COMP5721M: Programming for Data Science

Coursework 3: Data Analysis Project

Last modified: 4 December 2023

Data analysis and bestseller prediction based on Amazon Kindle Books Dataset

- Qiaoyu Xue, sc23qx@leeds.ac.uk (mailto:sc23qx@leeds.ac.uk)
- Jingya Zhou, ml22j7z@leeds.ac.uk (mailto:ml22j7z@leeds.ac.uk)
- Shutian Wang, sc23sw2@leeds.ac.uk (mailto:sc23sw2@leeds.ac.uk)
- Xinyun Chen, ml222xc@leeds.ac.uk (mailto:ml222xc@leeds.ac.uk)

Project Plan

The Data (10 marks)

Description of data to be used

The data is sourced from the Kindle dataset on Kaggle (click to see) (https://www.kaggle.com/datasets/asaniczka/amazon-kindle-books-dataset-2023-130k-books), comprising over 16 columns and a total of more than 130,000 records.

There are 16 columns in total, and each column's meaning is as follows:

- asin: Product ID from Amazon. (type: str)
- title: Title of the book. (type: str)
- author: Author(s) of the book. (type: str)
- soldby: Seller(s) of the book. (type: str)
- imgUrl: URL of the book cover image. (type: str)
- productURL: URL of the book. (type: str)
- stars: Average rating of the book. If 0, no ratings were found. (type: float)
- price: Price of the book. If 0, price was unavailable. (type: float)
- reviews: Number of reviews. If 0, no reviews were found. (type: int)
- KindleUnlimited: Whether the book is available through Kindle Unlimited. (type: bool)
- category_id: Serial id assigned to the category this book belongs to. (type: int)
- isBestSeller: Whether the book had the Amazon Best Seller status or not. (type: bool)
- isEditorsPick: Whether the book had the Editor's Pick status or not. (type: bool)
- isGoodReadsChoice: Whether the book had the Goodreads Choice status or not. (type: bool)
- publishedDate: Publication date of the book. (type: str, format: YYYY-MM-DD)
- category_name: Name of the book category. (type: str), with a total of 31 categories.

Data Quality and method we use to clean the data:

1.A large number of entries have **missing values in 'reviews' and 'publishedDate'** columns, marked as 0 and nan respectively. However, actual values are present on the official website. These missing values, accounting for approximately one-third of the data, were filled accordingly.

2.There were 400 entries with **missing author information**. We utilized web scraping to retrieve this information and subsequently filled in the missing data.

3.The dataset seems contain some outdated information. We suppose that some of the books have been discontinued, leading to non-existent web pages, which means **invalid URL** like: (click to see) (https://www.amazon.com/dp/B095J864MD) We removed these records during web scraping.

4.For some books, the Kindle Amazon **website did not provide** the **publication date**, resulting in **missing values** in this column. We deleted this small subset of books.

5.We adjusted the data granularity by merging the subsidiaries and replacing their classes with the head office name(Will be used for Objective 3, 4).

6.The "soldBy" column in this dataset contains a few missing values, and upon verification, the website does not actually provide it (we dropped the missing books and will utilize this field for Objectives 3 and 4).

7.Time series outlier： there's a book release date of 2024 that's clearly an outlier (click to see) (https://www.amazon.com/dp/B0C6SN8WRB)

The final cleaned book dataset consists of **131,593** entries.

Project Aim and Objectives (5 marks)

- Correlation:
    1. Distinguishing books based on the possession of excellent labels('isBestSeller,' 'isEditorsPick,' 'isGoodReadsChoice') and whether they are part of the Kindle Unlimited program. Then, explore the correlation between filtered books and prices and rating stars.
    2. Examine the correlation between the publication year of books, book categories, and bestsellers.
    3. Investigate whether there is a relationship in star ratings among books published in different years.
    4. Identify the changes in the total number of books with various categories published each year.
    5. Determine the most important factors affecting a book that may become a bestseller.
- Search: By using a partial name search for the author, it is possible to conduct a fuzzy search and identify their highest-rated books.

- Visualization: Applying scatter plots, radar charts, line graphs, bar charts, and interactive ipywidgets to analyze and visualize over 130,000 data entries.
- Query Answering: Users can interactively input or select conditions such as publication date, seller, price, author, and book category through widgets to obtain whether the conditions of the book are likely to become a bestseller.
- Simulation: Simulate the annual changes in the proportion of various types of books being bestsellers and not being bestsellers, meanwhile showing in the decreasing order of the number of books published from 1806 to 2023.
- Classification: Use a machine learning model to determine the likelihood of being a bestseller.

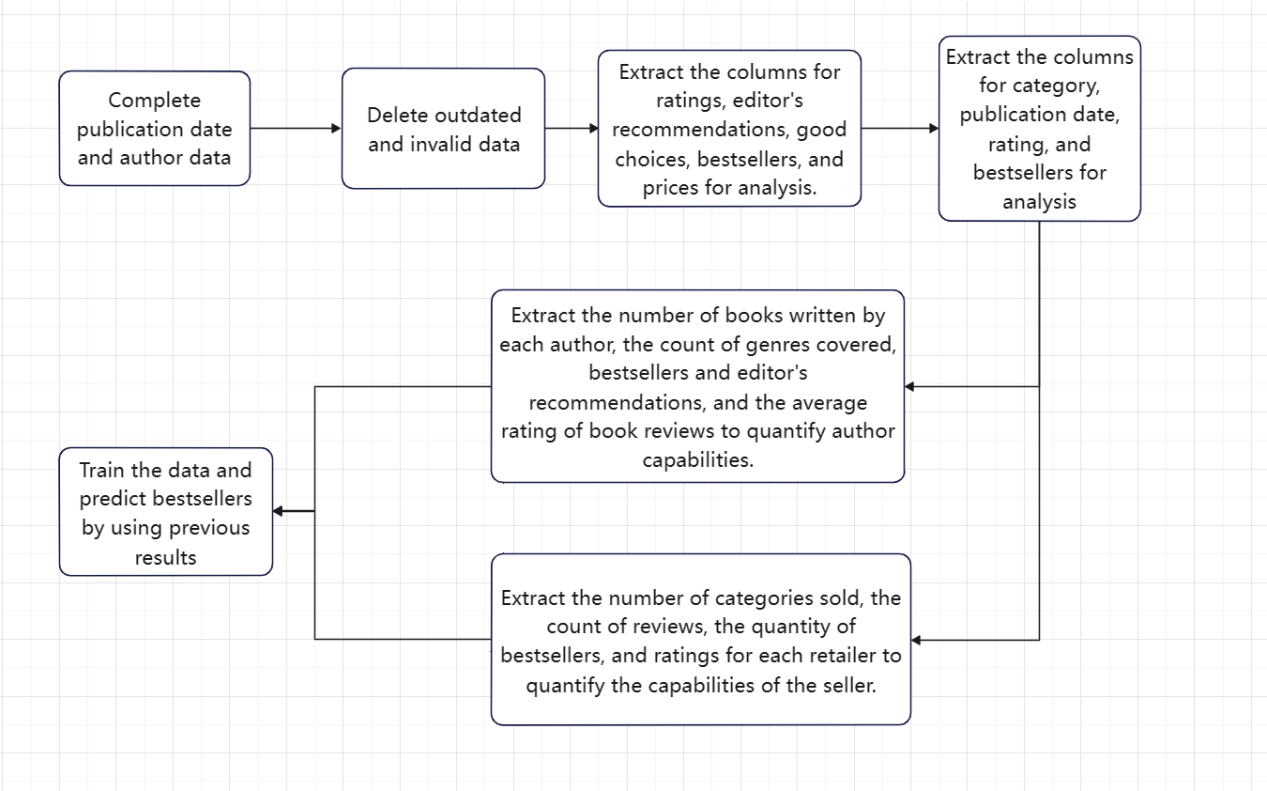
Specific Objective(s)

- **Objective 1:** *Explore whether books that excel in certain areas have commonalities in price and star ratings*
- **Objective 2:** *Trends in bestseller share, total releases and ratings over time, comparing books in different classifications*
- **Objective 3:** *Visualise and quantify seller competency, author competency respectively*
- **Objective 4:** *Helps authors predict whether their published book has the potential to become a bestseller*

System Design (5 marks)

Architecture

Architecture of our work is showed below:



data process flow_chart

Processing Modules and Algorithms

The most significant computational components and algorithms in our project include:

- Utilize web scraping to fill the missing data in the dataset, and remove the outdated data rows.
- Visualize the dataset by "pyecharts" and provide interactive ipywidgets.
- Apply the Analytic Hierarchy Process(AHP) and the Entropy Weight Method(EWM) to calculate the weight of the capabilities of sellers and authors respectively.
- Construct a special data structure, a decision tree. With the information provided about the seller, author, publication date, price, book category, etc., decision trees can be used to predict the probability of a book becoming a bestseller.

Program Code (15 marks)

Pakages that we used for project

We import packages for our project

In [1]:
```
import pandas as pd
import numpy as np

import csv
import math
import copy
import random
import threading
import time

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait

import pyecharts.options as opts
from pyecharts.charts import Timeline, Bar, Pie, Grid, Line,Radar,Scatter,Tab
from pyecharts.commons.utils import JsCode
from pyecharts.globals import ThemeType

from IPython.display import HTML,clear_output

from ipywidgets import Checkbox, HBox, HTML,VBox
from IPython.display import display
from ipywidgets import interact
import ipywidgets as widgets

from datetime import datetime

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler
```

Crawling section

We previously mentioned the issue of missing values in the publication date. We developed a web scraping script to fill these data.

We successfully utilized the Chromedriver automation control library to simulate user actions and retrieve the missing publication dates. Details of the simulation process are as follows:

Launch browser → Load page → Scroll page → (Click button) → Scrape data

Preparing for crawling

The method **set_crawling_strategy()** initializes chromedriver:

- maximize window: In this way, we can contain more information in one window and don't need to scroll down.And to some extent, save the time.
- set loading strategy as eager: Because the required information can be obtained before the page has fully loaded. Doing so can reduce the scraping time.
- load chrome driver path. **NOTION : chromedriver.exe** [(click to download) (https://sites.google.com/chromium.org/driver/)](https://sites.google.com/chromium.org/driver/) **needs to be put in the same folder as this document and to be careful about the version (it should be compatible with your Google Chrome).** Otherwise you may unable to run the code smoothly. But this code can still work if you jump crawler part.

In [2]:
```
def set_crawling_strategy():
    option= webdriver.ChromeOptions()
    # maximize the window
    option.add_argument("--start-maximized")
    #start the code with part of page to accelerate the
    option.page_load_strategy='eager'
    cService = webdriver.ChromeService(executable_path='./chromedriver.exe')
    return option,cService
```

In the data description, we mentioned the substantial volume of data that needs to be processed (48,000 entries). Crawling through each entry individually takes a considerable amount of time. Therefore, we introduced a threading class, **myThread()**, which can crawl several pages simultaneously.

myThread's member variables and functions:

- **name**: name of thread
- **link_range**: website task for this thread
- **driver**: the assigned driver for this thread
- rewrite **run()** and if we call start() method, run() method will be called

In [3]:
```
class myThread(threading.Thread):
    def __init__(self,name,link_range,driver):
        threading.Thread.__init__(self)
        self.name = name
        self.link_range = link_range
        self.driver=driver
    def run(self):  # rewrite
        crawler(self.name, self.link_range,self.driver)
```

The method **crawler(threadName,link_range,driver)** gets members variable of a certain thread as input, and will quit driver when it finishes its task without return value.The task mainly contains the following:

crawl reviews → crawl publication dates → all data finish quit driver

The detailed flow chart of **crawler(threadName,link_range,driver)** is listed below:

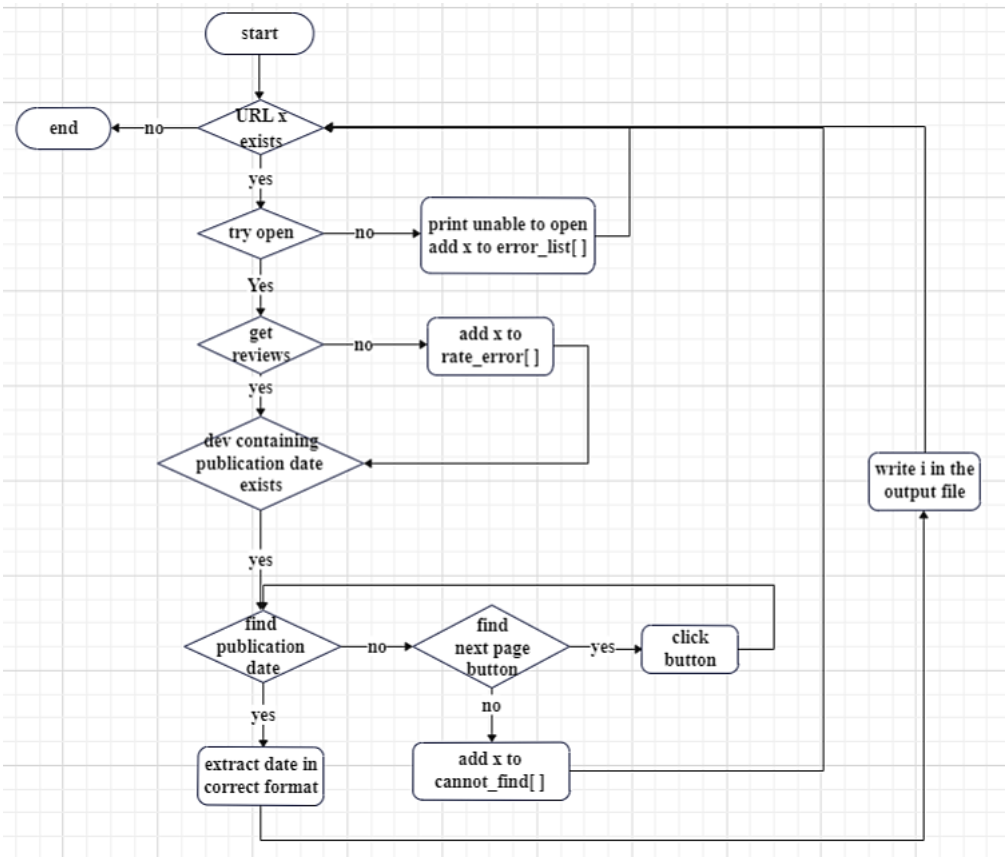


Figure1 crawler method flow chart

We give 10 times for the code to read the publication date and if this fails then we assume that we need to click the button to go to next page to find publication date. If there is no click button any more we assume that the page doesn't contain publication date.

In [4]:
```
def crawler(threadName,link_range,driver): #crawl reviews and publication dates
    for i in link_range:
        try:
            driver.get(i[5]) # the driver open the website by the url store in i[5]
            driver.execute_script("window.scrollBy(0,200)") #scroll down 200px to reach publication date dev
            wait1 = WebDriverWait(driver, 4, 0.5)#wait strategy1 wait 4s, and reflash 0.5s per time
            #Extract comment count.
            #If unable to find the number, record in 'rate_error'
            get_reviews(i,rate_error,wait1,driver)

            #wait until dev which contains publication date appear
            wait1.until(lambda x: x.find_element(By.ID,'rich_product_information'))
            B=driver.find_element(By.ID,'rich_product_information')

        except Exception as e:
            myerrors='unable to open: '+ i[5]+" book name is : "+i[1]+ " error;"
            print(myerrors)
            error_list.append(i)
            continue
        time=0
        time2=0
        flag=0
        time3=0
        while time3<=20:
            time3+=1
            try:
                A=B.find_element(By.ID,"rpi-attribute-book_details-publication_date")
            except Exception as e:  #if fail try 10 times, all fail assume not exists in this page
                time+=1
                if(time<=10):
                    continue
            time=0 #because we will go to next page, to give another 10 times try
            try:
                if A.text !="": #find publication date
                    break
                #try find next page button
                C=B.find_element(By.CSS_SELECTOR,"[class='a-button a-button-image a-carousel-button a-carousel-goto-nextpage
                C.click()
            except Exception as e:#try 10 times, all fail then publication date not exists in all pages
                time2+=1
                if(time2<=10):
                    continue
                myerrors='publication date not exists: '+ i[5]+" book name is : "+i[1]+ " error;"
                print(myerrors)
                cannot_find.append(i)
                flag=1
                break
        if flag==1:
            continue
        #extract and change format into day/month/year
        A=A.find_element(By.CSS_SELECTOR,"[class='a-section a-spacing-none a-text-center rpi-attribute-value']")
        a=A.text.split(" ")
        if len(a)==3: #if wrong length, a is definetly in wrong format!
            date=a[0][:-1]+"/"+a[1][:-1]+'/'+a[2][:-1]
        else:
            date_error.append(i)
            continue
        i[14]=date

        print("title: "+i[1]+"\nfind_date: "+i[14]+"\nURL: " +i[5]+"\n")
```

The method **get_reviews(i,rate_error,wait1)** gets input:

- **i**: the information of current proccessing data
- **rate_error**: error list to place wrong data
- **wait1**: wait strategy

get_reviews() will try to find the reviews that present in the website. If the review data is unable to find ,the code will put the information into rate_error.

In [5]:
```
def get_reviews(i,rate_error,wait1,driver):
    try:
        wait1.until(lambda y:y.find_element(By.ID,'acrCustomerReviewText'))
        Rate=driver.find_element(By.ID,'acrCustomerReviewText')
        rate=Rate.text.split(" ")
        rate=rate[0].split(",")
        i[7]="".join(rate)
    except Exception as e:
        myerrors='unable to find rate: '+ i[5]+" book name is : "+i[1]+ " error;"
        print(myerrors)
        rate_error.append(i)
```

Main body of crawling code

The main body of the code for crawling the publication date and reviews. In order to quickly show the result, we change the setting value of the variable **link_range_list**.

In [6]:
```
option,cService=set_crawling_strategy()

driver_list=[]
#build 5 drivers and put it into list then it will be assigned to each thread representively
for i in range(1,6):
    driver_list.append(webdriver.Chrome(service = cService,options=option))

link_range_list=[0,1,2,3,4,5] #just an example to show the function not what it actually did

#link_range_list=[0,2400,4800,7200,9600,12000]     #actually we set 5 thread to crawl the website.
                                                    #For the data from 0 to 11999 we seperated it to 5 groups
thread_list=[]
#lists to put different kind of error information
error_list=[]
date_error=[]
cannot_find=[]
rate_error=[]

reader=pd.read_csv("https://raw.githubusercontent.com/qiaoyu0906/mygit/master/kindle_data-v2.csv")

with open("sample.csv",'w',encoding='utf-8',newline="") as f2:
    writer=csv.writer(f2)
    listmyURL=[]
    count=0 #count how many books already have publication date
    for colname,i in reader.iterrows():
        if i[14] !="":
            count+=1
            continue
        listmyURL.append(i)

    print("number of book with date: "+str(count))
    #creat 5 thread
    for i in range(1,6): #Thread1-5
        thread=myThread('Thread-'+str(i),listmyURL[link_range_list[i-1]:link_range_list[i]],driver_list[i-1]) #each process
        thread.daemon=True
        thread.start() #call run() method
        thread_list.append(thread)

    for t in thread_list: #wait all thread finished
        t.join()
    for i in driver_list:
        i.close()
```

```
number of book with date: 133102
```

The result above shows that our code can successfully crawl down publication date and reviews for the books which lack of them before. Actually, we write output into files, and gather together to generate the complete data. We update the complete data **complete_kindle_data.csv** on github.[click to see] (https://raw.githubusercontent.com/qiaoyu0906/mygit/master/complete_kindle_data.csv)

We also find that there exists missing value in author column, so we write quite similar code for this problem as well and put **complete_kindle_data_add_author.csv** on github.[click to see] (https://raw.githubusercontent.com/qiaoyu0906/mygit/master/complete_kindle_data_add_author.csv), which we are going to use next.

In [7]:
```
#Update dataframe
BOOK_DF=pd.read_csv("https://raw.githubusercontent.com/qiaoyu0906/mygit/master/complete_kindle_data_add_author.csv",encoding=
```

Objective 1

In order to research the distribution of books in different areas("isBestSeller","isEditorsPick","isGoodReadsChoice"):

- We using its Boolean value to correlate these three variables into eight categories。
- Building a dictionary and returning a dataset that is not empty.

In [8]:
```
BOOKdf=BOOK_DF.copy()
TTT = BOOKdf[(BOOKdf["isBestSeller"] == True) & (BOOKdf["isEditorsPick"] == True) & (BOOKdf["isGoodReadsChoice"] == True)]
TTF = BOOKdf[(BOOKdf["isBestSeller"] == True) & (BOOKdf["isEditorsPick"] == True) & (BOOKdf["isGoodReadsChoice"] == False)]
TFT = BOOKdf[(BOOKdf["isBestSeller"] == True) & (BOOKdf["isEditorsPick"] == False) & (BOOKdf["isGoodReadsChoice"] == True)]
FTT = BOOKdf[(BOOKdf["isBestSeller"] == False) & (BOOKdf["isEditorsPick"] == True) & (BOOKdf["isGoodReadsChoice"] == True)]
TFF = BOOKdf[(BOOKdf["isBestSeller"] == True) & (BOOKdf["isEditorsPick"] == False) & (BOOKdf["isGoodReadsChoice"] == False)]
FTF = BOOKdf[(BOOKdf["isBestSeller"] == False) & (BOOKdf["isEditorsPick"] == True) & (BOOKdf["isGoodReadsChoice"] == False)]
FFT = BOOKdf[(BOOKdf["isBestSeller"] == False) & (BOOKdf["isEditorsPick"] == False) & (BOOKdf["isGoodReadsChoice"] == True)]
FFF = BOOKdf[(BOOKdf["isBestSeller"] == False) & (BOOKdf["isEditorsPick"] == False) & (BOOKdf["isGoodReadsChoice"] == False)]
#use filtering, to exclude nulls
dict_of_dataframes = {'TTT': TTT, 'TTF': TTF, 'TFT': TFT, 'FTT': FTT,
                      'TFF': TFF, 'FTF': FTF, 'FFT': FFT, 'FFF': FFF}
non_empty_dataframes = {key: df for key, df in dict_of_dataframes.items() if not df.empty}
for key, df in non_empty_dataframes.items():
    print(f"{key} is not empty")
```

```
TFF is not empty
FTF is not empty
FFT is not empty
FFF is not empty
```

- Turning it into a list makes it easier for us to draw graphs with this data
- Additionally, we collect all category lengths that will help us analysis their relationship.

In [9]:
```
#to get every length of the category.
TFF_price = TFF["price"].tolist()
FTF_price = FTF["price"].tolist()
FFT_price = FFT["price"].tolist()
FFF_price = FFF["price"].tolist()
book_counts = [len(FFF_price), len(FTF_price), len(FFT_price), len(TFF_price)]
categories = ['FFF','FTF','FFT', 'TFF']
for category, count in zip(categories, book_counts):
    print(f"{category} has {count} books")
```

```
FFF has 122483 books
FTF has 5577 books
FFT has 1319 books
TFF has 2196 books
```

The outcome shows that there are four categories which is not empty, and presents each length of the categories. We can use this information to look for relationships between variables and stars by using visualization.

The method **Stars_and_variable_relationship()**

- We plan to create three diagrams: one for Kindle Unlimited, another for the four screening categories, and a third one showing all the factors taken into account to help us understand their relationship with the ratings. We will use the 'tab' option to switch between these graphs.
- Due to the large amount of data, we have set up pullable x and y axes to make it easier to examine all the data or focus on a particular part of the ratings and price.
- And using JsCode helps us to know every point of information accurately. (We have also generated web links to help facilitate easy viewing)

In [10]:
```
tars_and_variable_relationship():

_band = FFF_price + TFF_price + FTF_price + FFT_price
=BOOKdf[BOOKdf['isKindleUnlimited'] == False]['price'].tolist()
=BOOKdf[BOOKdf['isKindleUnlimited'] == True]['price'].tolist()
=a+b

sKindleUnilinited = (
    Scatter()
    .add_xaxis(X)
    .add_yaxis("N_KU", BOOKdf[BOOKdf['isKindleUnlimited'] == False]['stars'].tolist(), symbol='diamond',symbol_size=8, color='
    .add_yaxis("KU", BOOKdf[BOOKdf['isKindleUnlimited'] == True]['stars'].tolist(), symbol='diamond', symbol_size=8, color='re
    .set_global_opts(
        title_opts=opts.TitleOpts(title="Stars and Prices"),
        xaxis_opts=opts.AxisOpts(type_="value", name="price"),
        yaxis_opts=opts.AxisOpts(type_="value", name="starts"),
        datazoom_opts=[opts.DataZoomOpts(filter_mode="none", orient="horizontal"), opts.DataZoomOpts(filter_mode="none", orien
        tooltip_opts=opts.TooltipOpts(axis_pointer_type='cross',
            formatter=JsCode(
                "function(params){return 'price: ' + params.value[0] + ', stars: ' + params.value[1];}"
            ))
    )

andEandR = (
    Scatter()
    .add_xaxis(X_band)
    .add_yaxis("NS_NE_NG", FFF["stars"].tolist(), symbol='circle', symbol_size=8, color='purple',itemstyle_opts=opts.ItemStyle
    .add_yaxis("S_NE_NG", TFF["stars"].tolist(), symbol='circle',  symbol_size=8, color='orange',itemstyle_opts=opts.ItemStyle
    .add_yaxis("NS_E_NG", FTF["stars"].tolist(), symbol='circle',  symbol_size=8, color='cyan',itemstyle_opts=opts.ItemStyleOp
    .add_yaxis("NS_NE_G", FFT["stars"].tolist(), symbol='circle', color='pink', symbol_size=8, itemstyle_opts=opts.ItemStyleOp
    .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
    .set_global_opts(
        title_opts=opts.TitleOpts(title="Stars and Prices"),
        xaxis_opts=opts.AxisOpts(type_="value", name="price"),
        yaxis_opts=opts.AxisOpts(type_="value", name="stars"),
        #Set the horizontal and vertical coordinates that you can slide
        datazoom_opts=[opts.DataZoomOpts(filter_mode="none", orient="horizontal"), opts.DataZoomOpts(filter_mode="none", orien
        #set axis pointer
        tooltip_opts=opts.TooltipOpts(axis_pointer_type='cross',
            formatter=JsCode(
                "function(params){return 'price: ' + params.value[0] + ', stars: ' + params.value[1];}"
            ))
    )

 conjunct two charts
ewisKindleUnilinited=copy.deepcopy(isKindleUnilinited)
verlap = newisKindleUnilinited.overlap(BandEandR)
ab = Tab()
ab.add(isKindleUnilinited, "Figure 1 ")
ab.add(BandEandR, "Figure 2")
ab.add(overlap, "Figure 3")
ab.render('object1.html')
ab.render_notebook()
eturn tab.render('objective1.html'),tab.render_notebook()
```

A scatterplot will be presented, which is specifically presented in the subsequent visualisation section

Objective 2

In order to find out differences between different categories in different year, we need to first extract year through **publishedDate**:

In [11]:
```
#add column "publishedYear"
a=BOOK_DF.copy()
a.insert(value=BOOK_DF["publishedDate"].str[:4],column="publishedYear",loc=16)
display(a.head(2))
```

	asin	title	author	soldBy	imgUrl	productURL	stars	reviews	price	isKindl
0	B00BNC8UPQ	"Termite" Terry's Termite License Exam Prepara...	"Termite" Terry Singleton	Amazon.com Services LLC	https://m.media-amazon.com/images/I/61OjK7PDdq...	https://www.amazon.com/dp/B00BNC8UPQ	4.6	89	9.99	
1	B00DLV0KN2	"Termite" Terry's Pest Control License Exam Pr...	"Termite" Terry Singleton	Amazon.com Services LLC	https://m.media-amazon.com/images/I/61KdtFa9d8...	https://www.amazon.com/dp/B00DLV0KN2	4.7	0	9.99	

Through the result above, we successfully extract year from **publishedDate** and insert it to **BOOK_DF** as column **publishedYear**.

And we then calculate the average stars rate for a certain category in a certain year.

In [12]:
```
#Calculate the average star rating of the combination of "category_name" and "publishedYear"
bardata2=a.groupby(["category_name","publishedYear"])["stars"].mean()
category=a["category_name"].unique().tolist()
drop2024=a[a["publishedYear"]<="2023"]
data_year=sorted(drop2024["publishedYear"].unique().tolist())
data_name=list(bardata2.index)
data_value=list(bardata2.values)

templist=[]
for i,j in zip(data_name,data_value):
    templist.append((i,j))
print(templist[:5])
```

```
[(('Arts & Photo graphy', '1958'), 4.4), (('Arts & Photo graphy', '1964'), 4.35), (('Arts & Photo graphy', '1970'), 4.7),
(('Arts & Photo graphy', '1972'), 4.6), (('Arts & Photo graphy', '1974'), 4.5)]
```

The result above is a tuple list, each item gets the format: ((some category **x** , year **y**), average star **z**). This tuple describes that "in year **y**, category **x** got average star **z**".
In order to make the data calculated above organise better, we define a class **mycategory** to store information. The class gets following member variables:

- **name**: the name of this category
- **total**: the total number of books this category has in a certain year
- **year**: the certain year
- **best**: the percentage of books belong to Bestsellers * average_score
- **notbest**: the percentage of books not belong to Bestsellers * average_score **[notion]**: average_score change from 5 to 100

In [13]:
```
class mycategory:
    def __init__(self,name,best,notbest,total,year):
        self.name=str(name)
        self.best=float(best)
        self.total=int(total)
        self.year=year
        self.notbest=float(notbest)
```

After getting a well organised data structure, we prepare detailed data for visualization.

In [14]:
```
allbooks={}
for i in data_year:
    booklist=[]
    category_y=a[a["publishedYear"]==i] #find all books in year i
    for j in category:
        flag=0
        category_sum=category_y[category_y["category_name"]==j] # find all j category books in year i
        Best_seller=category_sum["isBestSeller"].sum() # get how many bestseller are there for j category books in year i
        for k in templist:
            if k[0][0]==j and k[0][1]== i:
                BEST_percent=float( Best_seller / len(category_sum) )
                book=mycategory(j,float(np.power(2.51188643150958,k[1]))*BEST_percent,float(np.power(2.51188643150958,k[1]))*
                booklist.append(book)
                flag=1
                break
        if flag==0:
            book=mycategory(j,0,0,0,i)
            booklist.append(book)
    allbooks[i]=booklist
```

The method **get_year_chart(year)** get input **year**, and create the stack plot that year.
The stack plot contains the average score, the total number of books published, as well as divides the average score to two part to show the percentage of bestseller for each category in some year.

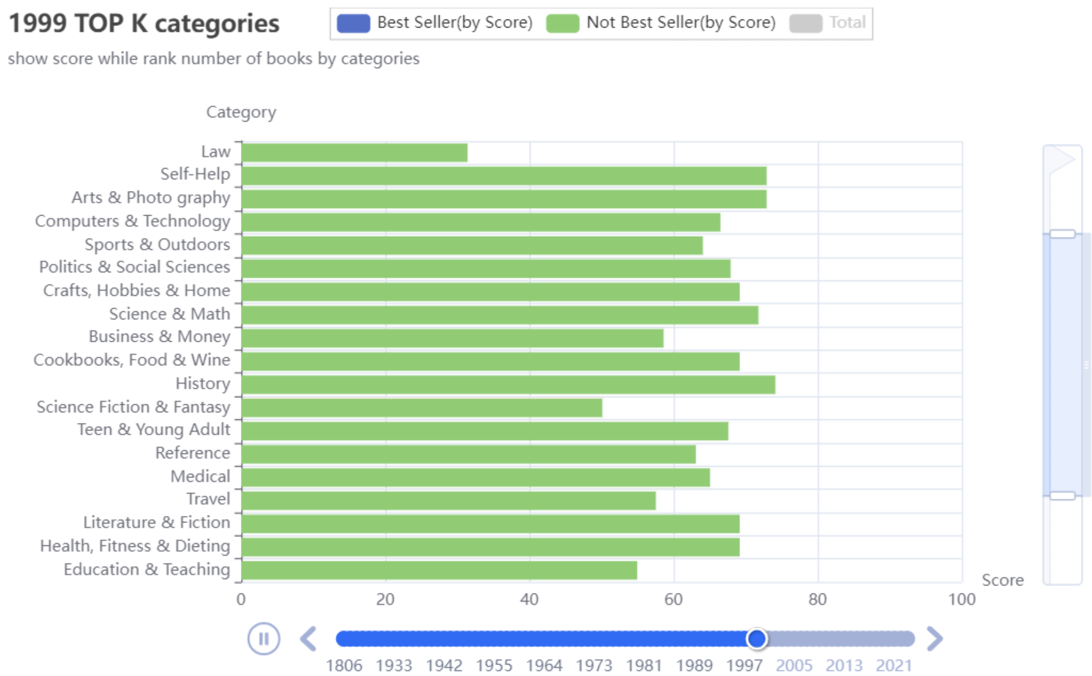
In [15]:
```
def get_year_chart(year):
    name_list=[]
    best_list=[]
    notbest_list=[]
    total_list=[]
    allbooks[year].sort(key=lambda x: x.total)
    for i in allbooks[year]:
        name_list.append(i.name)
        best_list.append(i.best)
        notbest_list.append(i.notbest)
        total_list.append(i.total)
    bar = (
        Bar()
        .add_xaxis(
            xaxis_data=name_list,
        )
        .add_yaxis(
            series_name="Best Seller(by Score)",
            y_axis=best_list,
            label_opts=opts.LabelOpts(is_show=False),
            stack="abc",
        )
        .add_yaxis(
            series_name="Not Best Seller(by Score)",
            y_axis=notbest_list,
            label_opts=opts.LabelOpts(is_show=False),
            stack="abc",
        )
        .add_yaxis(
            series_name="Total",
            y_axis=total_list,
        )
        .reversal_axis()
        .set_global_opts(
            datazoom_opts=opts.DataZoomOpts(
                filter_mode="none",
                orient="vertical",
                pos_right="10%"
            ),
            xaxis_opts=opts.AxisOpts(
                name="Score",
                max_=100
            ),
            yaxis_opts=opts.AxisOpts(
                axislabel_opts=opts.LabelOpts(interval=0),
                name="Category",
            ),
            title_opts=opts.TitleOpts(
                title="{} TOP K categories".format(year),
                subtitle="show score while rank number of books by categories"
            ),
            tooltip_opts=opts.TooltipOpts(
                is_show=True, trigger="axis", axis_pointer_type="shadow"
            ),
            legend_opts=opts.LegendOpts(
                selected_map={
                    "Score": True,
                    "Total":False
                }
            ),
        )
    )
    return bar
```

We build **draw_top_k_categories()** method to visualize how scores change between different categories. We use timeline() method provided by pyecharts to draw animated graph. We loop through years and display the stack plot by calling **get_year_chart(year)**.

In [16]:
```
def draw_top_k_categories():
    timeline = Timeline()
    for y in data_year:
        if y=="2024":
            break
        grid = (Grid()
            .add(get_year_chart(year=y),grid_opts=opts.GridOpts(pos_left="20%",pos_top="20%",pos_right="20%"))
            )
        timeline.add(grid, time_point=str(y))

    timeline.add_schema(is_auto_play=True, play_interval=1500)
    return timeline
draw_top_k_categories().render_notebook()
```

Out[16]:



In order to catch the up-and-down trend of scores for a particular category, we draw a line chart to represent this information. Likewise, we try to figure out each category's score list through the years. And gather them together in the dictionary: **allcategory**

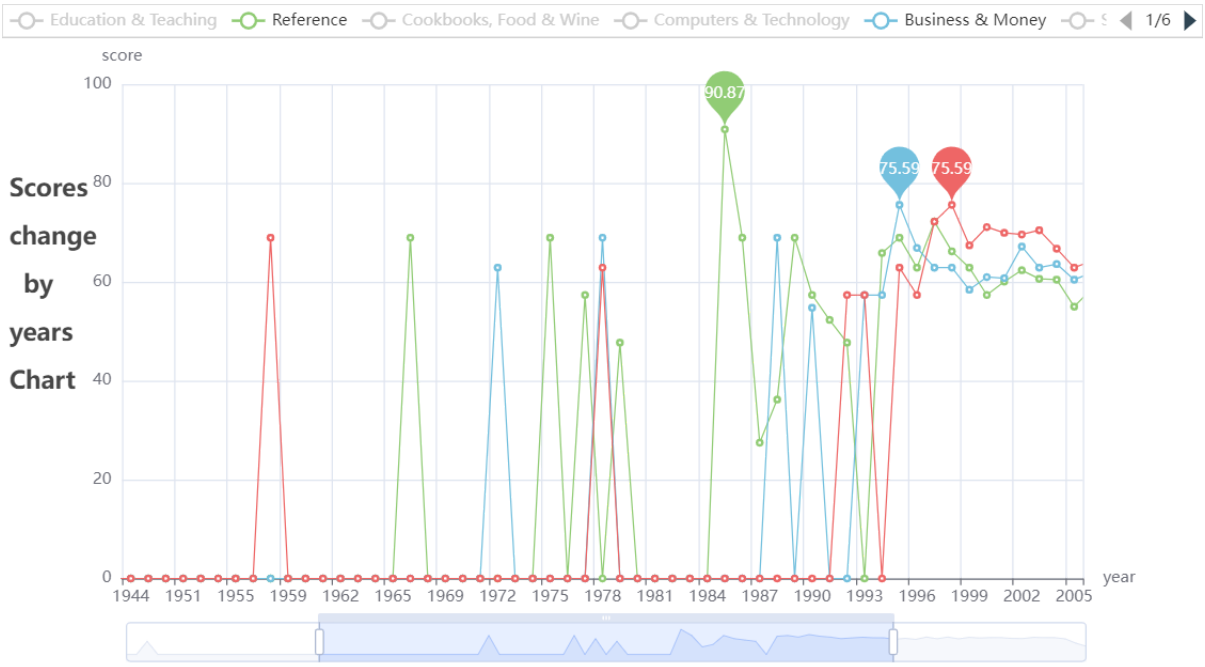
In [17]:
```
allcategory={}
setlegend={}
k=0
for i in category:
    k=k+1
    category_y=a[a["category_name"]==i]
    scorelist=[]
    if k==2 or k==5 or k ==13:
        setlegend[i]=True
    else:
        setlegend[i]=False
    for j in data_year:
        category_avg=category_y[category_y["publishedYear"]==str(j)]
        if pd.notna(category_avg["stars"].mean()) == False:
            scorelist.append(0)
        else:
            scorelist.append(format(float(np.power(2.51,category_avg["stars"].mean())), '.2f'))
    allcategory[i]=scorelist
print(i)
print(allcategory[i])
```

```
Nonfiction
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '68.94', 0, 0, 0, '75.59', 0, '60.61', '64.84', '58.69', '6
6.86', '64.05', '65.84', '65.84', '67.80', '62.88', '62.58', '68.00', '68.16', '67.50', '70.09', '68.02', '68.37', '68.30',
'64.60', '68.34', '69.34', '67.99', '66.27', '62.10']
```

The result above shows that we sucessfully extract the average score of Nonfiction books published in each year. We then use these data to draw the line chart which shows how scores change for years for a single category. We build **draw_line_score_chart()** method to visualize the change.

In [18]:
```
def draw_line_score_chart():
    line = Line()
    line.add_xaxis(xaxis_data=data_year)
    for i in category:
        line.add_yaxis(
            series_name=i,
            y_axis=allcategory[i],
            label_opts=opts.LabelOpts(is_show=False),
            markpoint_opts=opts.MarkPointOpts(data=[opts.MarkPointItem(type_="max")]),
        )
    line.set_global_opts(
        title_opts=opts.TitleOpts(title=" Scores\n\n change\n\n   by\n\n years\n\n Chart",pos_left=-5,pos_top="25%"),
        xaxis_opts=opts.AxisOpts(type_="category",name="year"),
        yaxis_opts=opts.AxisOpts(type_="value",name="score"),
        tooltip_opts=opts.TooltipOpts(trigger="axis"),
        legend_opts=opts.LegendOpts(
            type_="scroll",
            pos_top=0,
            pos_right=0,
            selected_map=setlegend
        ),
        datazoom_opts=opts.DataZoomOpts(
            filter_mode="none",
            pos_right="10%"
        ),
    )
    return line
draw_line_score_chart().render_notebook()
```

Out[18]:

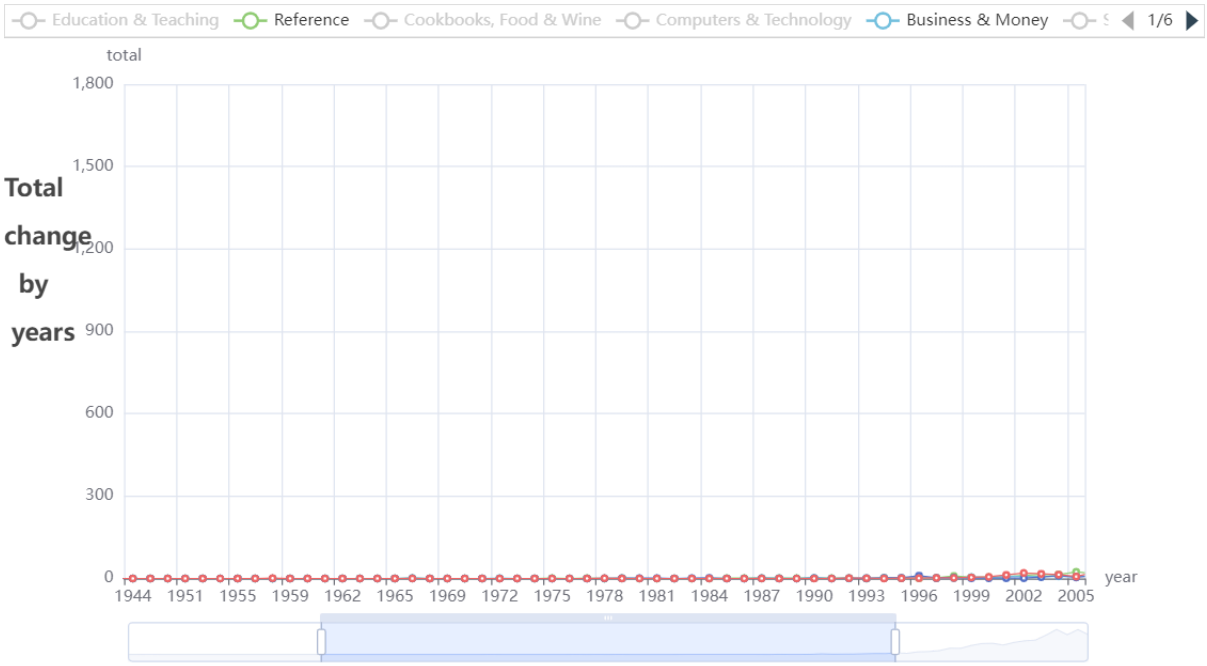


We do the same way. But this time, we try to draw the number of books each category published each year.

In [19]:
```
k=0
allcategory2={}
setlegend2={}
for i in category:
    k=k+1
    category_y=a[a["category_name"]==i]
    totallist=[]
    if k==2 or k==5 or k==10 or k==13:
        setlegend[i]=True
    else:
        setlegend[i]=False
    for j in data_year:
        category_sum=category_y[category_y["publishedYear"]==str(j)]
        if pd.notna(category_sum["stars"].count()) == False:
            totallist.append(0)
        else:
            totallist.append(int(category_sum["stars"].count()))
    allcategory2[i]=totallist

def draw_line_total_chart():
    line2 = Line()
    line2.add_xaxis(xaxis_data=data_year)
    for i in category:
        line2.add_yaxis(
            series_name=i,
            y_axis=list(allcategory2[i]),
            label_opts=opts.LabelOpts(is_show=False),
            markpoint_opts=opts.MarkPointOpts(data=[opts.MarkPointItem(type_="max")]),
        )
    line2.set_global_opts(
        title_opts=opts.TitleOpts(title="Total\n\nchange\n\n  by\n\n years",pos_left=-5,pos_top="25%"),
        xaxis_opts=opts.AxisOpts(type_="category",name="year"),
        yaxis_opts=opts.AxisOpts(type_="value",name="total"),
        tooltip_opts=opts.TooltipOpts(trigger="axis"),
        legend_opts=opts.LegendOpts(
            type_="scroll",
            pos_top=0,
            pos_right=0,
            selected_map=setlegend
        ),
        datazoom_opts=opts.DataZoomOpts(
            filter_mode="none",
            pos_right="10%"
        ),
    )
    return line2
draw_line_total_chart().render_notebook()
```

Out[19]:



Objective 3

With this objective 3, we have two tasks that go into researching the situation of the seller(objective 3a) and the situation related to the author(objective 3b).

objective 3a code

We wil use filterable radar map to show their individual information to evaluate the seller's potential for sales, and comprehend the relationship between the seller and the book kinds, whether or not it is a best-seller, and between reviews and ratings.

- We need to get information about each 'soldBy'. We can initially notice that some vendors are using different identities, but they actually share the same head office.In order to avoid to meet minimums, we all refer to the head office when discussing the remaining details.
- The mean quantity of reviews and star ratings for a book are utilized to indicate its level of performance.
- Calculating the truth of percentage(The number of best sellers is a percentage of the number of all books in that seller) to indicate the seller's advantage in terms of 'isBestSeller' and find all kinds of books sold by the seller.
- We create a new dateframe ‘outer’to show the radar charts easier.

In [20]:
```
name_mapping = {
    "Harper Collins": "Harper Collins",
    "HarperCollins Publishers": "Harper Collins",
    "HarperCollins Publishing": "Harper Collins",
    "Simon & Schuster Digital Sales Inc.":'Simon and Schuster Digital Sales Inc',
     'Amazon Digital Services LLC GU':"Amazon",
    "Amazon Digital Services LLC HN":"Amazon",
    "Amazon Digital Services LLC MK":"Amazon",
    "Amazon.com":"Amazon",
    "Amazon.com Services LLC":"Amazon",
    "Harlequin Digital Sales Corp":"Hachette Book Group",
    "Macmillan Higher Education":"Macmillan",
    "Penguin Group (USA) LLC":"Penguin Random House",
    "Penguin Random House Australia Pty Ltd":"Penguin Random House",
    "Penguin Random House Grupo Editorial":"Penguin Random House",
    "Penguin Random House Publisher Services":"Penguin Random House",
    "Random House India":"Random House",
    "Random House LLC":"Random House",
    "Random House NZ":"Random House",
    "Random House ZA":"Random House",
}
BOOKdf["soldBy"] = BOOKdf["soldBy"].replace(name_mapping)
B = BOOKdf.groupby("soldBy")["isBestSeller"].mean().reset_index(name="percentage_of_bestseller")
C = BOOKdf.groupby(["soldBy"])["category_name"].nunique().reset_index(name="category_count")
R = BOOKdf.groupby(["soldBy"])["reviews"].mean().reset_index(name="average_reviews")
S = BOOKdf.groupby(["soldBy"])["stars"].mean().reset_index(name="average_stars")
outer = pd.merge(C, B, on="soldBy", how="outer")
outer = pd.merge(outer, R, on="soldBy", how="outer")
outer = pd.merge(outer, S, on="soldBy", how="outer")
outer["category_count"] = outer["category_count"].astype(float)
outer1=outer.copy()
outer
```

Out[20]:

	soldBy	category_count	percentage_of_bestseller	average_reviews	average_stars
0	Amazon	30.0	0.018663	911.541669	4.361772
1	Book Republic	1.0	0.000000	135.000000	4.100000
2	Cengage Learning	18.0	0.004005	127.718291	4.473298
3	DC Comics	1.0	0.000000	25.000000	4.366667
4	De Marque	28.0	0.164948	3533.804124	4.371649
5	Disney Book Group	24.0	0.037879	2106.037879	4.575758
6	EDIGITA	8.0	0.000000	1012.833333	3.883333
7	Editorial Planeta, S.A.U.	16.0	0.001669	917.338898	4.270451
8	Flammarion Lt.	3.0	0.000000	1093.964286	4.250000
9	Gallimard Lt.	4.0	0.027027	1739.729730	4.308108
10	Games Workshop	3.0	0.000000	979.202532	4.611392
11	GeMS SpA	4.0	0.000000	1050.615385	3.923077
12	Giunti Editore S.p.A.	2.0	0.000000	423.000000	4.233333
13	Hachette Book Group	29.0	0.012270	2514.114026	4.480749
14	Harlequin Digital Sales Corp.	21.0	0.000000	1772.032037	4.436613
15	Harper Collins	30.0	0.018015	2346.598090	4.518267
16	INTERFORUM	7.0	0.000000	2301.083333	4.066667
17	Immatériel fr	2.0	0.000000	3820.833333	4.008333
18	JOHN WILEY AND SONS INC	23.0	0.001679	131.554343	4.360134
19	Macmillan	29.0	0.011631	2749.192067	4.433373
20	Marvel Entertainment US	3.0	0.044444	835.633333	4.565556
21	Newton Compton	1.0	0.000000	1202.500000	4.450000
22	PRH UK	25.0	0.000000	431.837989	4.455587
23	Pearson Education,Inc.	19.0	0.000000	80.716031	4.346947
24	Penguin Random House	30.0	0.010498	2317.739114	4.479782
25	Pottermore	5.0	0.153846	20976.487179	4.689744
26	RCS MediaGroup S.p.A.	1.0	0.000000	508.000000	4.300000
27	RH AU	7.0	0.000000	324.636364	4.318182
28	Random House	29.0	0.008596	2900.635260	4.488421
29	Scholastic Trade Publisher	7.0	0.000000	7608.950495	4.645545
30	Simon and Schuster Digital Sales Inc	29.0	0.016298	2073.309444	4.494985
31	Versilio	1.0	0.000000	3796.500000	4.200000
32	Volumen	2.0	0.000000	880.000000	2.833333
33	W. W. Norton & Company	13.0	0.008333	48.650000	4.126667
34	Yen Press LLC	7.0	0.000000	1251.419753	4.769136
35	hhh	1.0	0.000000	0.000000	4.200000

We find there are 35 sellers collected, and there are some extreme values we need to consider, which will help us set the radar's dimensions. Example: Average reviews of each book have one more than 20000 value.

We use MinMax normalisation to place the data between 1 and 10 to help us to quantified data

In [21]:
```
MinMaxdf = outer.iloc[:, 1:]
scaler = MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(MinMaxdf), columns=MinMaxdf.columns)
df_quantized = 1 + df_normalized * 9
outer = pd.concat([outer['soldBy'], df_quantized], axis=1)
```

We intend to use the Analytic Hierarchy Process（AHP） to quantify the competence of our vendors, and to score each of them. So we consider the type of sales and the percentage of bestsellers to be relatively important for sellers, and construct a judgment matrix.

In [22]:
```
def ahp(matrix):
    normalized_matrix = matrix / matrix.sum(axis=0)
    weights = normalized_matrix.mean(axis=1)
    normalized_weights = weights / weights.sum()
    return normalized_weights

pairwise_matrix = np.array([
     [1, 1/1.5, 2, 1/2],
     [1.5, 1, 2, 1.5],
     [1/2, 1/2, 1, 1/2],
     [2, 1/1.5, 2, 1]
])

factors = outer.columns[1:]
weights = ahp(pairwise_matrix)

weights_dict = dict(zip(factors, weights))

print("weights: ", weights_dict)

outer['score'] = outer.apply(lambda row: sum(w * row[f] for w, f in zip(weights, factors)), axis=1)
outer.sort_values(by='score', ascending=False)
```

```
weights:  {'category_count': 0.21596638655462186, 'percentage_of_bestseller': 0.34180672268907564, 'average_reviews': 0.140
54621848739496, 'average_stars': 0.3016806722689076}
```

Out[22]:

	soldBy	category_count	percentage_of_bestseller	average_reviews	average_stars	score
4	De Marque	9.379310	10.000000	2.516185	8.151993	8.256622
25	Pottermore	2.241379	9.394231	10.000000	9.630887	8.005988
5	Disney Book Group	8.137931	3.066761	1.903599	9.100939	5.818880
15	Harper Collins	10.000000	1.982931	2.006812	8.833652	5.784435
30	Simon and Schuster Digital Sales Inc	9.689655	1.889234	1.889557	8.725410	5.636251
24	Penguin Random House	10.000000	1.572779	1.994430	8.654728	5.588524
13	Hachette Book Group	9.689655	1.669479	2.078685	8.659222	5.567750
28	Random House	9.689655	1.469019	2.244523	8.694890	5.533300
19	Macmillan	9.689655	1.634637	2.179546	8.438960	5.503568
0	Amazon	10.000000	2.018305	1.391099	8.106071	5.490493
22	PRH UK	8.448276	1.000000	1.185281	8.542236	4.909965
14	Harlequin Digital Sales Corp.	7.206897	1.000000	1.760294	8.454025	4.696073
18	JOHN WILEY AND SONS INC	7.827586	1.091586	1.056444	8.098456	4.655234
2	Cengage Learning	6.275862	1.218541	1.054798	8.624579	4.521998
20	Marvel Entertainment US	1.620690	3.425000	1.358530	9.053508	4.442907
29	Scholastic Trade Publisher	2.862069	1.000000	4.264634	9.425395	4.402755
23	Pearson Education,Inc.	6.586207	1.000000	1.034631	8.037143	4.334270
34	Yen Press LLC	2.862069	1.000000	1.536924	10.000000	4.192733
7	Editorial Planeta, S.A.U.	5.655172	1.091089	1.393586	7.681496	4.107491
9	Gallimard Lt.	1.931034	2.474662	1.746434	7.856574	3.878526
33	W. W. Norton & Company	4.724138	1.454687	1.020873	7.013010	3.776646
10	Games Workshop	1.620690	1.000000	1.420129	9.266614	3.686973
27	RH AU	2.862069	1.000000	1.139286	7.903409	3.504345
21	Newton Compton	1.000000	1.000000	1.515935	8.516263	3.340024
16	INTERFORUM	2.862069	1.000000	1.987284	6.734056	3.270757
8	Flammarion Lt.	1.620690	1.000000	1.469367	7.586416	3.187010
3	DC Comics	1.000000	1.000000	1.010726	8.128827	3.152137
31	Versilio	1.000000	1.000000	2.628895	7.353954	3.145800
26	RCS MediaGroup S.p.A.	1.000000	1.000000	1.217958	7.818878	3.087757
12	Giunti Editore S.p.A.	1.310345	1.000000	1.181489	7.508929	3.056150
6	EDIGITA	3.172414	1.000000	1.434558	5.881696	3.002957
17	Immatériel fr	1.310345	1.000000	2.639335	6.462851	2.945463
35	hhh	1.000000	1.000000	1.000000	7.353954	2.916865
11	GeMS SpA	1.931034	1.000000	1.450768	6.066474	2.792883
1	Book Republic	1.000000	1.000000	1.057922	6.889031	2.784747
32	Volumen	1.310345	1.000000	1.377566	1.000000	1.120089

We can see that each merchant has been quantified and a "score" has been added to the last column.

The method **show_the_seller_information()**. We create an interactive radar map that displays the various abilities of the sellers.

- We first define the basic information and set the radar's dimensions to ensure that all details can be shown.
- Create a set of check boxes that allow us to select any seller we decide upon, including all vendors.

In [23]:
```
def show_the_seller_information():
    # Set dimension
    book_schema=[
    {"name": "Category", "max": 30.0, "min": 1.0},
    {"name": "BestSeller", "max": 0.165, "min": 0.0},
    {"name": "Reviews", "max": 20977, "min": 25},
    {"name": "Stars", "max": 4.8, "min": 2.8},
    ]
    def select_all_change(c):
        for checkbox in checkboxes:
            checkbox.value = select_all_checkbox.value
    series_names = outer1["soldBy"].tolist()

    # Create a set of check boxes
    select_all_checkbox = widgets.Checkbox(description='Select All', value=False)
    select_all_checkbox.observe(select_all_change, names='value')
    checkboxes = [widgets.Checkbox(description=name, value=False) for name in series_names]
    checkboxes_container = widgets.VBox(children=[select_all_checkbox] + checkboxes , layout=widgets.Layout(align_items='cente
    checkboxes_container.layout.width = "1000px"
    checkboxes_container.layout.height = "200px"
    checkboxes_container.layout.align_items = 'center'
    for checkbox in checkboxes:
        checkbox.layout.width = '300px'


    # Create button
    submit_button = widgets.Button(description="Submit", button_style="primary")
    display(checkboxes_container, submit_button)

    radar_container = widgets.Output()
    display(radar_container)

    # Click the event handler to connect to the button
    submit_button.on_click(lambda b: button_click(b, outer1, checkboxes, radar_container,book_schema))
```

The method **button_click(b, outer1, checkboxes, radar_container,book_schema)**.

- Create a button that enables us to input our selected sellers and generate a map upon submission. We've also generated web links to facilitate easy viewing.

In [24]:
```
def button_click(b, outer1, checkboxes, radar_container,book_schema):
    color_dict = {}
    for i in range(len(outer1)):
        color_dict[i] = "#{:02x}{:02x}{:02x}".format(
            random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)
        )

    # Create a radar
    radar = (
        Radar()
        .add_schema(
            schema=book_schema,
            shape="circle",
            splitarea_opt=opts.SplitAreaOpts(is_show=False),
        )
        .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
        .set_global_opts(
            title_opts=opts.TitleOpts(title="soldBy"),
            legend_opts=opts.LegendOpts(type_="scroll", pos_left="right", orient="vertical", item_width=4, item_gap=5),
        )
    )
    selected_series = [

    ]
    for checkbox in checkboxes:
        if checkbox.value:
            selected_series.append(checkbox.description)

    if selected_series:
        outer2 = outer1[outer1["soldBy"].isin(selected_series)]

        # color
        for i in range(len(outer2)):
            radar.add(
                series_name=outer2.iloc[i, 0],
                data=[outer2.iloc[i, 1:].tolist()],
                areastyle_opts=opts.AreaStyleOpts(opacity=0.5),
                label_opts=opts.LabelOpts(is_show=False),
                color=color_dict[i],
            )

        with radar_container:
            clear_output(wait=True)
            radar_notebook = radar.render_notebook()
            display(radar_notebook)
            radar.render('objective3a.html')
```

A radar map will be presented, which is specifically presented in the subsequent visualisation section.

objective 3b code

We would like to research the author's ability by the number of books he or she has written, the types of books covered, whether the books are editor-recommended, whether his or her new books are likely to be bestsellers, and what star ratings are like.

The method **calculate_ability_of_author()**

- Define a function called "calculate_ability_of_author" that takes the authors' name and returns a list contains:
  - the number of books that writer writes
  - the number of book categories that writer writes
  - average book ratings of one author
  - editor recommendations and best sellers.
- According to the authors' name passed in, find some lines containing this author, and calculate the average score of the book written by this author according to the data of these lines and add it to the ability list.
- Calculate the number of bestsellers owned by the author and the number of editor's recommendations. If the author has neither bestsellers nor editor's recommended books, then set Bestseller_EditorsPick to 0; if the author has both bestsellers and editor's recommended books, then set Bestseller_EditorsPick to 2. In other cases, the value is 1。
- For log(Am_book)+1 books, the range is too large, from one to more than two hundred books, but most of them are concentrated in one, so this method is used to make all the values clearly visible and reflect the differences.
- Find the number and category of books written by the author according to **asin**(similar to book id).
- The larger the Bestseller_EditorsPick, the more similar the author is to the taste of the market and editor, the better the price can be sold. That is, here we use five metrics - number of books, category of books, book ratings, editor recommendations, and bestsellers - to speculate on an author's ability.

In [25]:
```
def calculate_ability_of_author(author_name):
    ability=[]
    am_book=df[df["author"]==author_name] #Find the lines that contain the author based on author_name
    Stars=am_book["stars"].mean() #Calculate the average rating of the books written by this author

    ability.append(float(np.power(2.51189,Stars)))   #Add ratings to the ability list
    Bestseller=am_book["isBestSeller"].sum()   #Count the number of bestsellers that author has
    EditorsPick=am_book["isEditorsPick"].sum()  #Count the number of the author's books have been recommended by editors

    if Bestseller==0 and EditorsPick==0:
        Bestseller_EditorsPick=0
    elif Bestseller>0 and EditorsPick>0:
        Bestseller_EditorsPick=2
    else:
        Bestseller_EditorsPick=1

    ability.append(Bestseller_EditorsPick)  #Calculate the popularity of the author's book among the market and editors

    Am_book=len(am_book["asin"]) #Find the number of books written by the author based on "asin".
    Am_book=math.log(Am_book)+1
    ability.append(Am_book)
    Category_count=len(am_book["category_name"].unique())#Identify the genres of books written by the author based on "catego
    Category_count=math.log(Category_count)+1
    ability.append(Category_count)
    return ability
print(calculate_ability_of_author("A Zun Mo"))
```

```
[1.0, 0, 1.0, 1.0]
```

we successfully return four dimensions of a writer which can judge his ability.We know this author got 1 mark out of 100, and have not had a book been selected as editorPick or Bestseller. What's more, he only writes 1 book with 1 category. Then we try to figure out the weight to quantify the competence of author through this four-dimension.

use entropy weight method

- use the entropy weight method to calculate the weight of each index of the author
- use this weight to calculate the author's ability score
- the author's ability can be concretely displayed with the score, and prepare for the next step to predict the bestseller

In [26]:
```
df=BOOK_DF.copy()
ac = []
ad = []
empty_matrix = np.empty((0, 4))
j = 0
unique_authors = df["author"].unique().tolist()
for author in unique_authors:
    ability = calculate_ability_of_author(author)
    if ability is not None:  #Check that the result is None before appending
        ad.append(ability)
        j = j + 1
        if j==100:
            break
        print(f"Processing {j}\r", end="")

#Use loops to insert lists
for ability_list in ad:  #Change variable names to avoid conflicts with the list
    empty_matrix = np.vstack([empty_matrix, ability_list])

def entropy_weight(matrix):
    #Calculate the probability distribution for each column of the matrix
    row, col = matrix.shape
    p = np.zeros(col)

    for j in range(col):
        if np.sum(matrix[:, j]) == 0:
            pij = np.ones(row) / row
        else:
            pij = matrix[:, j] / (np.sum(matrix[:, j]) + 1e-10)
        pij[pij <= 0] = 1e-10
        p[j] = -np.sum(pij * np.log(pij))

    #Calculated entropy
    e = 1 - p / np.log(col)

    #Calculated weight
    w = e / np.sum(e)

    return w

weights = entropy_weight(empty_matrix)
print("weights: ", weights)
```

```
weights:  [0.3050774  0.07885428 0.30568336 0.31038496]
```

we successfully calculate the weight now.

[notion]: only small part of data because we got 70000 authors which can cost a lot of time. And the actual weight is (0.26727662, 0.19783145, 0.26673595, 0.26815598)

By using this weight, we can calculate The final score for each author, and written into the csv file. We upload this file on Github (click to see)
(https://raw.githubusercontent.com/qiaoyu0906/mygit/master/add_competence.csv)

In [27]:
```
ak={}
j=0
for i in list(df["author"].unique()):
    s=calculate_ability_of_author(i)
    k=s[0]*0.26727662+s[1]*0.19783145+s[2]*0.26673595+s[3]*0.26815598
    ak[i]=k
    j=j+1
    if j==100:
        break
    print(f"Processing {j}\r", end="")
a=BOOK_DF.copy()
# Open the CSV file and write the file
a=pd.read_csv("https://raw.githubusercontent.com/qiaoyu0906/mygit/master/add_competence.csv",encoding="UTF-8")
print(a.iloc[:,[1,17]])
```

```
              asin  competence
0       B00BNC8UPQ   20.268258
1       B00DLV0KN2   20.268258
2       B00EB9Z9WA   20.280218
3       B07TGF8NGV   20.280218
4       B07QN7R55B   17.399014
...            ...         ...
131570  B0CGF8ST46    1.635556
131571  B0CGF92SYR    1.635556
131572  B0CGF9LTC7    1.635556
131573  B006L8SE58   22.766144
131574  B00O561XG0   14.561854

[131575 rows x 2 columns]
```

After counting the quantified ability of author, we start to make this value more visiable.
We build the method **radar_chart()**

- Defines a function called 'radar_chart' that takes data_to_display and returns the all_data_display list
- Add all the ability lists (calculate_ability_of_author returns the ability list) to the list of all_data_display,

In [28]:
```
#create radar chart
def radar_chart(data_to_display):
    all_data_display=[]
    for i in data_to_display:
        all_data_display.append(calculate_ability_of_author(i))
    return all_data_display
```

The method **display_radar()**

- Defines a function called "display_radar" with the argument button_click, which is triggered by clicking a button.
- Check the checkbox to determine the authors' radar map that needs to be displayed.
- Create Radar map objects using the pyecharts library's radar class. The add_schema method is used to set the radar map indicators (the names and maximum values of each radar axis and configure some style parameters.
- The selected author data is added via the add method.
- Run the set_global_opts method to set global options.
- Adding the container radar_container ensures that the radar chart is only updated in the container at a time

In [29]:
```
#display a radar chart and buttons
def display_radar(button_click):
    selected_datasets = []
    for i in checkbox_dataset:
        if i.value:
            selected_datasets.append(i.description)

    radar=(
        Radar(init_opts=opts.InitOpts(bg_color="#CCCCCC"))
            .add_schema(
                schema=[
                    opts.RadarIndicatorItem(name="rating", max_=100),
                    opts.RadarIndicatorItem(name="Bestseller&Editorpick", max_=2),
                    opts.RadarIndicatorItem(name="amount of books", max_=6.5),
                    opts.RadarIndicatorItem(name="amount of categories", max_=4.5),
                ],
                splitarea_opt=opts.SplitAreaOpts(
                    is_show=True, areastyle_opts=opts.AreaStyleOpts(opacity=1)
                ),
                textstyle_opts=opts.TextStyleOpts(color="#fff"),
            )
            .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
            .set_global_opts(
                title_opts=opts.TitleOpts(title="基础雷达图"),
                legend_opts=opts.LegendOpts(type_="scroll",pos_left="right", orient="vertical", item_width=4, item_gap=5)
            )
    )


    chart_data=radar_chart(selected_datasets)#Enter the list of selected authors
    for i,j in zip(select_author,chart_data):
        radar.add(series_name=i,data=chart_data,areastyle_opts=opts.AreaStyleOpts(opacity=0.4),label_opts=opts.LabelOpts(is_s
    with radar_container:
        display(radar.render_notebook())
        clear_output(wait=True)
```

Set **clear_radar()**

- Defines a clear_radar function that clears the original graph when the button is clicked.

In [30]:
```
#Clear the component
def clear_radar(button_click):
    for i in checkbox_dataset:
        i.value=False
    clear_output()
```

Set **all_radar()**

- Defined a all_rada function to select all when the button is clicked.

In [31]:
```
#Select all component
def all_radar(button_click):
    for i in checkbox_dataset:
        i.value=True
```

Enter author's name (can be incomplete)

- Enter the author's name the users want to query, and form a list of all authors with input content. There are no duplicate elements in the list.
- If the list is longer than 100, select only the first 100 data and save the cover URL of one of the highest-rated books and the author name
- If there are no more than one hundred, save the cover URL of one of the highest-rated books and the author name

An interactive interface is created where the user can select an image through a check box and then perform the corresponding action by clicking a button

- Through the loop, this code creates a combination of images and check boxes for each author.
- select_author and select_image are the author's name and the corresponding image path list.
- Each combination includes an HTML-formatted label for displaying the image and a check box for selecting the author.
- create 'update radar', 'clear', 'select all' buttons, and when the users click them, respectively invoke ' display_radar ', 'clear_radar', 'all_radar' functions.

In [32]:
```
checkbox_dataset=[]
display_vboxset=[]

keyword=input("input part of author name to search:")  #Enter the author name the users want to query(Supports fuzzy querying
select_author=df[df["author"].str.lower().str.contains(keyword.lower())]["author"].unique().tolist()
select_image=[]

if len(select_author)>100:
    select_author=select_author[:100]
    for i in select_author:
        author_booklist=df[ df["author"]==i]
        best_stars=author_booklist["stars"].max()
        select_image.append(author_booklist[author_booklist[ "stars" ] == best_stars]["imgUrl"].tolist()[0])
else:
    for i in select_author:
        author_booklist=df[ df["author"]==i]
        best_stars=author_booklist["stars"].max()
        select_image.append(author_booklist[author_booklist[ "stars" ] == best_stars]["imgUrl"].tolist()[0])


for i,j in zip(select_author,select_image):
    html_label = HTML(f'<img src="{j}"height="200" width="100"/>')  #Create the check box and set the description in HTML for
    checkbox = Checkbox(value=False,description=i)   #Set the styles for checkboxes and images
    checkbox.layout.display = 'flex'
    checkbox.layout.align_self = 'center'   #Adjust the vertical alignment of the check box
    checkbox_with_image = HBox([html_label, checkbox])
    checkbox_dataset.append(checkbox)
    display_vboxset.append(checkbox_with_image)

button = widgets.Button(description='update radar')
button.on_click(display_radar)
button2 = widgets.Button(description='clear')
button2.on_click(clear_radar)
button3 = widgets.Button(description='select all')
button3.on_click(all_radar)
button_row = widgets.HBox([button, button2])  #Display interactive component
buttons=widgets.HBox([button,button3,button2])


radar_container=widgets.Output()
    #radar_container.set_trait()

button.on_click(display_radar)

def display_author_radar():
    display(VBox(display_vboxset))
    display(buttons)
    display(radar_container)
```

```
input part of author name to search:jk
```

Objective 4

We first extract the outcome that we generate in objective 3a and 3b, and merge them in merged_df which will feed into our decision tree model. Factors include:

- Price of the book
- Category of the book
- Seller's ability(calculate in objective 3a)
- Author's ability(calculate in objective 3b)
- The publicaition date

In [33]:
```
author_df = a[["author","competence"]]
sold_df = BOOKdf[['price', 'category_id','soldBy','publishedDate','isBestSeller','author']]
sold_df1 = sold_df.copy()
sold_df1['isBestSeller'] = sold_df1['isBestSeller'].astype(int)
sold_df1 = sold_df1.dropna(subset=['soldBy'])
#  start merge
soldby_df = outer[['soldBy', 'score']]
merged_df = pd.merge(sold_df1, soldby_df, on='soldBy', how='left')#pd.merge
merged_df = pd.merge(merged_df, author_df, on='author', how='left')#pd.merge
merged_df['publishedDate'] = pd.to_datetime(merged_df['publishedDate'], format='%Y/%m/%d')  #change str to datetime type
start_date=merged_df['publishedDate'].min() #find the earliest day as start
merged_df['since_days_start'] = (merged_df['publishedDate'] - merged_df['publishedDate'].min()).dt.days #get the length of da
day_df = merged_df[['publishedDate','since_days_start']]

merged_df = merged_df.dropna(subset=['author'])
merged_df_show = merged_df.copy()

merged_df = merged_df.drop('soldBy', axis=1)
merged_df = merged_df.drop('author', axis=1)
merged_df = merged_df.drop('publishedDate', axis=1)

merged_df = merged_df.rename(columns={'score': 'soldBy'})# rename 'score' to 'soldBy'
merged_df = merged_df.rename(columns={'competence': 'author'})# rename 'competence' to 'author'
merged_df = merged_df.rename(columns={'since_days_start': 'publishedDate'})# rename 'since_days_sart' to 'publishedDate'

merged_df
```

Out[33]:

	price	category_id	isBestSeller	soldBy	author	publishedDate
0	9.99	3	0	5.490493	20.268258	41296
1	9.99	3	0	5.490493	20.268258	41296
2	9.99	22	0	5.490493	20.268258	41305
3	9.99	22	0	5.490493	20.268258	41305
4	19.99	18	0	5.636251	20.280218	41938
...	...	...	...	...	...	...
858041	2.99	8	0	5.490493	1.635556	45180
858042	2.99	8	0	5.490493	1.635556	45180
858043	2.99	8	0	5.490493	1.635556	45180
858044	13.99	30	0	5.588524	22.766144	39390
858045	3.49	14	0	5.490493	14.561854	41912

858046 rows × 6 columns

We build a decision tree to classify whether a book can be a bestseller.

In [34]:
```
shuffled_df = merged_df.sample(frac=1, random_state=42) #shuffle the data make it random

X = shuffled_df.drop("isBestSeller", axis=1)
y = shuffled_df["isBestSeller"]

# divide train 80% and test part 20%
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, random_state=42)

k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# k cross-validation
for train_index, test_index in kf.split(X_train1,y_train1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    #obtain the indices for the training and testing sets, and retrieve the corresponding X and y based on these indices.

    # choose the hyperparameters
    clf = DecisionTreeClassifier(max_depth=10, class_weight='balanced', min_samples_split=20, min_samples_leaf=35,
                                 random_state=50)
    #train mode
    clf.fit(X_train, y_train)

    # show accuracy
    accuracy = clf.score(X_test, y_test)
    print("accuracy:", accuracy)

#judge the tree
# from sklearn.metrics import confusion_matrix
y_pred = list(clf.predict(X_test1))#predict by X_test

# print report
class_report = classification_report(y_test1, y_pred)#truth ground (y_test) and prediction (y_pred) category report
print("class report:\n", class_report)
conf_matrix = confusion_matrix(y_test1, y_pred)#get confusion matrix
print("confusion matrix:\n", conf_matrix)

recall = conf_matrix[1, 1] / (conf_matrix[1, 1] + conf_matrix[1, 0])
print("recall:", recall)
```

```
accuracy: 0.8027649903851757
accuracy: 0.8435831506260607
accuracy: 0.83192873323767
accuracy: 0.8696671935434528
accuracy: 0.8736005594120346
class report:
               precision    recall  f1-score   support

           0       0.99      0.88      0.93    168428
           1       0.10      0.71      0.17      3182

    accuracy                           0.87    171610
   macro avg       0.55      0.79      0.55    171610
weighted avg       0.98      0.87      0.92    171610

confusion matrix:
 [[147843  20585]
 [   928   2254]]
recall: 0.7083595223130107
```

And we build **show_decision_tree()** to visualize our decision tree. And judge which factor is most important or may be the key for a book to become bestseller.

In [35]:
```
def show_decision_tree():
    #the visualisation of decision tree
    plt.figure(figsize=(120, 80))
    tree.plot_tree(clf, feature_names=X.columns.to_list(), class_names=["0", "1"], filled=True)
    plt.show()

def find_important_feature():
    # find wich feature is most important
    feature_importance = clf.feature_importances_

    feature_importance_dict = dict(zip(X.columns, feature_importance))

    # sorted to find the most important one
    sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

    # print out
    print("Sorted Feature Importance:")
    for feature, importance in sorted_feature_importance:
        print(f"{feature}: {importance}")
```

Since we already have constructed a well-weighted decision tree, we can now use this model to predict. By giving the sufficient factor information we can then use **predict(price,category,soldby,author,date)** to classify the book.

In [36]:
```
def predict(price,category,soldby,author,date):
    # change input information to dataframe
    input_data = pd.DataFrame({'price': [price], 'category_id': [category], 'soldBy': [soldby], 'author': [author], 'publishe
    # use clf to predict
    prediction = clf.predict(input_data)
    # give the result
    with output:
        output.clear_output()
        time.sleep(2)
        if prediction[0] == 1:
            print("This may be bestseller.")
        else:
            print("This may not be bestseller.")
```

We use **ipywidgets** to make it convenient for user to use our model. The following is how we program this system.

In [37]:
```
soldby_values = merged_df_show['soldBy'].unique().tolist()
dic_cate_id={}
for i in category:
    cate=BOOK_DF[BOOK_DF["category_name"]==i]
    dic_cate_id[i]=cate.iloc[0,10]
```

We have developed **get_selected_date()** to determine the duration between the date selected by the user and the **start_date**.

In [38]:
```
#define the function which can obtain date
def get_selected_date():
    year = year_dropdown.value
    month = month_dropdown.value
    day = day_dropdown.value

    date_str = f"{year}-{month}-{day}"

    date = datetime.strptime(date_str, '%Y-%m-%d')

    calculate_days = (date - start_date).days
    return calculate_days
```

To make the date widgets more reasonable, we build **update_day_dropdown(change)** to update the option of day once we detect change of year and month.

In [39]:
```
# define the function
def update_day_dropdown(change):
    year = year_dropdown.value
    month = month_dropdown.value
    # update by selected year and month
    if month in [1, 3, 5, 7, 8, 10, 12]:
        day_options = list(range(1, 32))
    elif month in [4, 6, 9, 11]:
        day_options = list(range(1, 31))
    else:
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            day_options = list(range(1, 30))
        else:
            day_options = list(range(1, 29))
    #update day_options
    day_dropdown.options = day_options
```

Also, we will hear whether the user click the button, if he or she does so, we will get the information from form, and try to use these to predict by calling **predict(price,category,soldby,author,date)**

In [40]:

```
def on_button_click(b):

    price= price_input.value
    category=cate_dropdown.value

    date=int(get_selected_date())


    soldby=soldby_dropdown.value

    #calculate author's ability
    stars_value=stars_input.value
    bestseller=best_dropdown.value
    editorpick=edit_dropdown.value

    stars_value=float(np.power(2.51189,float(stars_value)))   #Add ratings to the ability list

    Bestseller_EditorsPick=0
    if bestseller=="have not" and editorpick=="have not":
        Bestseller_EditorsPick=0
    elif bestseller=="have" and editorpick=="have":
        Bestseller_EditorsPick=2
    else:
        Bestseller_EditorsPick=1

    Am_book=math.log(int(number_books_input.value))+1

    Category_count=math.log(int(number_cate_input.value))+1

    author=stars_value*0.26727662+Bestseller_EditorsPick*0.19783145+Am_book*0.26673595+Category_count*0.26815598

    predict(price,dic_cate_id[category],outer[outer["soldBy"]==soldby].iat[0,5],author,date)
```

The following code is the main body of how we build the form. And we try to get information from users to help us predict whether his or her new books are likely to be bestsellers.

In [41]:

```
# build input textbox
price_input = widgets.Text(description='Price:')
cate_dropdown = widgets.Dropdown(
        options=category,
        description='Category:'
    )
#date information
year_range = list(range(1900, 2030))
year_dropdown = widgets.Dropdown(
    options=year_range,
    description='Year:',
    layout=widgets.Layout(width='auto')
)

month_dropdown = widgets.Dropdown(
    options=list(range(1, 13)),
    description='Month:',
    layout=widgets.Layout(width='auto')
)

day_dropdown = widgets.Dropdown(
    description='Day:',
    layout=widgets.Layout(width='auto')
)
date_picker = widgets.HBox([year_dropdown, month_dropdown, day_dropdown])
soldby_dropdown = widgets.Dropdown(
        options=soldby_values,
        description='Soldby:'
    )
stars_input = widgets.Text(description='Stars:')
best_dropdown = widgets.Dropdown(
        options=["have","have not"],
        description='BestSeller:'
    )
edit_dropdown = widgets.Dropdown(
        options=["have","have not"],
        description='EditorsPick:'
    )
number_books_input = widgets.Text(layout=widgets.Layout(width='150px') )
number_cate_input = widgets.Text(layout=widgets.Layout(width='150px') )
button = widgets.Button(description="predict")
output = widgets.Output()

def display_format():
    display(price_input)
    display(cate_dropdown)
    display(date_picker)
    year_dropdown.observe(update_day_dropdown, 'value')
    month_dropdown.observe(update_day_dropdown, 'value')
    update_day_dropdown(None)
    #soldby information
    display(soldby_dropdown)
    # information about writer himself
    display(stars_input)
    display(best_dropdown)
    display(edit_dropdown)
    display(widgets.HBox([widgets.Label('Num of books written:'), number_books_input]))
    display(widgets.HBox([widgets.Label('Num of categories involved:'), number_cate_input]))
    # build submit button
    display(button)
    button.on_click(on_button_click)
    display(output)
```

Project Outcome

Overview of Results

We successfully reach our goals:

- We are able to help authors predict whether their upcoming book has the qualities to be a bestseller.
- Based on first orientation, it can show the correlation between ratings, prices bestsellers, editor recommendations, and good choices and visualize them as scatter plots.
- According to the second orientation, we draw a heap diagram and can use this diagram to find:
    1. The market share of different types at different time nodes.
    2. Distribution of bestsellers among different categories.
    3. The reputation of different types in the market at different time nodes (stars).
- Create a line chart and be able to use it to show trends in the number of books published and word-of-mouth for each category over time.
- According to the third orientation, we quantified the abilities of authors and sellers and presented them using a radar chart.
- According to the fourth orientation, we achieved the qualities that help authors predict whether their forthcoming book will be a bestseller. Using the previously obtained date, category, price, seller ability, and author ability to draw a decision tree, through the decision tree to help the authors in need to know whether their forthcoming books can be best-selling, to a certain extent, it can also help them to choose the book pricing, seller selection, publication time.

Objective 1

To gain a preliminary understanding of this dataset, we explored the relationship between various attributes and the dimensions of “price” and rating “stars”, visualized through scatter plots.

Explanation of Results

Looking at the concentration distribution depicted in the scatter plot, we derived the following results:

- **Kindle Unlimited Preferences:** Kindle Unlimited(nearly 27%) tends to select books priced below 120 pounds, allowing for a relatively wide pricing range. In the meanwhile, Kindle Unlimited books appear to be popular with readers, always gaining higher ratings.
- **Inclusivity of 'EditorsPick':** “EditorsPick” books occupy 4.23%, demonstrating that editors take an inclusive consideration in both pricing and ratings. One of the possible reasons for this is that editors might have different standards for evaluating books compared to readers. In terms of pricing, there may not be a direct correlation between the price and the content of a book, so books recommended by editors may not necessarily be priced too high.
- **Strict standards of 'GoodReadsChoice':** 'GoodReadsChoice' books make up 0.93%, indicating higher standards, demanding affordable pricing and reader acclaim. In addition, it is worth noting that there are no books with stars below 3.6, while the other categories in these results contain books with ratings ranging from 0 to 3.6 stars.
- **The potential of becoming bestsellers:** The percentage of 'Bestsellers' is 1.67%, showing that lower prices and higher ratings of books are likely to become bestsellers. Lower prices may suggest higher cost-effectiveness, while higher ratings may reflect better book quality, more in line with the reader’s book-buying standards.
- **Lack of excellent labels:** Obtaining excellent labels is relatively challenging, resulting in a large share of books not having any of the 'BestSeller,' 'EditorsPick,' and 'GoodReadsChoice' labels. Besides, we regret to find that none of the books has received multiple excellent labels.

Visualisation

Figure 1 illustrates a scatter plot of the relationship between e-book prices and rating stars, with price on the x-axis and stars on the y-axis. In this plot, black dots showcase where 'isKindleUnlimited' is “False”(labeled as “N_KU”), while red dots showcase where 'isKindleUnlimited' is “True”(labeled as “KU”).

Similarly, Figure 2 utilizes the same x-axis (price) and y-axis (stars), which are differentiated with different colors on all conditional combinations of "isBestSeller" "isEditorsPick" and "isGoodReadsChoice". After reviewing the combinations, we found four distinct combinations present in the dataset. They are denoted as "NS_NE_NG", "S_NE_NG", "NS_E_NG" and "NS_NE_G" in the figure2, where "N" shows the absence of the condition, "S" represents "isBestSeller," "E" means "isEditorsPick," and "G" indicates "isGoodReadsChoice".

Figure 3 shows the overlap of Figure 1 and Figure 2. When labels from Figure 1 and Figure 2 both appear, the scatter points present a combination of diamond and square shapes. A smaller number of the books with the premium label in Figure 2 are also Kindle Unlimited.

In [42]:
```
html,notebook=Stars_and_variable_relationship()
print(html) #output a html
notebook
```

```
C:\Users\ASUS\Desktop\学校课程\数据科学编程\cw3\objective1.html
```

Out[42]:

Figure 1 Figure 2 Figure 3

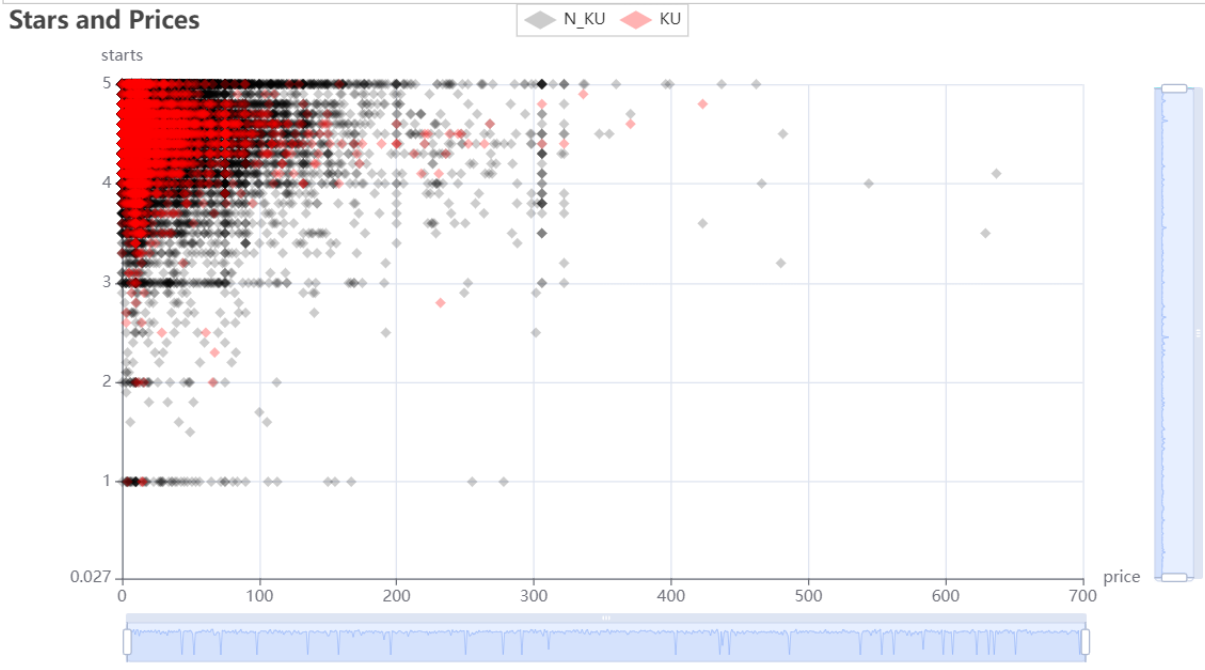


Figure 1 illustrates a scatter plot of the relationship between e-book prices and rating stars, with price on the x-axis and stars on the y-axis. In this plot, black dots showcase where 'isKindleUnlimited' is “False”(labeled as “N_KU”), while red dots showcase where 'isKindleUnlimited' is “True”(labeled as “KU”).

The scatter plot shows that KindleUnlimited books are predominantly concentrated in the price range below 120 pounds and within the 3.5-5 stars rating interval. Besides, a significant proportion of Kindle e-books, including both KindleUnlimited and non-KindleUnlimited, are priced range within 0-300 pounds and have ratings from 2 to 5 stars.

Similarly, Figure 2 utilizes the same x-axis (price) and y-axis (stars), which are differentiated with different colors on all conditional combinations of "isBestSeller" "isEditorsPick" and "isGoodReadsChoice". After reviewing the combinations, we found four distinct combinations present in the dataset. They are denoted as "NS_NE_NG", "S_NE_NG", "NS_E_NG" and "NS_NE_G" in the figure2, where "N" shows the absence of the condition, "S" represents "isBestSeller," "E" means "isEditorsPick," and "G" indicates "isGoodReadsChoice".

The scatter plot and book statistics show that the majority of 'NS_NE_NG'(consisting of 122,501 e-books) are priced in the range of 0-135 pounds and star ratings falling between 3 and 5. 'NS_E_NG' contains 5,577 books, mainly distributed in the price range of 0-40 pounds with star ratings above 3.7. 'NS_NE_G' includes 1,219 books with a minimum rating of 3.6 stars, holding a large number of books priced below 20 pounds and rated higher than 4.1. 'S_NE_NG' maintains 2,196 books, predominantly distributed in the price range from 0 to 20 pounds stars with a rating of 4 stars and above.

Figure 3 shows the overlap of Figure 1 and Figure 2.

Objective 2

Explanation of Results

We examined the annual publication trends of various types of books over time and discovered numerous phenomena.

- Firstly, we find that children eBooks mainly got the highest total number from 1806-2023, since we rank the categories by the number of books that it published that year.
- Secondly, As the years progress, there is a noticeable increase in the variety of book genres being published. Moreover, various genres are beginning to have their own bestsellers, rather than being limited to a few specific categories.
- Another noticeable thing is that the majority of categories got highest average stars around 1975-1998, which we could conclude that people favours books that have been around longer.
- Last but not least, it seems that some categories exhibit a phenomenon where when a book receives a high rating, there is a sharp increase in the number of similar books published the following year. However, as the quantity of books in the market grows, the ratings start to decline. This could be one of the reasons for the complex and fluctuating nature of ratings.

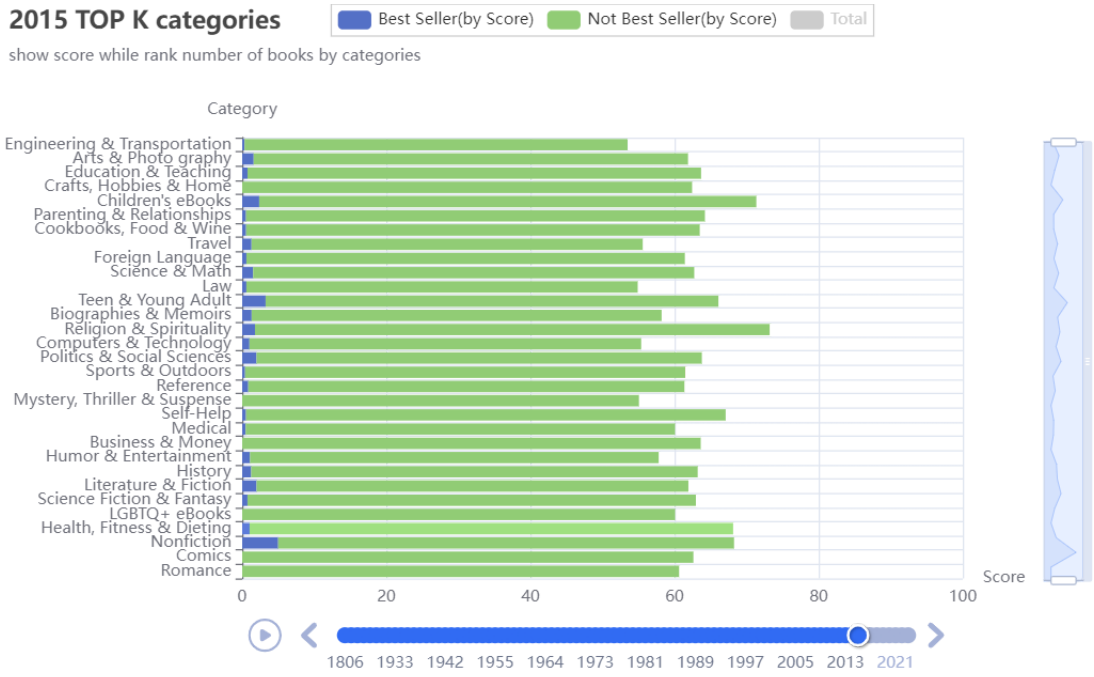
Visualisation

Chart usage

- The horizontal coordinate of this graph is the year, which can be increased over time (the users can also manually select a year or click the arrow button to see the situation of the previous year and the next year), and the vertical coordinate is the category, which can be manually selected. The blue legend represents the best seller, the green legend represents the non-best seller, and the orange legend represents the number of books (the orange legend does not need to be checked).
- The bar chart presented in this chart represents the best seller and non-best seller ratings for each selected category in a given year, sorted from top to bottom by the total number of books. First of all, we need to calculate the average score of all books in this category. After calculation, it is found that the difference between the average

In [43]:
```
draw_top_k_categories().render("top k categories.html")
draw_top_k_categories().render_notebook()
```

Out[43]:



in this chart, we can look at the popularity of certain categories in a particular year, so as to predict the social trend and background at that time.

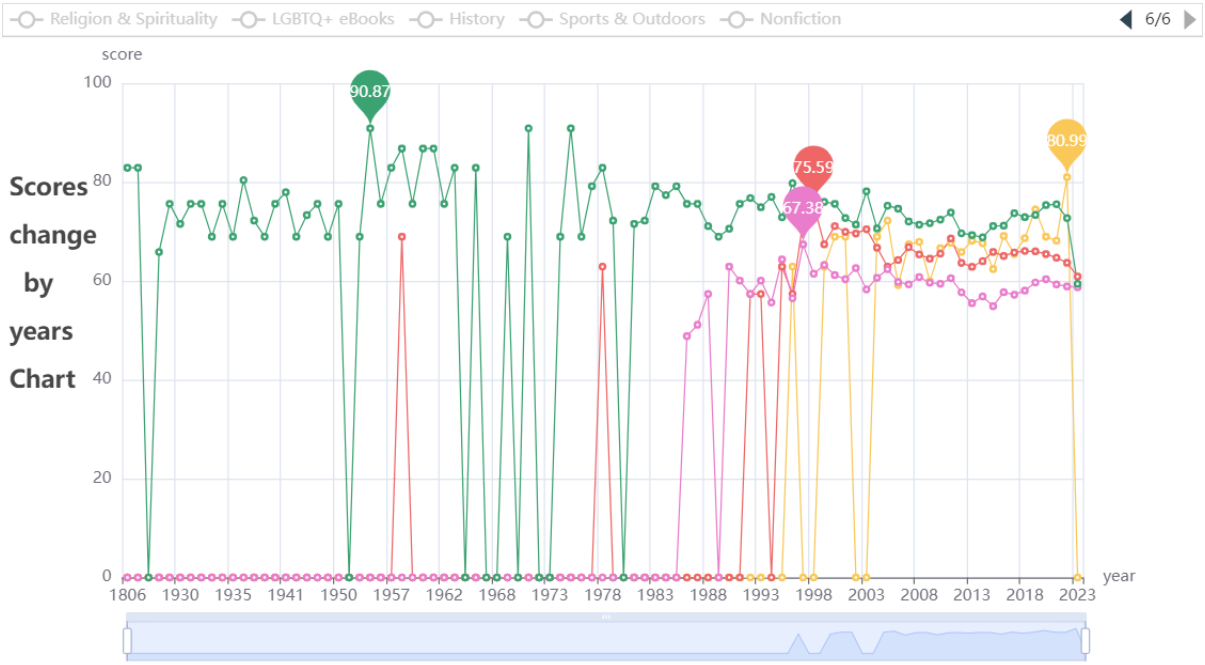
- In 2012-2021, the nonfication category has the largest number of bestsellers.

Also it can prove what we conculde in the result part:

- According to the figure above, we can find that 'Childrens' book' was the only kind of book sales on the market from 1937 to 1941. Until 1941, other types gradually appeared.
- It was not until 1979 that the first best-seller category, childrens' books, appeared. In 2001, the categories of books was 31.

In [44]:
```
draw_top_k_categories().render("score_line_chart.html")
draw_line_score_chart().render_notebook()
```

Out[44]:

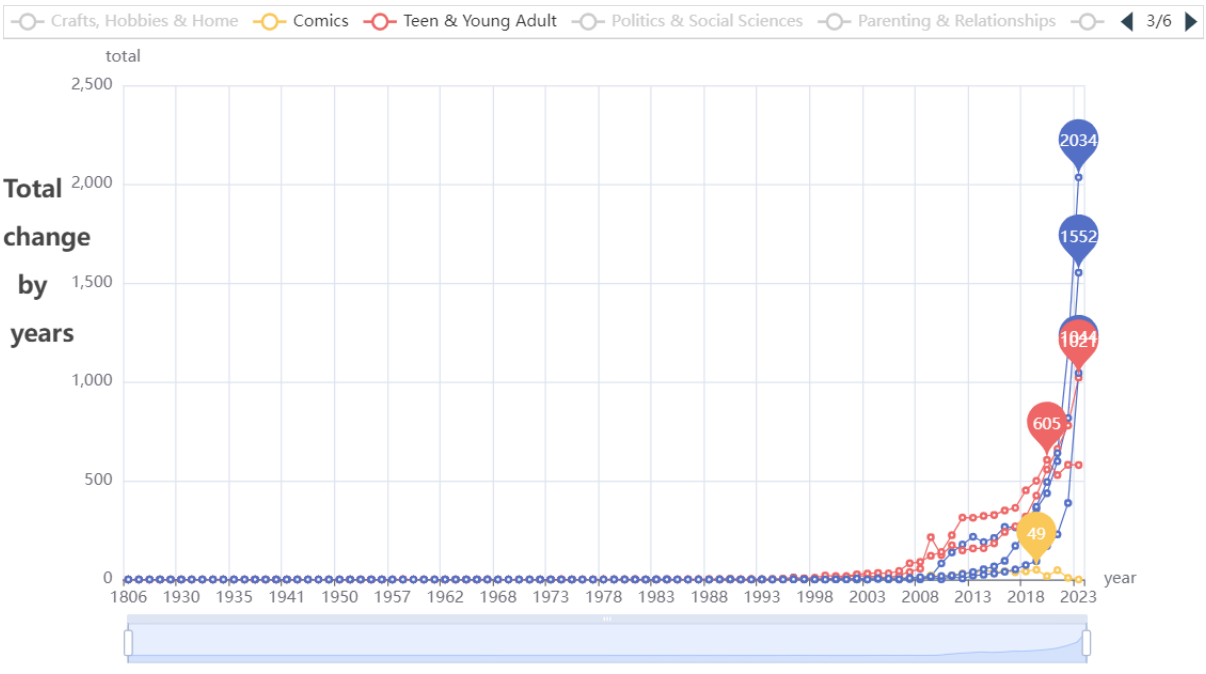


Through the second figure we can clearly see:

- **When the category start springing up**. From the example figure we can find that Reference books started being rated by customer at 1972, while people rated books about Bussiness and Money much earlier, which is in 1914.
- **The highest score that a category have from 1806-2023**. "Reference" type of books achieved thier highest score in 1985. Meanwhile year 1995 witnessed "Business and Money"'s score reached to the top with 75.59.
- **Compare when the category reach their highest score**. If we select all lables we can find that the **majority** of categories reach to top around 1975-1998. However people favoured **Education & Teaching**, **Science & Math**, **children's ebooks** which published much earlier long before 1960. **Comic books** different from others reached its highest score in 2022, this shows that people might love reading new comic books rather than those classic or maybe outdated comic books.
- **We can compare the highest scores between different types of books.** We can see that Childrens' ebooks and reference got the highest mark, which all books published in one year reached 90.87. While people seldom gave high marks for **Mystery, Thriller & Suspense** which only earn 67.38 in average.

In [49]:
```
draw_top_k_categories().render("total_number_line_chart.html")
draw_line_total_chart().render_notebook()
```

Out[49]:



According to the third figure we can find out that:

- How the number of different categories change is hard to compare at the very beginning years. So for this part we need to watch the first figure.
- Most categories of books experienced a significant surge in the number of book releases around the year 2000, except comics.

When we combine the 2 graphs together, we get some interesting views:

- One typical example to prove what we have said in result part is "Teen & Young Adult" category, it got its highest mark in 1998,and from then people crowed in writing this kind of books.

Objective 3

Explanation of Results

Sellers' Section

As shown by the radar chart comparison and quantified data, we derived the following results from the standpoint of the seller and author:

- **Being a professional company :** De Marque are one of the more specialised book companies with a wider range of books, whose reader interaction and book ratings are better for other companies to reference.
- **The company with a series of labels:** The "Pottermore" is higher rated compared to other companies.When we compare companies similar to them, i.e. companies that both have book series(like Disney Book Group,Marvel Entertainment US), we can see that they all have a higher star ratings and the amount of reader interaction isn't bad either. we advise vendors to think about developing a brand sequence modeled by popular properties such as Disney, Marvel, and Harry Potter to boost the amount of sales.
- **Smaller book firms:** For publishers and sellers who focus on one book categories and have low interaction and scores, It could be a social phenomenon，but it may be beneficial to experiment with new themes or find ways to increase interest with their books.

Authors' Section

- After quantifying author capabilities, it was found that the author with the highest score is Matt Abrahams, with a rating of 5, 2 books, and 2 genres, including one bestseller. Among the authors with the lowest and tied scores, there are a total of 2219 authors. Their ratings are all 0, with no tags for editor recommendations or bestsellers, and they have only 1 book and 1 genre each.
- It is observed that the majority of authors ranking higher have written more than two books, with the number of genres exceeding one. Therefore, we recommend authors write as many books as possible and broaden their knowledge by reading more. Doing so can enhance their overall score.

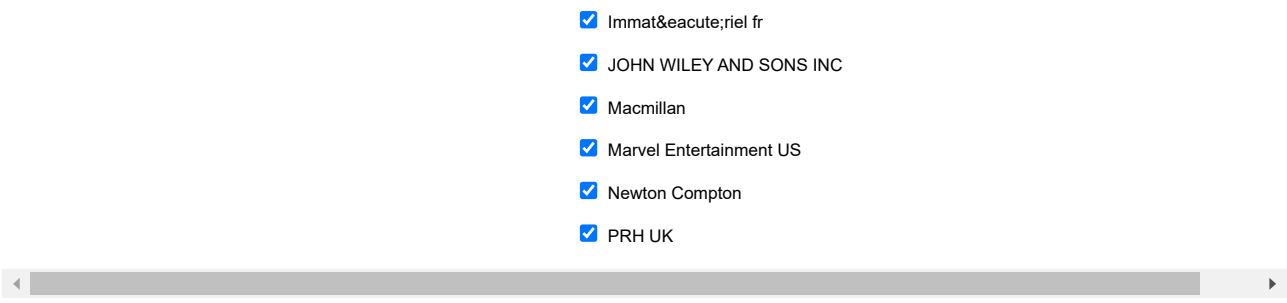
Visualisation 1: sellers' section

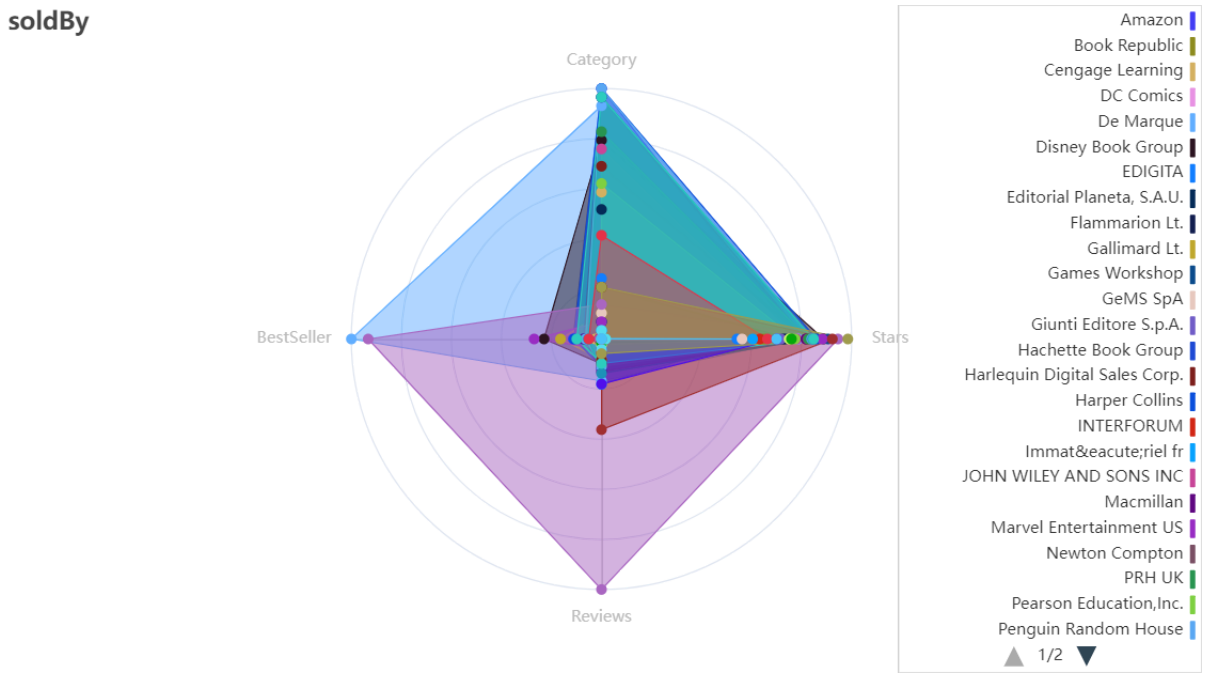
The initial interface shows a combination of multiple checkboxes, which represent all the sellers, and a submit button. After selecting the sellers you want to view the situations and clicking the “submit” button the corresponding radar chart will be displayed.

The radar chart with the four dimensions of “Category”, “BestSeller”, “Stars”, and “Reviews”. The range of each dimension axis is determined based on the maximum and minimum values of those data sources. Specifically, there are 1-30 types of categories, so the range of value on “Category” is between 1 and 30. Similarly, “BestSeller” denotes the proportion of bestsellers in all books sold, and “Stars” and “Reviews” represent the average stars ratings and average reviews per sold book respectively.

The checkboxes can be selected and submitted multiple times. After selecting the "Select All" checkboxes and submitting, a radar chart is generated containing all merchants, showcasing their performance on these four dimensions. Although most merchants may lack the advantage of "Reviews" and "Bestsellers", Pottermore and De Marque have the highest number of book reviews(20,976 per book) and bestsellers(0.16%) respectively. Moreover, both Pottermore and De Marque demonstrate the highest overall strength, crossing three dimensions, with the former ranking first (achieving the highest values in "Stars" and "Reviews") and the latter ranking second. In addition, Penguin Random House and Amazon provide the most categories of the books.

In [46]:
```
show_the_seller_information()
```



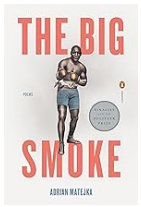


Visualisation 2: authors' section

- The author's radar chart comprises four indicators. 'Rating' represents the average rating of all books written by the author, 'Bestseller & Editor's Pick' indicates the rating based on the number of times the author's works have been recommended by editors and have become bestsellers, 'Amount of Books' denotes the total number of books written by the author, and 'Amount of Categories' reflects the variety of genres covered by the author's works. These four dimensions are used to evaluate the author's capabilities, with a larger radar chart area indicating a stronger overall ability. **Chart usage**
- After running the function, an input box will first appear, allowing the user to enter the name of the author they want to explore (supporting fuzzy search). After entering the name and pressing Enter, all authors that meet the search criteria along with the covers of their representative works will be displayed. Once the user selects an author of interest, clicking the "Update Chart" button will reveal a radar chart depicting the chosen authors.

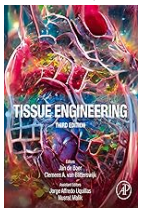
Sellers' and authors' capabilities are assessed to help us make forecasts。

In [47]:
```
display_author_radar()
```

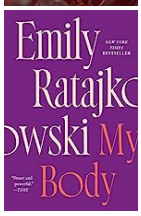


☑ Adrian Matejka

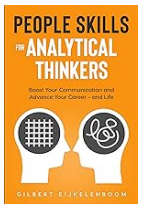
☑ Aga Bojko



☑ Clemens van Blitterswijk

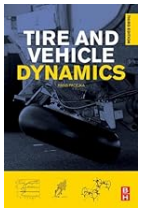


☑ Emily Ratajkowski



☑ Gilbert Eijkelenboom

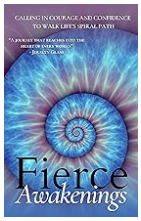
☑ Gojko Adzic



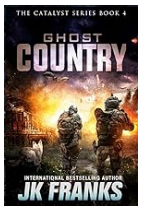
☑ Hans Pacejka

☑ Hans van Dijk

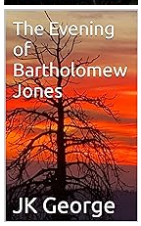
☑ Jan van Dijk



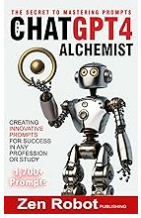
☑ Jennifer Baljko



☑ JK Franks

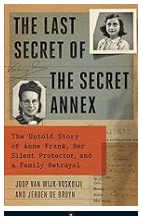


☑ JK George



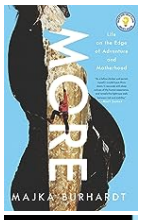
☑ Jk Novak

☑ JKSManga

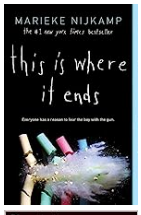


☑ Joop van Wijk

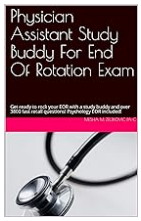
☑ Ken Matejka



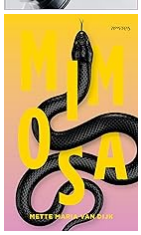
☑ Majka Burhardt



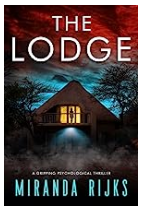
☑ Marieke Nijkamp



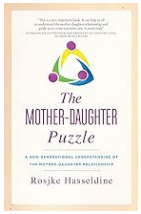
☑ Mesha M. Zeljkovic PA-C



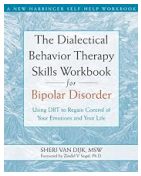
☑ Mette Maria van Dijk



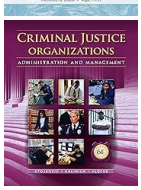
☑ Miranda Rijks



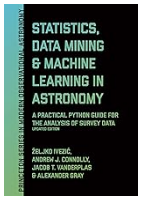
☑ Rosjke Hasseldine



☑ Sheri Van Dijk



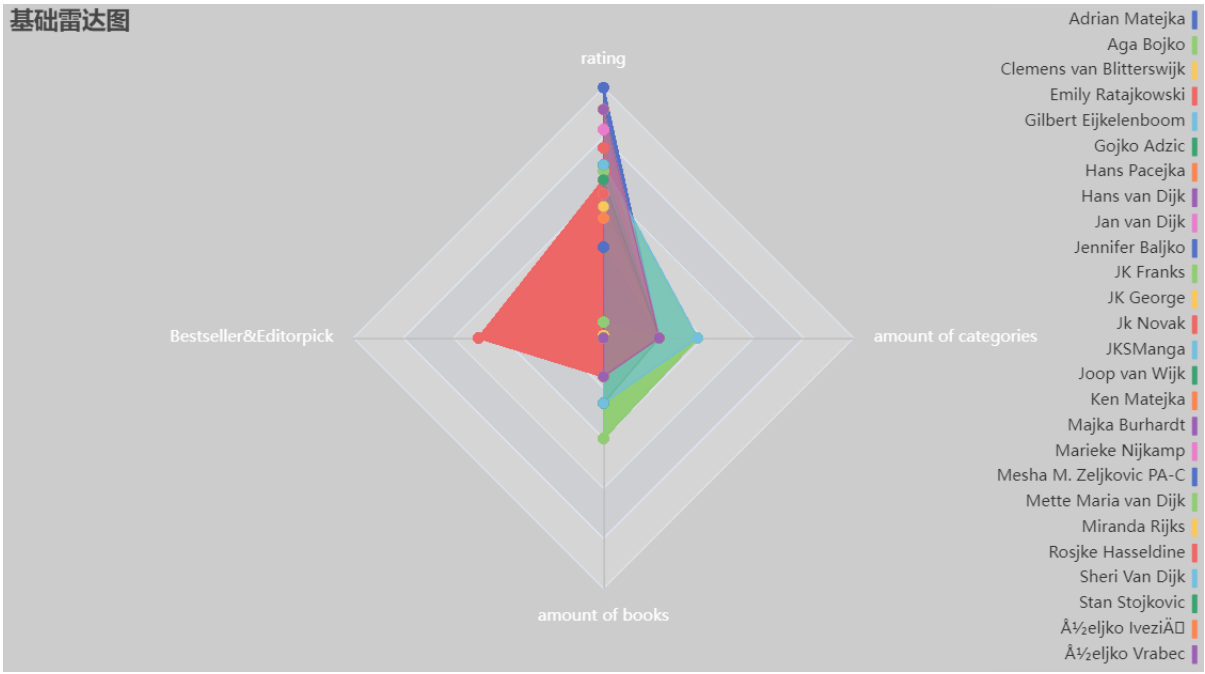
☑ Stan Stojkovic



☑ Å½eljko IveziÄ□

☑ Å½eljko Vrabec

update radar select all clear



Objective 4

Explanation of Results

We used features such as price, category, seller rating, author rating, and publication date, with a target label(whether the book is a bestseller) to train a decision tree model. The following is the result of our classification model:

- During the process of training the tree model, the average accuracy of K-fold cross-validation(K=5) is approximately 0.834.
- The model has a recall of 0.71 in predicting books as bestsellers, which means the model can successfully identify 71% of all samples that truly are bestsellers.
- Due to the significant imbalance in number of samples between bestsellers(label=1) and non-bestsellers(label=0) in the dataset(3182: 168428), predicting non-bestsellers will be noticeably better than predicting bestsellers with the model. Examining the F1-score, which considers both precision and recall of the classification model, the performance of the model in different classes (label=0 and label=1) can be easily observed, with scores of 0.93 and 0.17 respectively.
- The most influential factor in determining whether a book is a bestseller is the publish date, contributing greatly to the classification with a weight of 0.33. On the contrary, the least influential factor is the seller, with a minimum weight contribution of 0.06. The rest of the factors contribute very close value, ranked in descending order of importance as follows: author(0.22), price(0.19), and category(0.19).

Visualisation

We asked the user to fill out the form. After the user submit by click **predict button** we then feed this data into our decision model. You need take care the format when you are completing the form, this format is easy and doesnot have a high robust to bear the wrong behaviour of user, though we tried our best to avoid this kind of thing happen, like giving dropdown box and give the default value.

- Price need to be a float number, any other type may trigger error.
- Need to choose the day otherwise error may occur
- Stars means the average stars that you recieved for all of your book, need to range from 0-5 though this won't trigger error but lead you to a wrong prediction
- Num of books and Num of categories need to be integer, any other numeric input will not make the code break down but may give you a wrong advice

In [48]:
```
display_format()
```

Price: 9.9

Category: Foreign Language

Year: 2024 Month: 9 Day: 6

Soldby: Pottermore

Stars: 4.4

BestSeller: have not

EditorsPick: have

Num of books written: 4

Num of categories involved: 2

predict

```
This may be bestseller.
```

Conclusion (5 marks)

Achievements

- We find that kindle e-books are more popular, the books recommended by editors have little relationship with price and stars, and the books with good choice labels have generally higher ratings, and only 0.93% of books have good choice labels. So we speculate there are more stringent screening criteria for good choices.
- We can predict the social context of the time, show the market share of different types at different time nodes, distribution of bestsellers among different categories, the reputation of different types in the market at different time nodes (stars), trends in the number of books published and word-of-mouth for each category over time.
- In terms of authors, we find that the top authors tend to have many strengths(more categories and books) and have significant advantages in terms of ratings. In terms of sellers, the big book companies focus on high-quality books and provide customers with various kinds of books. Companies with multiple brands, such as Disney, have higher reader engagement and ratings.
- By using our previous work, we create machine learning models and predictions that can help authors in need know whether their upcoming books will be the best seller, and help them decide on pricing, seller selection, and publication time.

Limitations

- Visualization: in the radar chart, there is a maximum value of 20977 in "reviews"(not an extreme value that needs to be removed), while the majority of them have fewer than 3,500 reviews. As a result, after selecting all merchants and submitting them, the comparison is not clear in the radar chart.
- Classification model: there is a significant imbalance in the quantity of the two classes(bestsellers and non-bestsellers) in our dataset since becoming a bestseller is challenging for the majority of books. At present, we do not utilize other effective methods to address this class imbalance problem. Just adjusted parameters to enhance the model training, using "class_weight='balanced'".
- In the dataset, the number of pages for each book is not provided. If there were pages recorded in the dataset, there might be other interesting discoveries.

Future Work

- Although the radar chart highly intuitively shows the overall performance of sellers, data disparities are a problem. Thus, we will continue to explore more suitable visualization methods in the future.