Preprocessing data

goals:

1. Clean data

a. Names has the issue of no consistency, there are random symbols inside the names.

b. Author and Narrator both have WrittenBy and NarratedBy before the names.

c. Time is represented as 2 hrs and 20 mins we can change it to be the total minutes for easier analysis.

d. releasedate is also represented differently throughout the different rows. I need to standardize that.

e. stars is repersented awkwardly as well, instead of just the rating it's presented as '5 out of 5 stars34 ratings' since there all out of 5 what we really need is the initial number

f. price, throughout the price column you'll see some being represented as floats and others as int. i'll standardize those so that they resemble more closely to financial information.

importing necessary libraries and file path

In [ ]:
```
import pandas as pd #Pandas for data manipulation
import numpy as np #Numpy for simple analysis and numerical functions
import re #Re for text manipulation

data = pd.read_excel("C:\\Users\\vturn\\OneDrive\\audiobooks.xlsx")
data = pd.DataFrame(data)
```

EDA

In [ ]:
```
print(data.shape) #shape to understand the number of columns and rows in the dataset
```

```
(87489, 8)
```

In [ ]:
```
print(list(data.columns)) #columns to get the column names.
```

```
['name', 'author', 'narrator', 'time', 'releasedate', 'language', 'stars', 'price']
```

In [ ]:
```
print(data.isna().sum()) #isna() and sum() for the total number of missing values.
```

```
name           0
author         0
narrator       0
time           0
releasedate    0
language       0
stars          0
price          0
dtype: int64
```

In [ ]:
```
print(data.dtypes) #checking for data types per column.
```

```
name           object
author         object
narrator       object
time           object
releasedate    object
language       object
stars          object
price          object
dtype: object
```

one thing i've noticed so far is that all column names are listed as objects, which wouldn't make sense for price and stars. Also, a personal preference but the column names are lower case, i'll change that to be capitalized. I'll start with cleaning up the price data column, I expected it to be a float or an int since its an object that suggest to me that there are some string values in the dataset. I'll find them and handle them.

In [ ]:
```
def check_for_strings(df, column):
    unique_values = set() # Creating a set to hold the unique values(since they cant hold duplicates)
    for value in df[column]:
```

```
        if type(value) != int and type(value) != float: #checking if value is neither an integer or a float
            unique_values.add(value) #if the value is neither it gets added to our set
    print(unique_values)


check_for_strings(data, 'price')
```

```
{'Free'}
```

The string throwing off our value is 'Free' which I can represent with a 0 and I can do this using the replace method. First I need to get the number of values that are being represented as Free so that when I complete replacing them I can make sure that all of the values were properly replaced.

In [ ]:
```
free_books = data[data['price']== 'Free']
print(len(free_books))
```

```
338
```

In [ ]:
```
data['price'].replace('Free', 0, inplace=True)
free_books = data[data['price']== 'Free']
print(free_books)
```

```
Empty DataFrame
Columns: [name, author, narrator, time, releasedate, language, stars, price]
Index: []
```

In [ ]:
```
formatted_price = data['price'].map("{:.2f}".format)
print(formatted_price)
```

```
0        468.00
1        820.00
2        410.00
3        615.00
4        820.00
          ...
87484    596.00
87485    820.00
87486    938.00
87487    680.00
87488    569.00
Name: price, Length: 87489, dtype: object
```

The stars was identified as an object so I believe that it included strings, I can do a quick overview to see if I can the strings are obviously evident.

In [ ]:
```
print(data['stars'])
```

```
0           5 out of 5 stars34 ratings
1         4.5 out of 5 stars41 ratings
2         4.5 out of 5 stars38 ratings
3         4.5 out of 5 stars12 ratings
4        4.5 out of 5 stars181 ratings
                    ...
87484                Not rated yet
87485                Not rated yet
87486                Not rated yet
87487                Not rated yet
87488                Not rated yet
Name: stars, Length: 87489, dtype: object
```

The strings were fairly evident, what I need is the first number since they are all based on a rating scale of 0 - 5. They are formatted two separate ways

1. 4.5 out of 5 stars181 ratings
2. Not rated yet

For option 1 I only want the first number so using the extract method i'll create a regex pattern to identify the number needed and convert it to a float. For option 2 I'll convert the the not rated yet to be np.nan.

In [ ]:
```
data['stars'].replace('Not rated yet', np.nan, inplace=True)
```

In [ ]:
```
data['stars'] = data['stars'].astype(str).str.extract('(\d+\.?\d*)').astype(float)
```

now that we've made the conversion, and we understand that there are missing values lets see how many missing values are present in the dataset.

In [ ]:
```
print(data['stars'].isna().sum())
```

```
72417
```

next up is language, I expected it to be a object since its filled with strings but checking to see if it is mixed with other data types is good practice.

In [ ]:
```
def check_string(df, column):
  for value in df[column]:
    if type(value) == str:
      continue
    elif type(value) == int or type(value) == float:
      print(value)

check_string(data, 'language')
```

no return so were good to continue with the remaining columns. Next up is release data.

In [ ]:
```
print(data['releasedate'])
```

```
0        2008-04-08 00:00:00
1        2018-01-05 00:00:00
2        2020-06-11 00:00:00
3        2021-05-10 00:00:00
4                  13-01-10
               ...
87484    2017-09-03 00:00:00
87485                21-02-17
87486                30-12-16
87487                23-02-11
87488    2017-07-03 00:00:00
Name: releasedate, Length: 87489, dtype: object
```

The date format is a bit all over the place, we could correct this in excel but since im working in Python i'll be doing it in Python. first we need to convert it to a string.

In [ ]:
```
data['releasedate'] = data['releasedate'].astype(str)
```

Following that, if we use pd.to_datetime it allows pandas to recognize and standardize the format the dates in our dataset is being represented as. I use coerce to handle errors like we would with try, except. So if python doesn't recognize the dates it gets formatted as NAT(not a time the data equivalent to nan not actual number)

In [ ]:
```
data['releasedate'] = pd.to_datetime(data['releasedate'], errors='coerce')
print(data['releasedate'])
```

```
0       2008-04-08
1       2018-01-05
2       2020-06-11
3       2021-05-10
4              NaT
           ...
87484   2017-09-03
87485          NaT
87486          NaT
87487          NaT
87488   2017-07-03
Name: releasedate, Length: 87489, dtype: datetime64[ns]
```

The author column is displaying in an awkward way as well the format seems to be camel case and it includes the WrittenBy: we can go without that. Using string replace and not the traditional replace because string replace looks inside the cell and makes adjustments accordingly replace just verifies that the cell meets the initial conditions and rewrites the full cell.

In [ ]:
```
data['author'] = data['author'].str.replace('Writtenby:', '')
print(data['author'])
```

```
0        GeronimoStilton
1            RickRiordan
2             JeffKinney
3            RickRiordan
4            RickRiordan
                ...
87484       ChrisStewart
87485      StephenO'Shea
87486          MarkTwain
87487     LaurenceSterne
87488      MarkKurlansky
Name: author, Length: 87489, dtype: object
```

Got rid of the Written by portion but the first name and the last name are still positioned in camel case. Using the re libary for text manipulation, and using the apply function to apply the function throughout our sheet.

In [ ]:
```
def add_spaces(name):
    return re.sub(r'(?<=[a-z])(?=[A-Z])', ' ', name)
data['author'] = data['author'].apply(add_spaces)

print(data['author'])
```

```
0        Geronimo Stilton
1            Rick Riordan
2             Jeff Kinney
3            Rick Riordan
4            Rick Riordan
                ...
87484       Chris Stewart
87485      Stephen O'Shea
87486          Mark Twain
87487     Laurence Sterne
87488      Mark Kurlansky
Name: author, Length: 87489, dtype: object
```

Same issue with the narrator column, each row is listed as Narratedby: ill start by removing the Narratedby: with the str.replace() method.

In [ ]:
```
data['narrator'] = data['narrator'].str.replace('Narratedby:', '')
print(data['narrator'])
```

```
0            BillLobely
1         RobbieDaymond
2            DanRussell
3        SoneelaNankani
4        JesseBernstein
              ...
87484      ChrisStewart
87485        RobertFass
87486         FloGibson
87487       AntonLesser
87488       FleetCooper
Name: narrator, Length: 87489, dtype: object
```

With the Narratedby: removed we can use the same function we created earlier to remove the camel case.

In [ ]:
```
def add_spaces(name):
    return re.sub(r'(?<=[a-z])(?=[A-Z])', ' ', name)
data['narrator'] = data['narrator'].apply(add_spaces)

print(data['narrator'])
```

```
0            Bill Lobely
1         Robbie Daymond
2            Dan Russell
3        Soneela Nankani
4        Jesse Bernstein
              ...
87484      Chris Stewart
87485        Robert Fass
87486         Flo Gibson
87487       Anton Lesser
87488       Fleet Cooper
Name: narrator, Length: 87489, dtype: object
```

Next is the name column, from an initial overview of the spreadsheet I noticed that there was a plethora of unique symbols that were obstructing the titles like the one below.

In [ ]:
```
print(data['name'].loc[9])
```

```
The Tyrantâ€™s Tomb
```

By running a for loop to see what unique characters are in the column name we can work to begin removing them. Starting with an empty set to hold the values since they dont have duplicates.

In [ ]:
```
unique_symbols = set()
```

Follow that with a for loop to iterate through the name list

In [ ]:
```
for text in data['name']:
    if isinstance(text, str): #if the text is a string
        unique_symbols.update(set(text))  # add the unique symbols from this text to our earlier created set

print(unique_symbols)
```

```
{'¦', 'Â', '§', 'l', 't', 'b', '…', ':', '&', 'Y', 'Ã', 'è', '¶', ']', 'W', '¢', '|', 'g', 'H', '©', '£', '¸', '
é', '\x8f', '2', 'Ð', 'Ø', '^', '¹', 'Ä', 'V', '¾', '¿', 'º', '\xad', 'â', '¨', '¡', '²', '‡', 'j', ',', 'm', '\
x90', '"', 'R', 'D', '\x8d', 'G', '-', 'S', '›', 'C', '0', '»', 'p', '°', '†', 'a', ' ', '”', 'µ', 'f', 'å', 'c'
, 'q', 'L', 'æ', '3', '‹', 'Ÿ', '=', 'Ì', 'u', '!', 'A', '5', 'M', "'", 's', ')', '(', '•', 'N', '/', '‘', 'Ñ',
'ª', '4', 'U', '·', 'ƒ', 'ä', '#', 'ž', '½', 'e', '8', '’', '@', 'È', '¯', '¤', '*', '.', 'Q', '+', 'Ž', 'v', 'x
', '€', '7', '$', '[', '~', 'w', 'J', 'y', '³', '™', '6', 'O', '´', 'Î', '_', 'h', '®', 'k', 'Å', 'X', 'Ù', 'z',
'%', 'n', 'ç', 'K', 'r', 'Œ', 'Ï', 'i', 'd', 'Z', '¥', '‰', '\x81', '?', '¼', '9', '„', 'I', 'o', '«', 'Š', '˜',
'š', '–', '±', 'P', 'T', '\x9d', 'B', 'à', '“', 'ï', ',', '1', '¬', 'F', '\xa0', ';', 'E', 'œ', '—', 'ã'}
```

One issue so far, you'll notice that the unique symbols set also includes traditional letters, letters that we'd like to keep. To make sure that they dont get caught in the removal process we can modify the unique symbols set with a for loop.

In [ ]:
```
unique_symbols = {char for char in unique_symbols if not char.isalnum() and not char.isspace()}
print(unique_symbols)
```

```
{'¦', '_', '·', '\x8d', '§', '#', '®', '…', '%', ':', '&', '-', '’', '¶', '›', '@', '»', '¯', '¤', ']', '°', '†'
, '*', '¥', '‰', '¢', '|', '”', '.', '\x81', '?', '©', '+', '£', '¸', '\x8f', '„', '˜', '«', '‹', '—', '±', '=',
'€', '!', ')', '(', "'", '\x9d', '•', '$', '[', '~', '¿', '“', '/', ',', '‘', '¬', '\xad', '™', '¨', '¡', '‡', '
;', '\x90', ',', '–', '"', '´'}
```

This removed all the alphanumeric symbols from our name column and left only the data that caused the original issues. We can create a function that takes the unique values left in the set and remove them from the column names.

In [ ]:
```
def replace_symbols(text):
    if isinstance(text, str):
        for symbol in unique_symbols:
            text = text.replace(symbol, '')
    return text

data['name'] = data['name'].apply(replace_symbols)
```

Last step checking to make sure we dropped the symbols throwing things off originally, using the same example from earlier.

In [ ]:
```
print(data['name'].loc[9])
```

```
The Tyrantâs Tomb
```

Language wasn't an issue but it does have some inconsistencies in regards to the way each of the entries are represented with some being lower case and others being upper case a small for loop should clear that.

In [ ]:
```
def capitalize(df, column):
    df[column] = df[column].apply(lambda x: str(x).title() if isinstance(x, str) else x)
capitalize(data, 'language')
```

In [ ]:
```
print(data.dtypes)
```

```
name                    object
author                  object
narrator                object
time                    object
releasedate     datetime64[ns]
language                object
stars                  float64
price                  float64
dtype: object
```

In [ ]:
```
# Capitalize column names
data.columns = [col.capitalize() for col in data.columns]

# Show the new column names
data.columns
```

Out[ ]:
```
Index(['Name', 'Author', 'Narrator', 'Time', 'Releasedate', 'Language',
       'Stars', 'Price'],
      dtype='object')
```

Now that thats been cleared up, we need to handle our missing values, lets start with getting a look at the total amount of missing values per column and make a decision from there.

In [ ]:
```
print(data.isna().sum())
```

```
Name               0
Author             0
Narrator           0
Time               0
Releasedate    52254
Language           0
Stars          72417
Price              0
dtype: int64
```

In [ ]:
```
print(data.shape)
```

```
(87489, 8)
```

We're missing a lot of values but luckily this an issue in only two columns, releasedate and stars, unfortunately these are the two columns that provide important insight. I have a few options here I can set the missing values to represent the mean but the issue is that in total there are have 87,489 values total and 72147 and 52254 missing values respectively. Setting them to be the mean would severely skew the results. I could opt to do more research using web scraping as a quick way to get the missing information but some sites don't allow that. For now I'll leave the values as missing and revisit this project in the future when I have access to more information.

Conclusion

I accomplished every goal that I set out to,

a. Removed the random symbols from the names column to make it more readable

b. Corrected the author and narrated columns removing the preface and separating first and last names

c. I didn't change the way time is represented because I personally prefer the way it is currently set up.

d. standardized release date using pythons built in to_datetime function.

e. removed the preface for stars keeping just the first number to represent rating.

f. Converted every entry in the price column to be floats so that it looks more like a financial input.