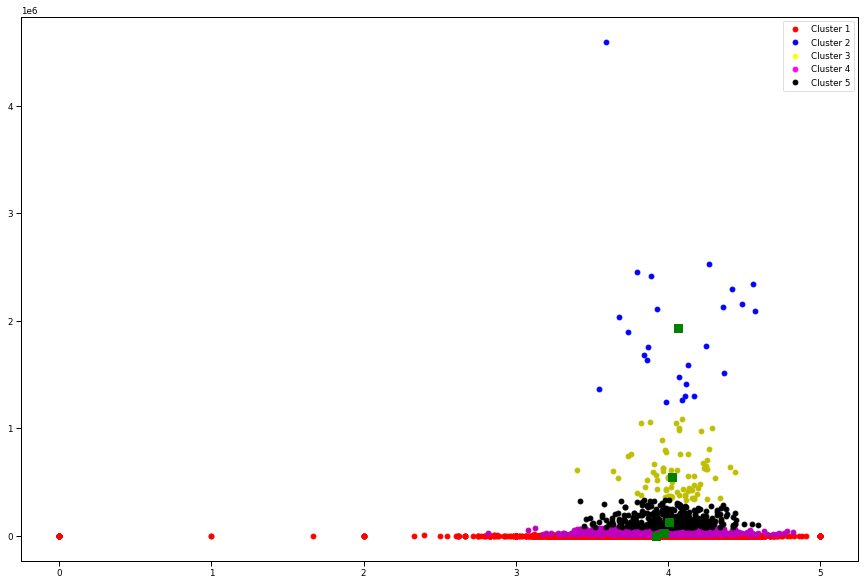
fig = plt.figure(figsize=(10,7))

plt.plot(range(2,30), distortions, 'bx-')

plt.title("Elbow Curve")



From the above plot, we can see that the elbow lies around the value K=5, so that's what we will attempt it with.



From the above plot, now we can see that once the whole system can be classified into clusters. As the count increases, the rating would end up near the cluster given above. The green squares are the centroids for the given clusters.

As the rating count seems to decrease, the average rating seems to become sparser, with higher volatility and less accuracy.

**Get index from Title**

def get\_index\_from\_name(name):

    return df[df["title"]==name].index.tolist()[0]

all\_books\_names = list(df.title.values)

**Get ID from partial name (Because not everyone can remember all the names)**

def get\_id\_from\_partial\_name(partial):

    for name in all\_books\_names:

        if partial in name:

            print(name,all\_books\_names.index(name))

**Print the similar books from the feature dataset. (This uses the Indices metric from the nearest neighbors to pick the books)**

def print\_similar\_books(query=None,id=None):

    if id:

        for id in indices[id][1:]:

            print(df.iloc[id]["title"])

    if query:

        found\_id = get\_index\_from\_name(query)

        for id in indices[found\_id][1:]:

            print(df.iloc[id]["title"])

