**Big Data Class Project – Part 2**

**Amazon Customer Review Data Analysis**

**Introduction:**

Big data is a term applied to data sets whose size or type is beyond the ability of traditional relational databases to capture, manage and process the data with low latency. Big data has one or more of the following characteristics: high volume, high velocity or high variety. Big data analytics is the often complex process of examining large and varied data sets, or big data, to uncover information -- such as hidden patterns, unknown correlations, market trends and customer preferences -- that can help organizations make informed business decisions.

**Problem description:**

In this project we are practicing and demonstrating out Big Data (BD) and Analytics skills. I am using Amazon customer review data which has more than 160 Million (160,796,570) observations. Using EMR, HDFS, Spark and Tableau, I am handling the data and systematically extracting information from. I am performing exploratory analysis on the review dataset and I am focusing on features like reviews and rating for detailed analysis. I am using Latent Dirichlet Allocation model for processing reviews and finding hidden patterns.

**Data Set:**

Amazon reviews dataset: [*https://registry.opendata.aws/amazon-reviews/*](https://registry.opendata.aws/amazon-reviews/)

**Project Environment:**

AWS EMR - AWS Educate Class account.

Project tools:

1. AWS EMR

2. AWS HDFS

3. AWS Spark

4. Tableau

**Dataset Requirements:**

A) Use the following product categories:

- Digital\_Ebook\_Purchase

- Books

- Wireless

- PC

- Mobile\_Apps

- Video\_DVD

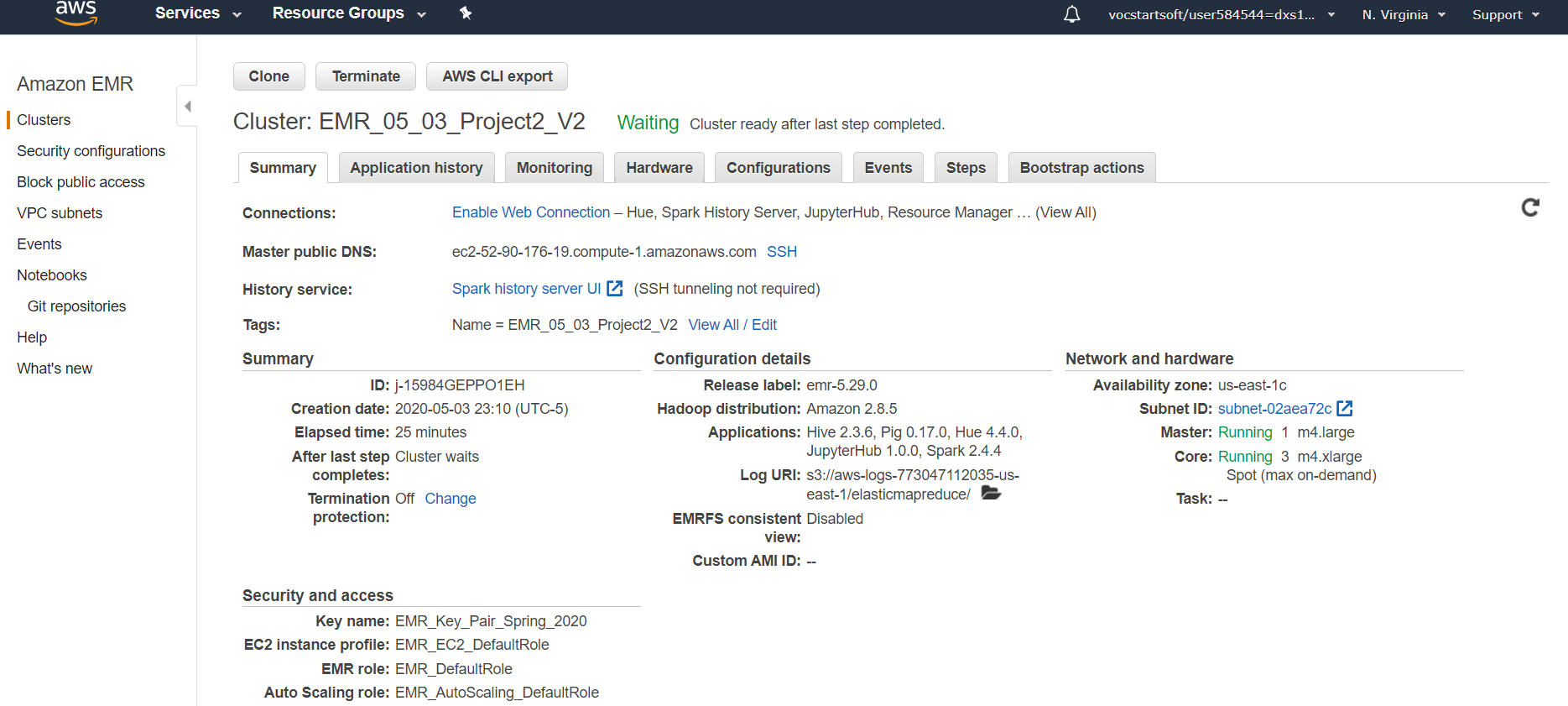
- Digital\_Video\_Download

B) Start your analysis from year 2005.

C) Exclude multiple reviews by the same users for the same product. In the case the same user has reviewed particular product more than once, exclude all reviews following the ﬁrst review. First review should remain as part of the analysis

**Step 1: AWS EMR:**

Provisioned EMR with below details:



**Step 2:-**

**A) Created directory for below each product category in HDFS:**

* Digital\_Ebook\_Purchase
* Books
* Wireless
* PC
* Mobile\_Apps
* Video\_DVD
* Digital\_Video\_Download

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Digital\_Ebook\_Purchase/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Books/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Wireless/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=PC/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Mobile\_Apps/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Video\_DVD/

hdfs dfs -mkdir -p /hive/amazon-reviews-pds/parquet/product\_category=Digital\_Video\_Download/

**B) Copying dataset from S3 in HDFS for each of the product category mentioned:**

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Wireless/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Wireless/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Digital\_Ebook\_Purchase/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Digital\_Ebook\_Purchase/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Books/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Books/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=PC/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=PC/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Mobile\_Apps/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Mobile\_Apps/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Video\_DVD/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Video\_DVD/

s3-dist-cp --src=s3://amazon-reviews-pds/parquet/product\_category=Digital\_Video\_Download/ --dest=hdfs:///hive/amazon-reviews-pds/parquet/product\_category=Digital\_Video\_Download/

**C) Importing dataset from HDFS into spark dataframe:**

**Code:-**

# Load Data Set

df = spark.read\

.option("header", "true")\

.option("inferSchema", "true")\

.option("basePath", "hdfs:///hive/amazon-reviews-pds/parquet/")\

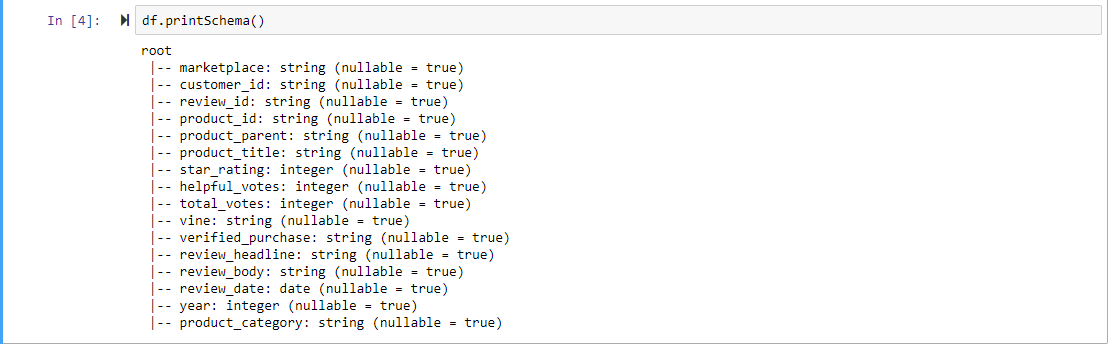
.parquet("hdfs:///hive/amazon-reviews-pds/parquet/\*")



**D) Checking the schema of dataframe:**

**Code:**

df.printSchema()



**E) Considering reviews after 2004 only:**

**Code:**

df1 = df.filter(F.col("year")>2004)



**F) Finding the unique rows based on the product category, product id and customer id:**

**Code:**

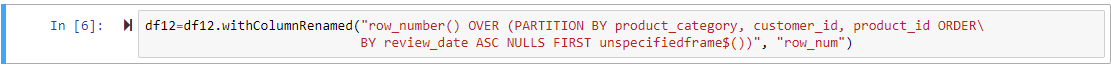
from pyspark.sql.window import Window

import pyspark.sql.functions as F

df12 = df1.select("\*",F.row\_number().over(Window.partitionBy("product\_category",'customer\_id', 'product\_id').orderBy(df['review\_date'])))



df12=df12.withColumnRenamed("row\_number() OVER (PARTITION BY product\_category, customer\_id, product\_id ORDER BY review\_date ASC NULLS FIRST unspecifiedframe$())", "row\_num")



# The count of rows which we will be removing

df12.where(F.col("row\_num")>1).count()



# Unique rows

df2=df12.where(F.col("row\_num")==1)



**Ans:** Df2 is the main filtered dataset on which I am performing my analysis.

**G) Number of reviews:**



**Ans**: The count of reviews on which I am performing my analysis is 65911069

**Step 3:**

**1. Explore the dataset and provide analysis by product-category and year:**

**1. Number of reviews:**

**Code:**

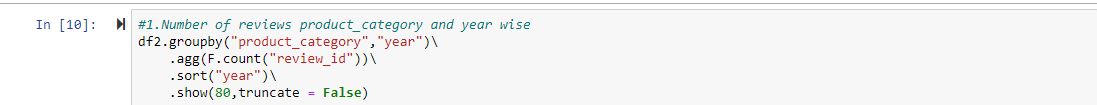
#1.Number of reviews product\_category and year wise

df2.groupby("product\_category","year")\

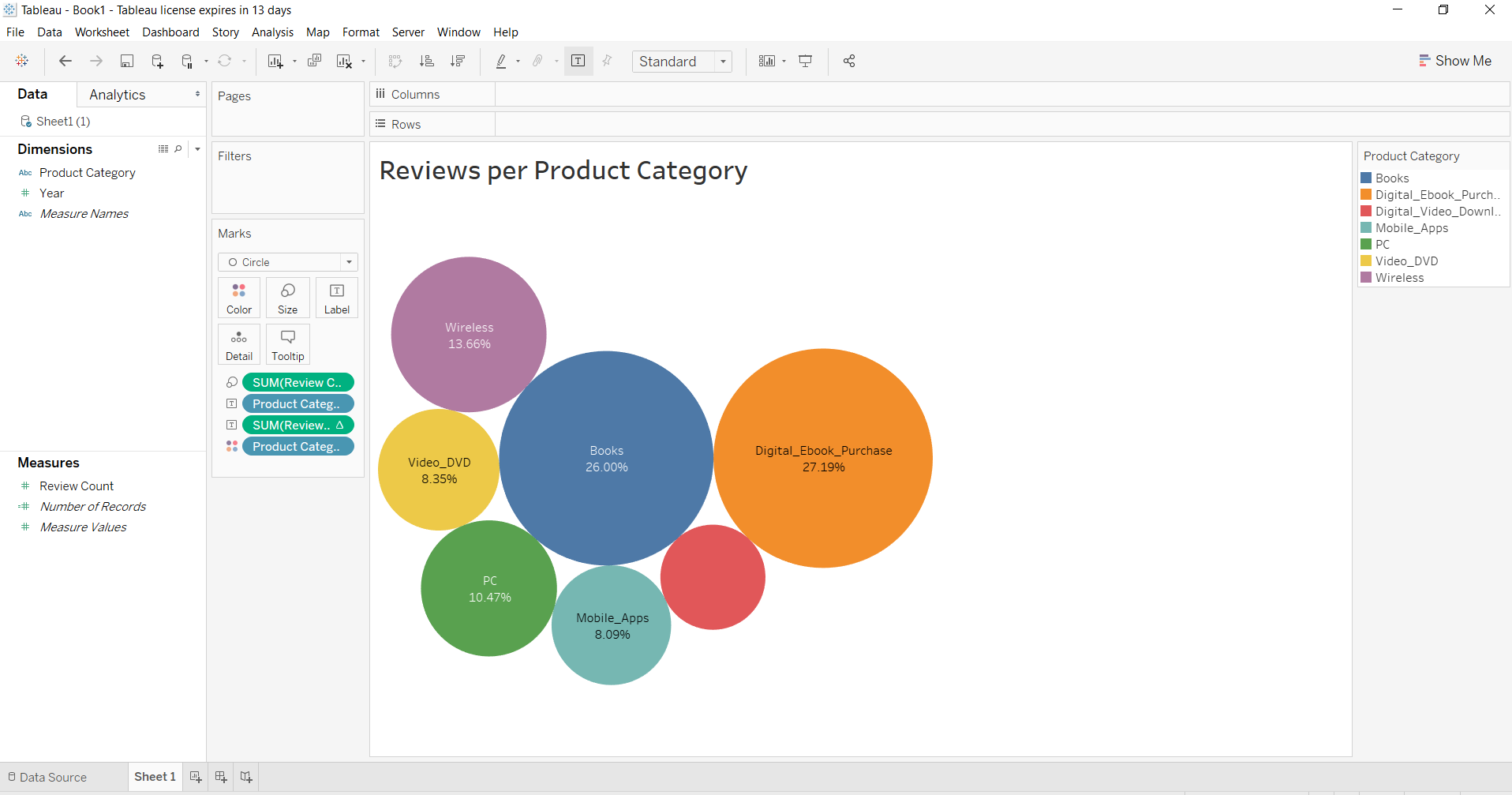
.agg(F.count("review\_id"))\

.sort("year")\

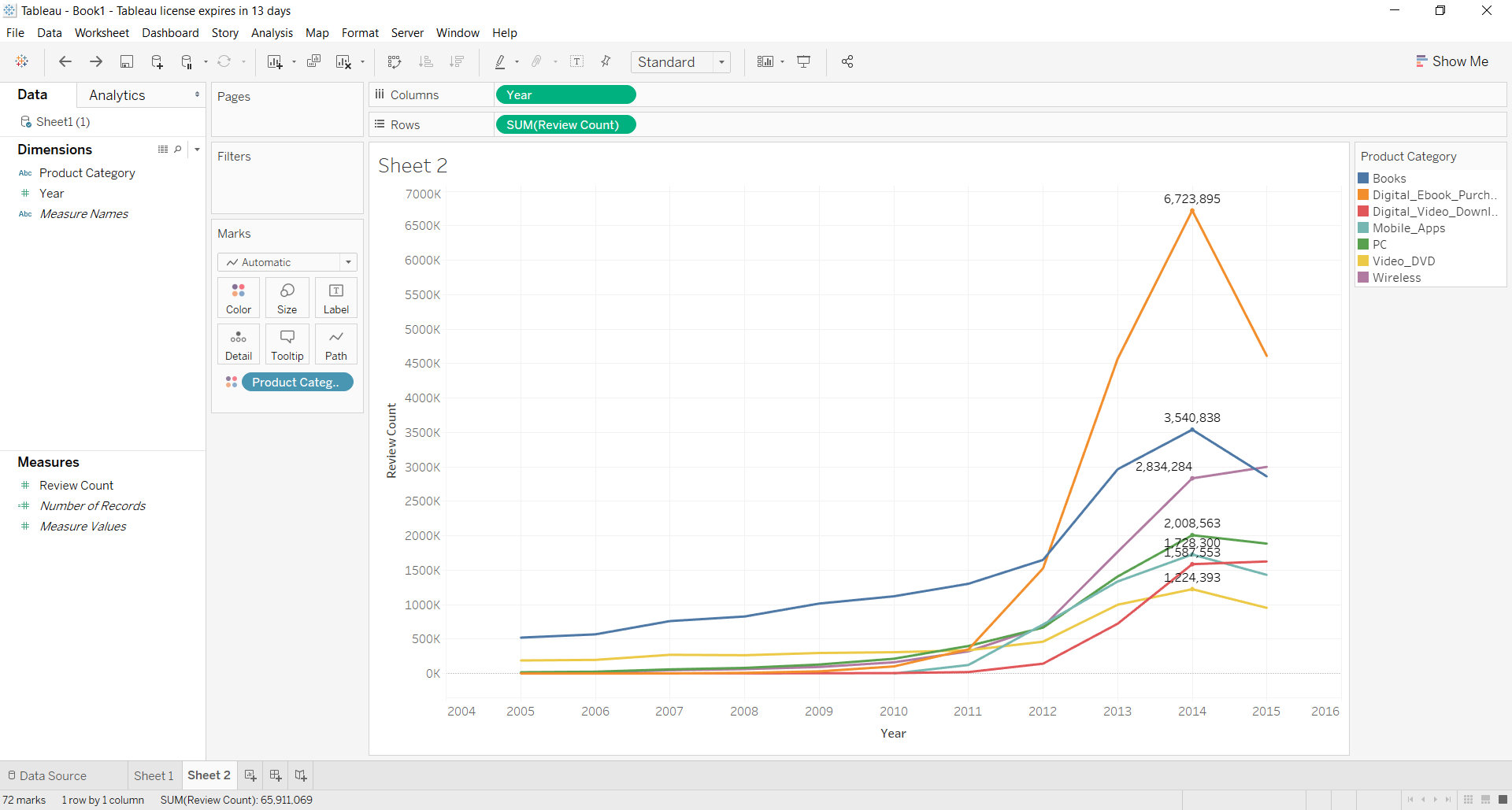
.show(80,truncate = False)

****

****

****

**Interpretation****:**We can see that Books and Digital\_Ebooks\_Purchase category constitutes more than 53% of the dataset.

****

**Interpretation:**We can see that Digital\_Ebooks\_Purchase observed maximum growth rate as well as the highest peak which indicate it as a booming category.

**2. Number of users:**

**Code:**

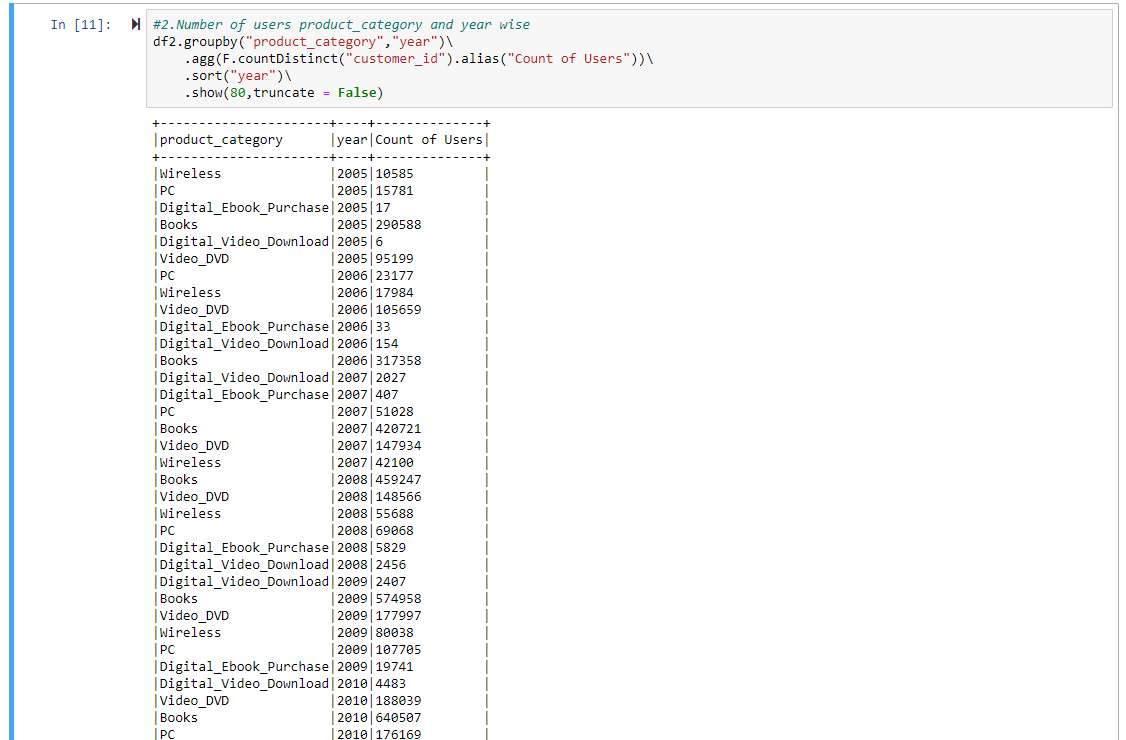
#2.Number of users product\_category and year wise

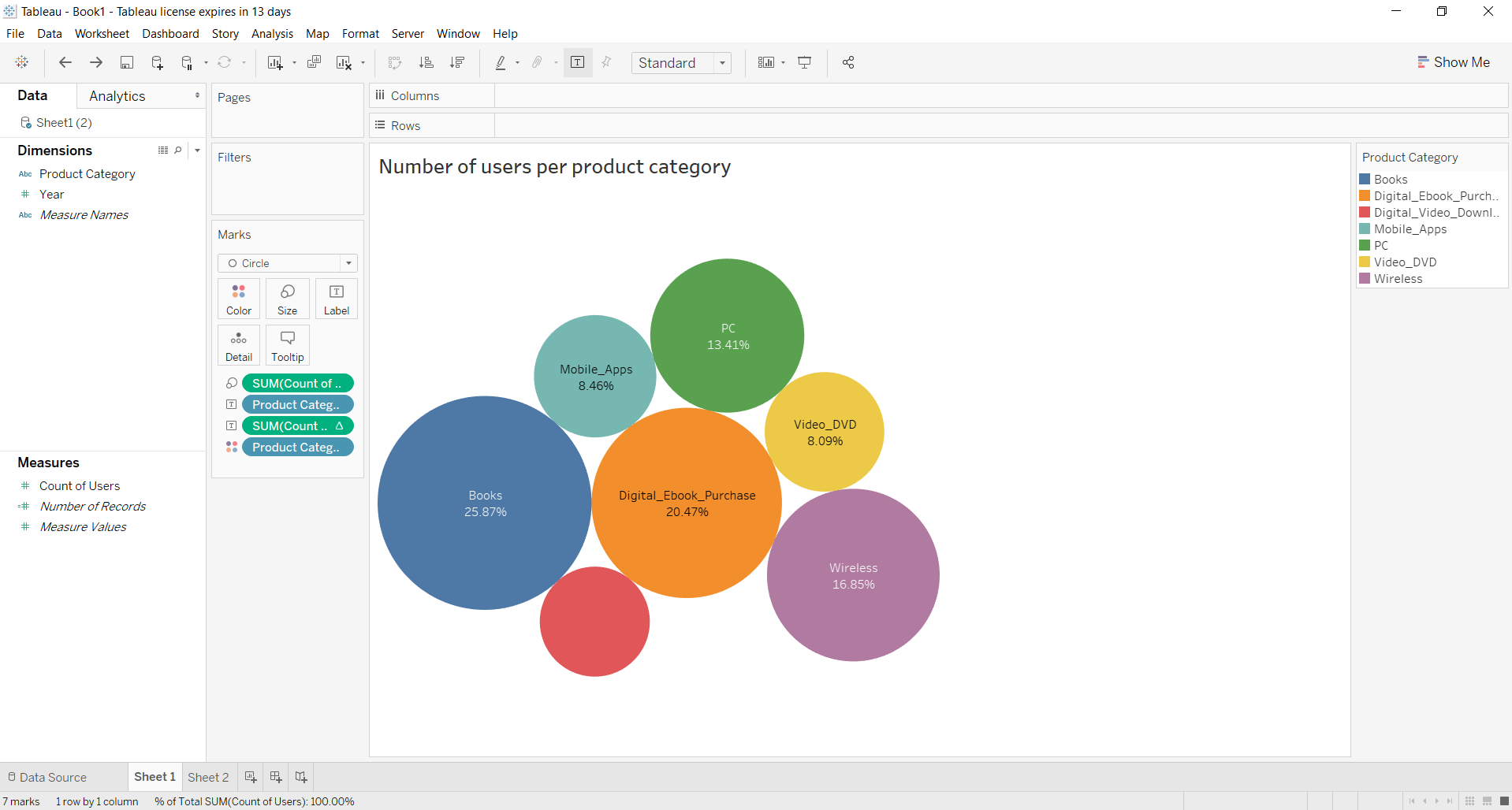
df2.groupby("product\_category","year")\

.agg(F.countDistinct("customer\_id").alias("Count of Users"))\

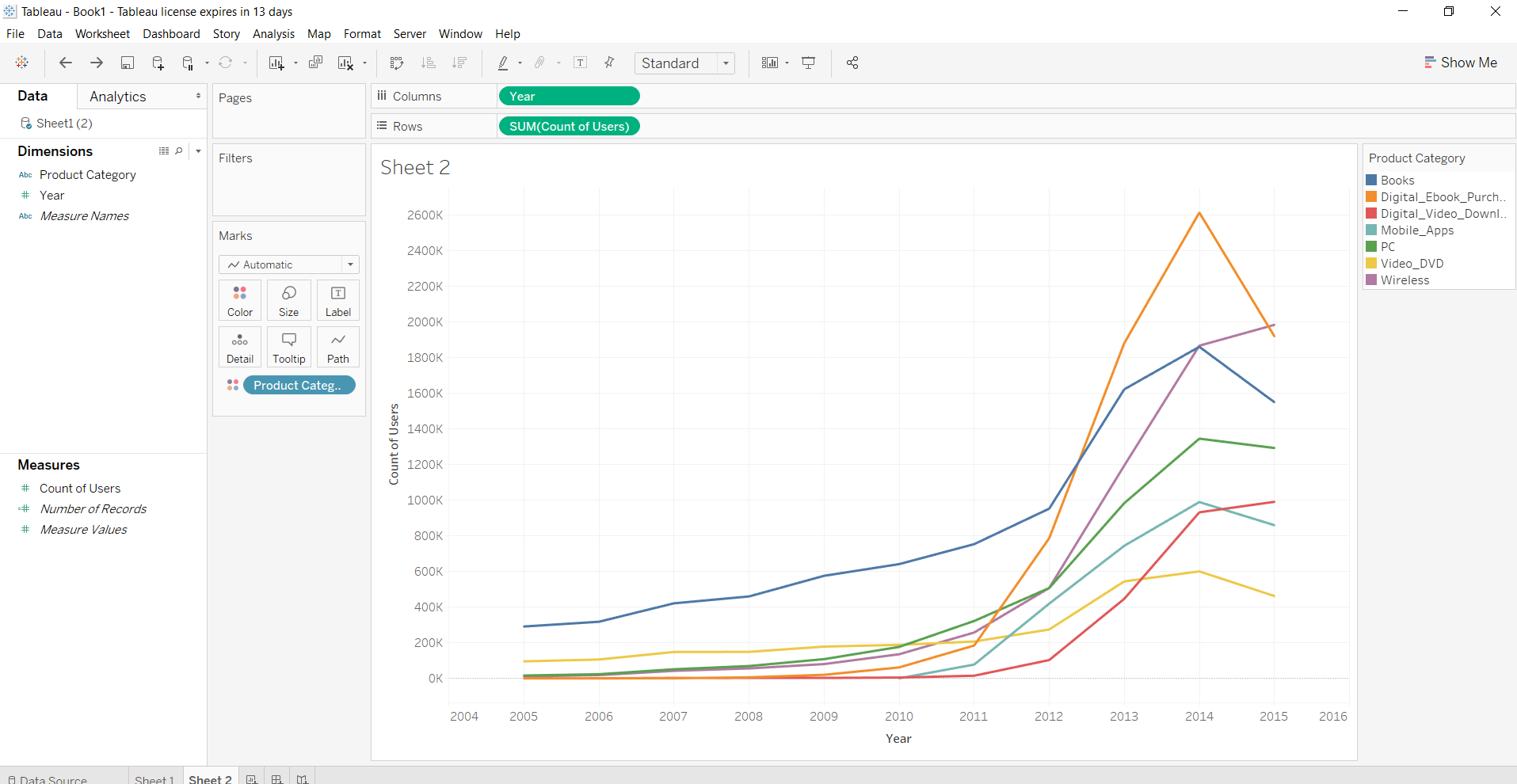
.sort("year")\

.show(80,truncate = False)

****

****

**Interpretation:** We can see from the above chart that customers buying books are 26% whereas for Ebooks its 20%. This shows there are more number of people who prefer buying books than Ebooks.

****

**Interpretation:** We can see that though all the categories are showing a growth in customers but top 3 are Books, Ebooks and Wireless categories. Digital\_Ebook\_Purchase has observed the maximum growth rate in customers.

**3. Average and Median review stars:**

**Code:**

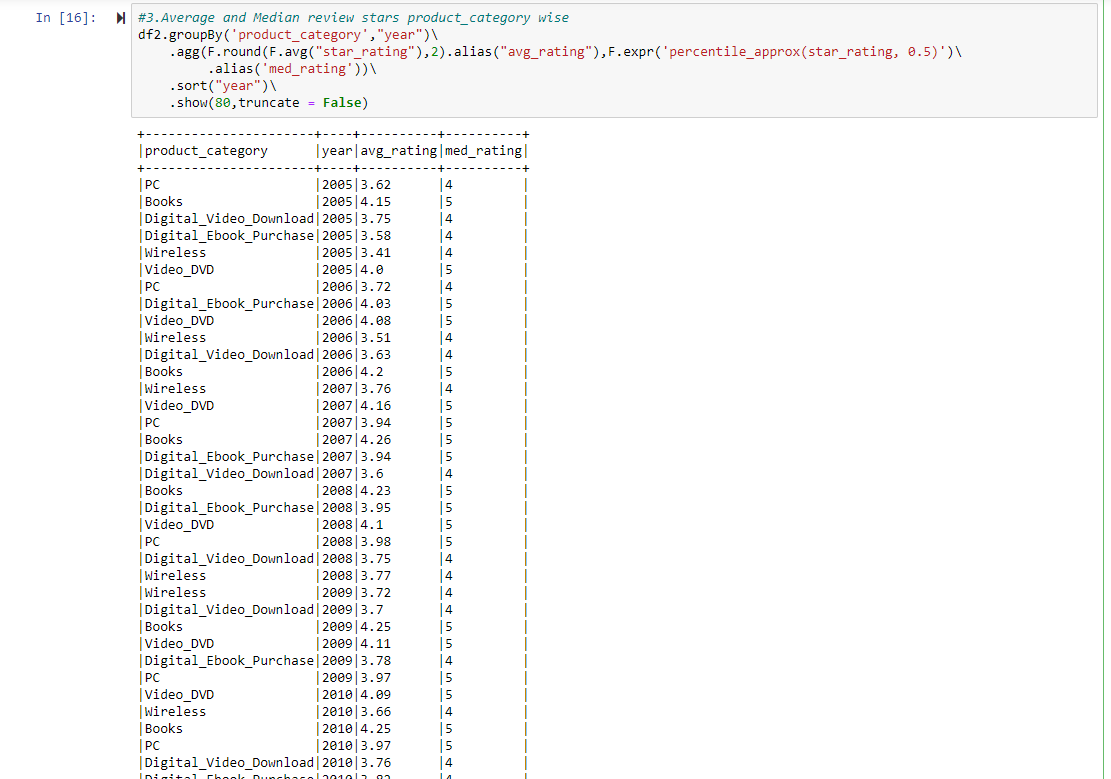
#3.Average and Median review stars product\_category wise

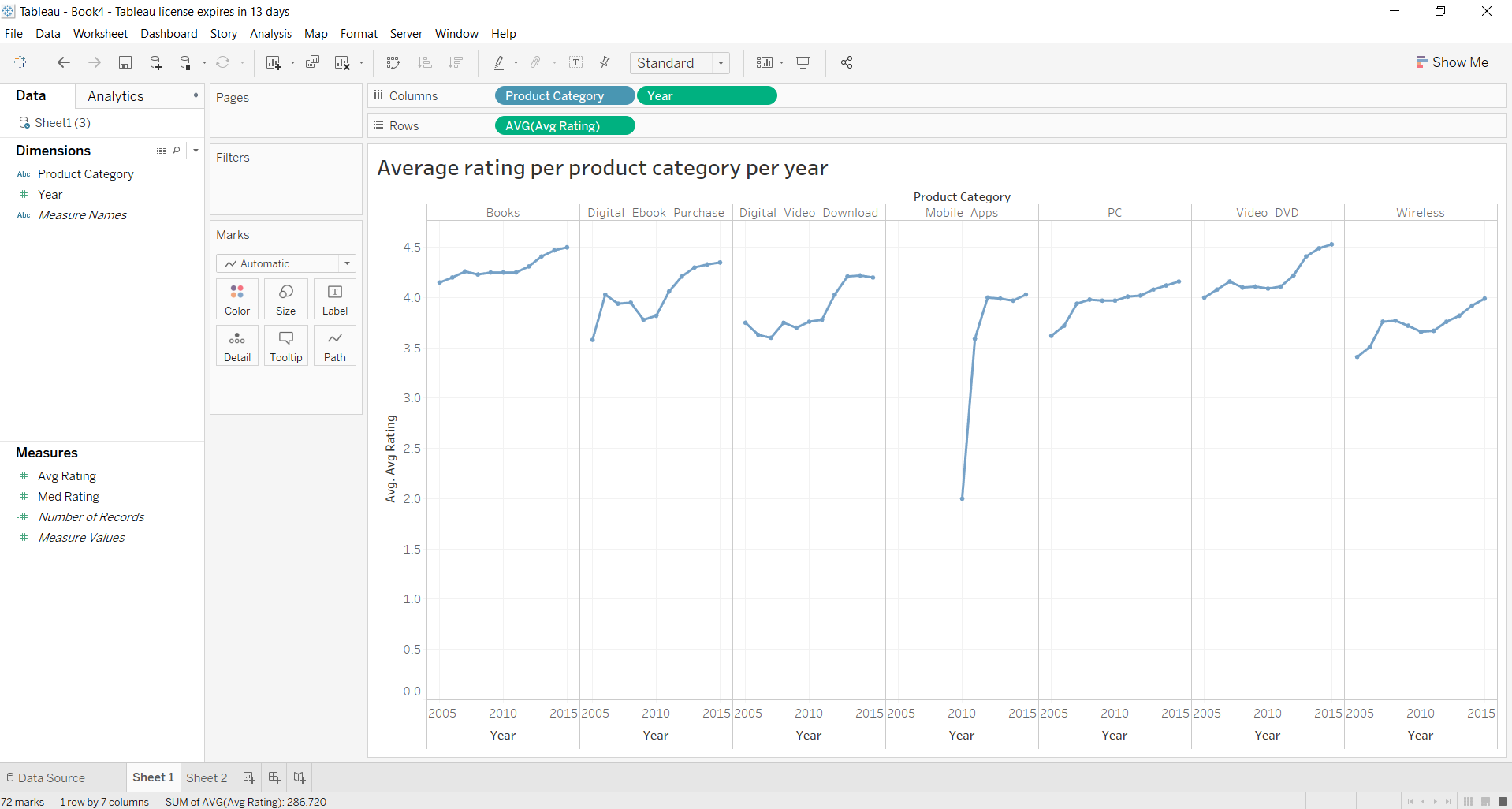
df2.groupBy('product\_category',"year")\

.agg(F.round(F.avg("star\_rating"),2).alias("avg\_rating"),F.expr('percentile\_approx(star\_rating, 0.5)').alias('med\_rating'))\

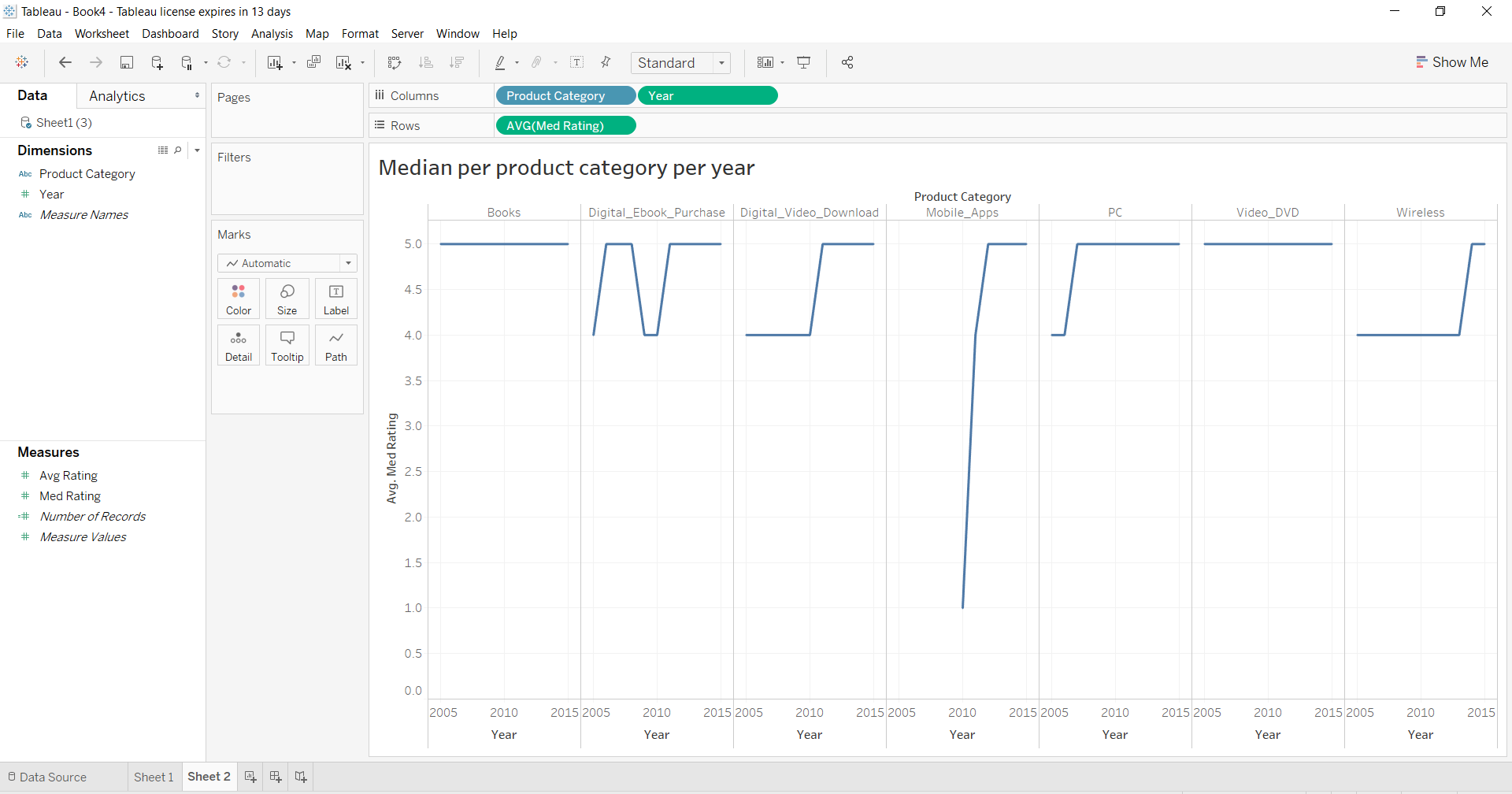
.sort("year")\

.show(80,truncate = False)





**Interpretation:** We can see from the above chart that average rating of all the product categories have increased over the years. The Mobile apps observed the maximum growth rate in average rating and proves that as the technologies improved the customers became more and more satisfied.



**Interpretation:** We can see from the above chart that median rating of all the product categories have been between 4 and 5 except Mobile Apps. The Mobile apps observed the maximum growth rate in median of rating and proves that as the technologies improved the customers became more and more satisfied and maximum customers started giving higher star rating.

**4. Percentiles of length of the review. Use the following percentiles: [0.1, 0.25, 0.5, 0.75, 0.9, 0.95]**

**Code:-**

a.#Below code performs percentile of length of the review over the entire dataset:

from pyspark.sql.functions import length

df3=df2.select("\*",length("review\_body"))

df4=df3.withColumnRenamed("length(review\_body)", "review\_len")



**Ans:** Seeing the percentile of length of reviews, we can say that only 5% of the reviews have length greater than 51,019 words.

b#Below code performs percentile of length of the review over the entire dataset per product category and year wise:

df4.groupby("product\_category","year").agg(F.round(F.expr('percentile\_approx(Review\_Len, 0.1)')).alias('%1'),\

F.round(F.expr('percentile\_approx(Review\_Len, 0.25)')).alias('%25'),\

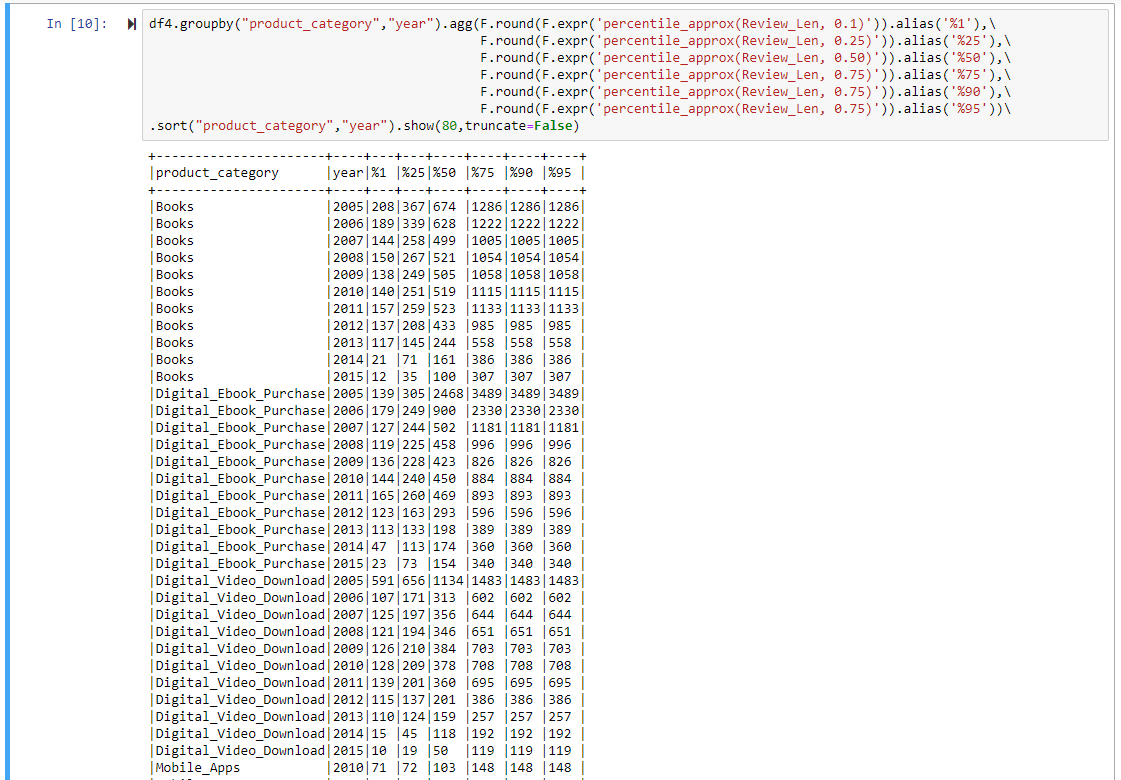
F.round(F.expr('percentile\_approx(Review\_Len, 0.50)')).alias('%50'),\

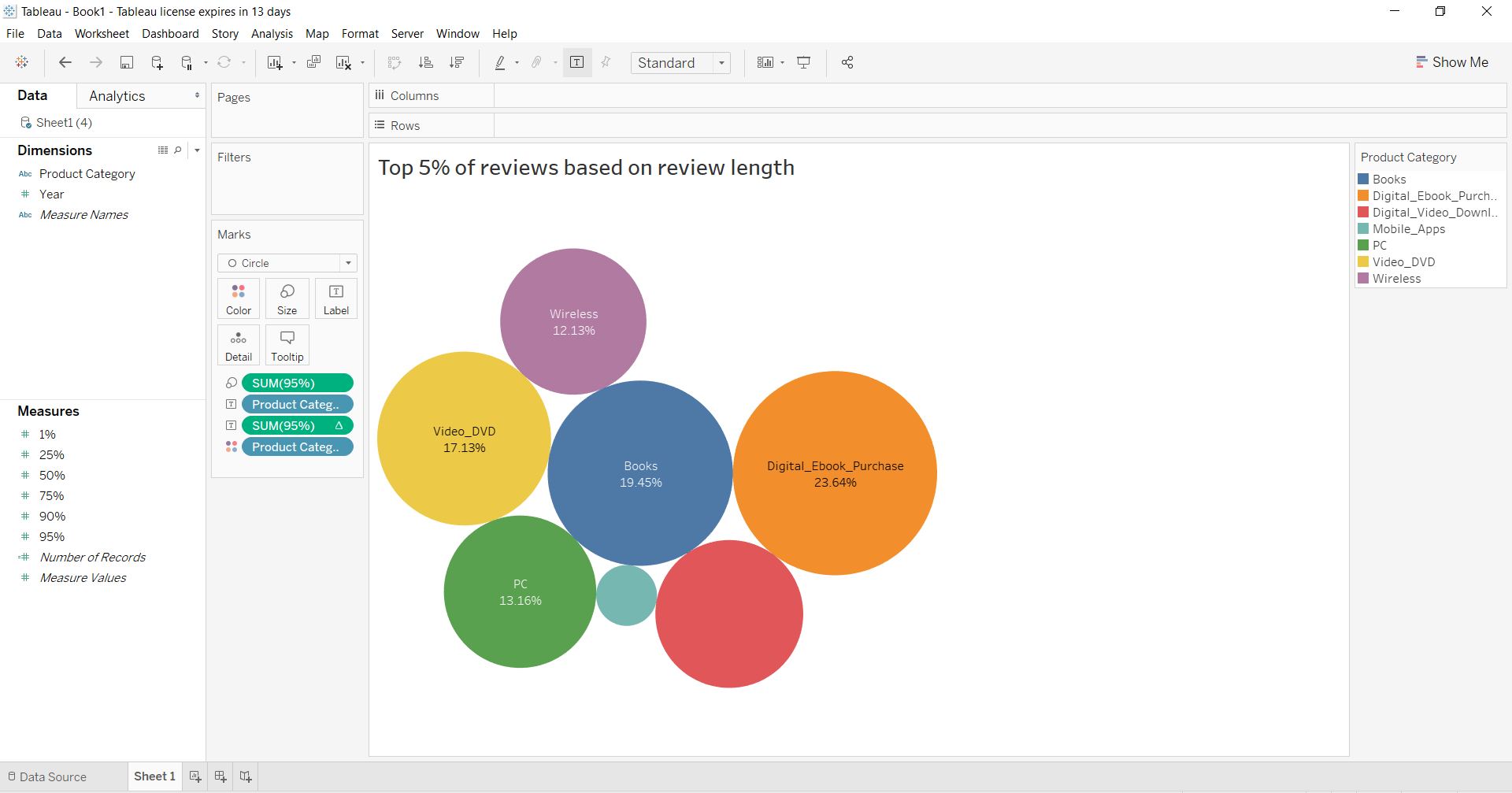
F.round(F.expr('percentile\_approx(Review\_Len, 0.75)')).alias('%75'),\

F.round(F.expr('percentile\_approx(Review\_Len, 0.75)')).alias('%90'),\

F.round(F.expr('percentile\_approx(Review\_Len, 0.75)')).alias('%95'))\

.sort("product\_category","year").show(80,truncate=False)





**Interpretation:** The above chart shows top 5% of the reviews based on reviews length. We can say that there is not significant difference in the size of the categories.

**5. Percentiles for number of reviews per product. For example, 10% of books got 5 or less reviews. Use the following percentiles: [0.1, 0.25, 0.5, 0.75, 0.9, 0.95]**

**Code:-**

#Below code performs 10,25,50,90 and 95 percentiles over product category and year for review count:

import pyspark.sql.functions as F

df\_Q5 = df2.groupBy("product\_category","year","product\_title").agg(F.count("review\_id").alias('review\_count'))

df\_Q5.groupby("product\_category","year").agg(F.round(F.expr('percentile\_approx(review\_count, 0.1)')).alias('%1'),\

F.round(F.expr('percentile\_approx(review\_count, 0.25)')).alias('%25'),\

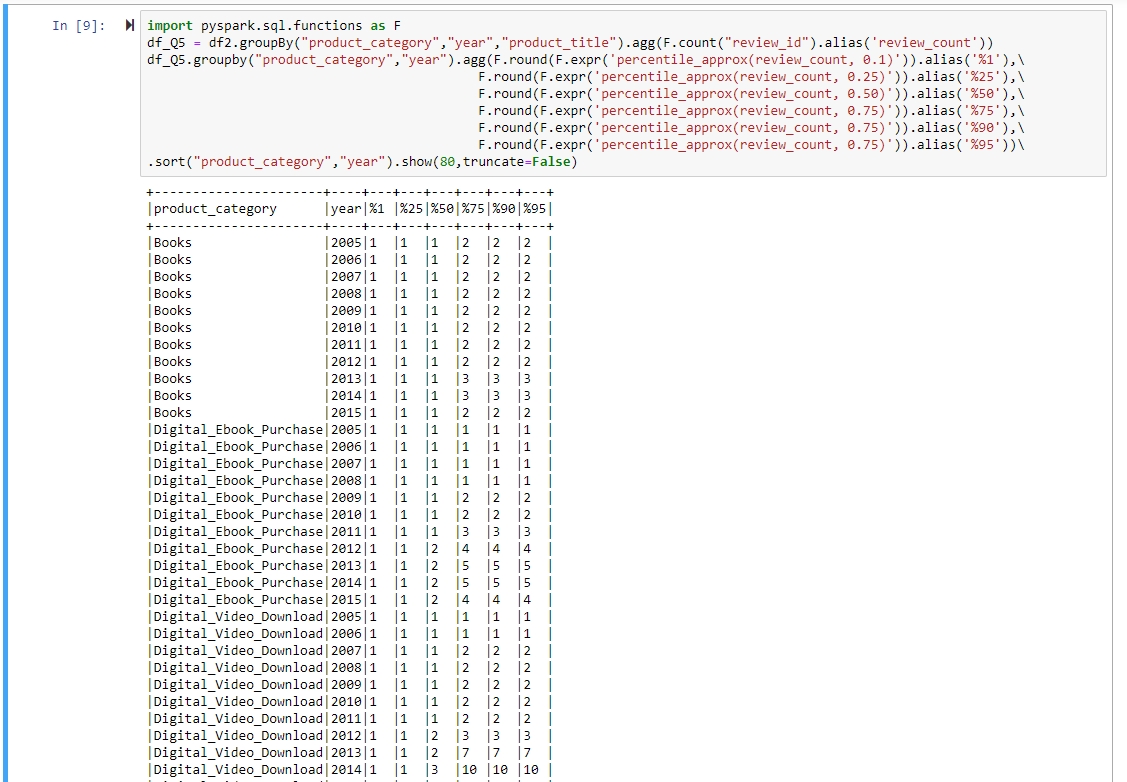
F.round(F.expr('percentile\_approx(review\_count, 0.50)')).alias('%50'),\

F.round(F.expr('percentile\_approx(review\_count, 0.75)')).alias('%75'),\

F.round(F.expr('percentile\_approx(review\_count, 0.75)')).alias('%90'),\

F.round(F.expr('percentile\_approx(review\_count, 0.75)')).alias('%95'))\

.sort("product\_category","year").show(80,truncate=False)



**6. Identify week number (each year has 52 weeks) for each year and product category with most positive reviews (4 and 5 star).**

**Code:-**

df4= df2.select("product\_category","year","review\_date",'star\_rating').withColumn('week',F.weekofyear(df2.review\_date))\

.where(F.column("star\_rating").isin([4,5]))

df6 = df4.groupby("product\_category","year","week")\

.agg(F.count("star\_rating").alias("rating\_count"))\

.sort("product\_category","year")

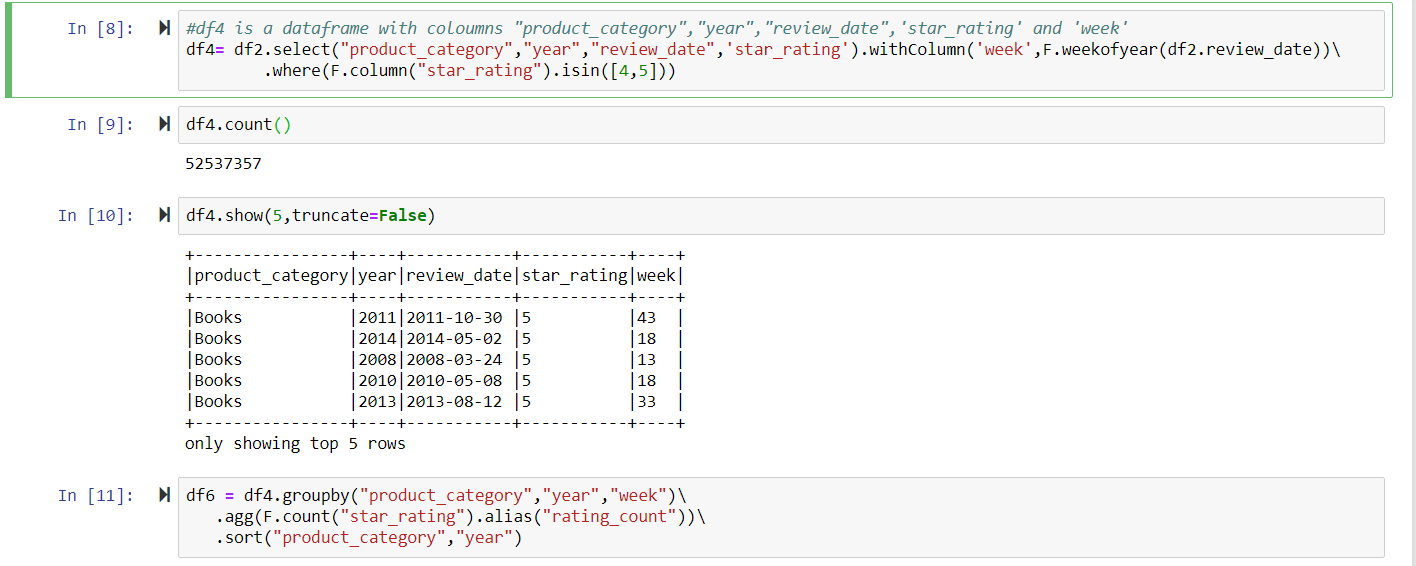
from pyspark.sql.window import Window

import pyspark.sql.functions as F

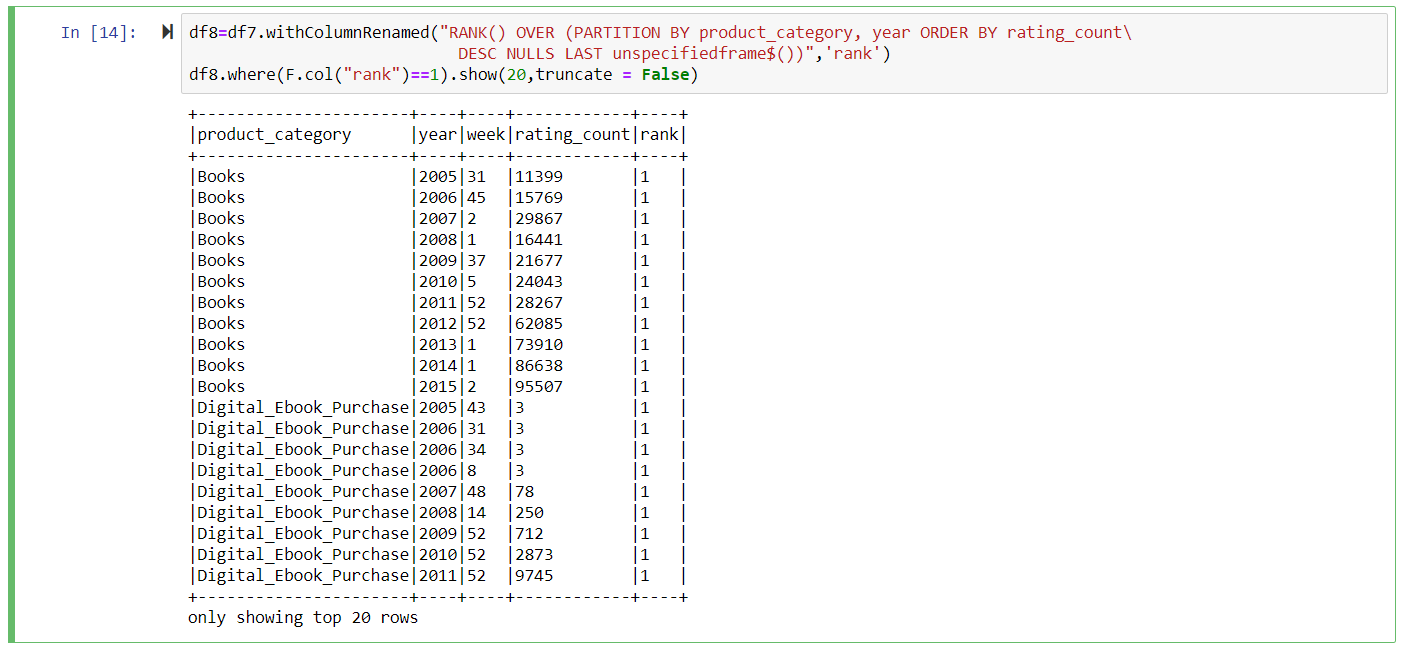
df7 = df6.select("\*",F.rank().over(Window.partitionBy("product\_category",'year').orderBy(F.col('rating\_count').desc())))

df8=df7.withColumnRenamed("RANK() OVER (PARTITION BY product\_category, year ORDER BY rating\_count DESC NULLS LAST unspecifiedframe$())",'rank')

df8.where(F.col("rank")==1).show(80,truncate = False)

****

****

****

**2. Provide detailed analysis of "Digital eBook Purchase" versus Books.**

**1. Using Spark Pivot functionality, produce DataFrame with following columns:**

**1. Year**

**2. Month**

**3. Total number of reviews for "Digital eBook Purchase" category**

**4. Total number of reviews for "Books" category**

**5. Average stars for reviews for "Digital eBook Purchase" category**

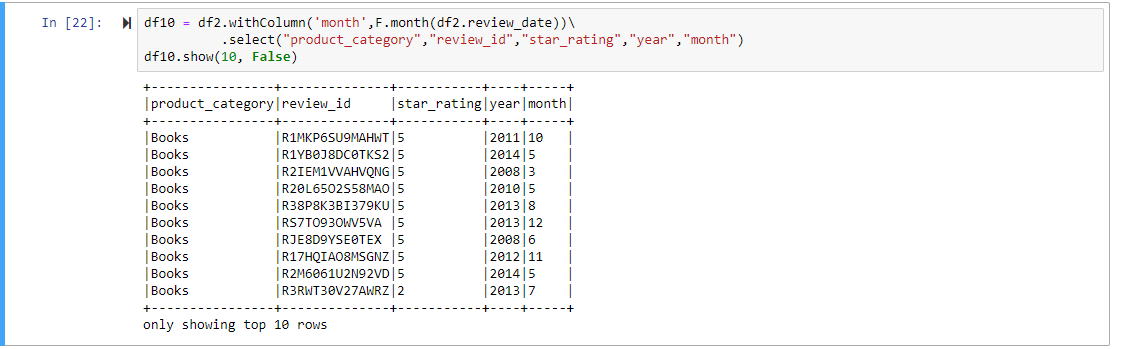
**6. Average stars for reviews for "Books" category**

**Code:-**

df10 = df2.withColumn('month',F.month(df2.review\_date))\

.select("product\_category","review\_id","star\_rating","year","month")

df10.show(10, False)

****

category\_to\_filter=["Digital\_Ebook\_Purchase","Books"]

Q21=df10.groupBy("Year","Month").pivot("product\_category",category\_to\_filter)\

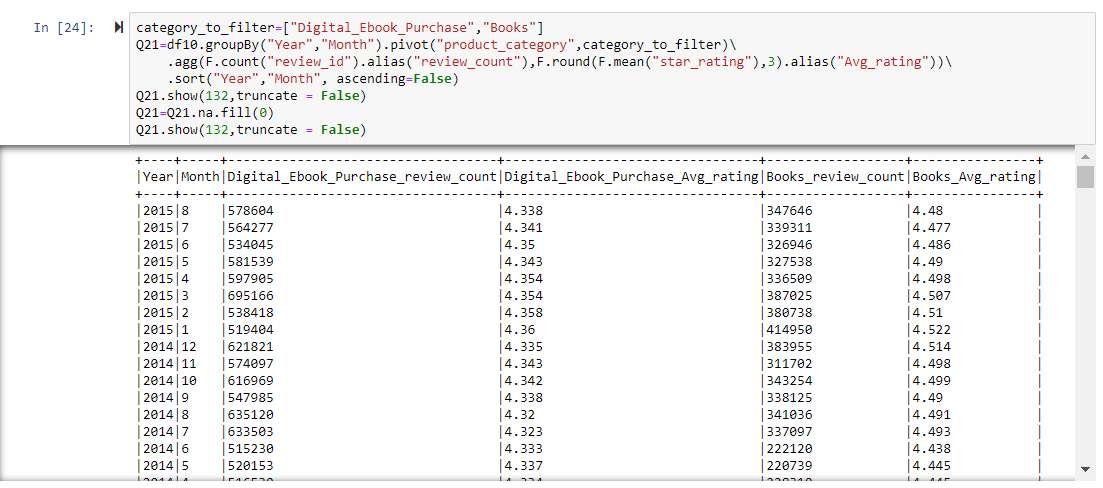
.agg(F.count("review\_id").alias("review\_count"),F.round(F.mean("star\_rating"),3).alias("Avg\_rating"))\

.sort("Year","Month", ascending=False)

Q21.show(132,truncate = False)

Q21=Q21.na.fill(0)

Q21.show(132,truncate = False)

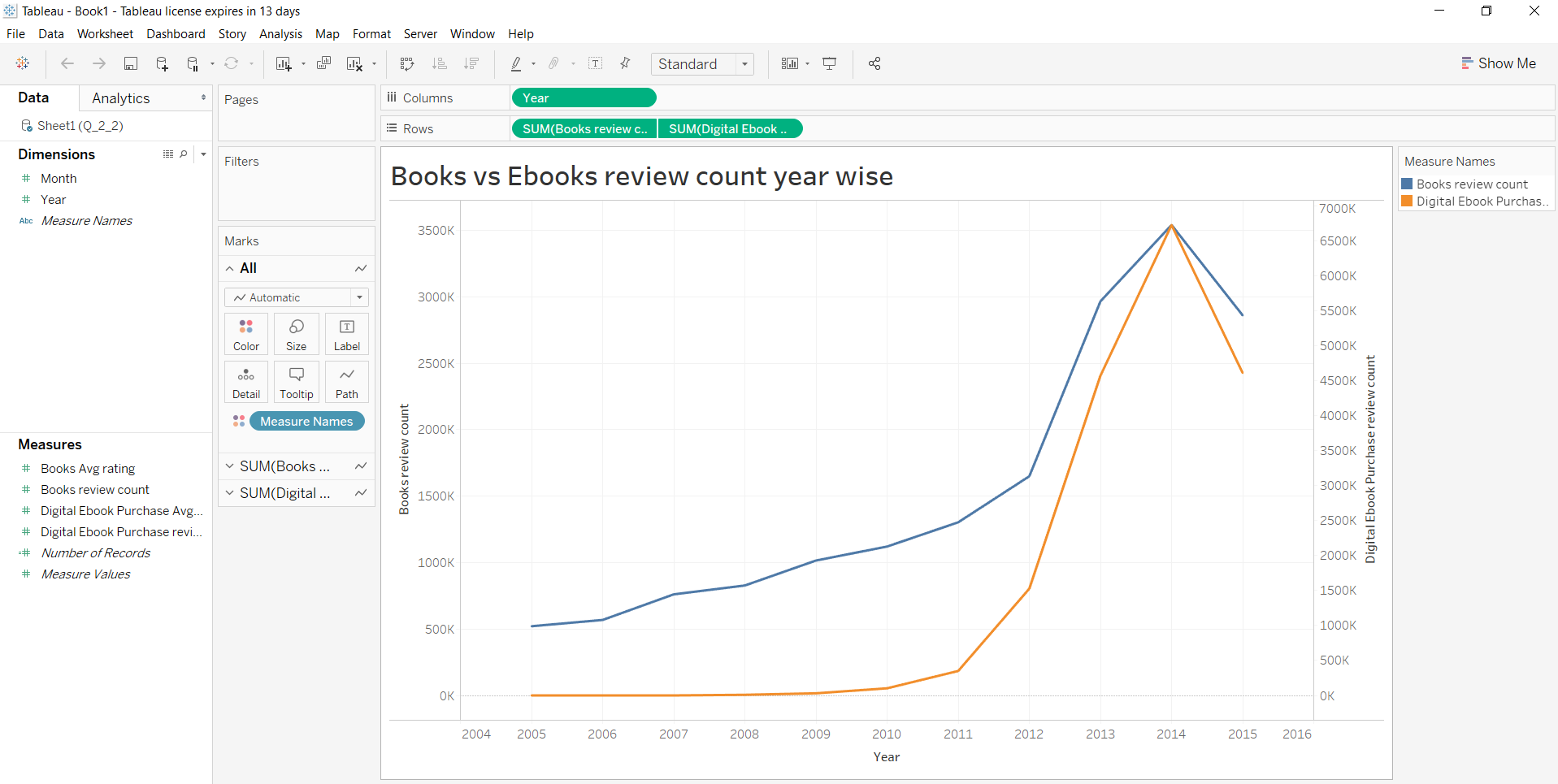
****

**Interpretation:** I have used sparks pivot functionality on Digital\_Ebook\_Purchase and Books product categories to show average star rating and average review count per month per year.

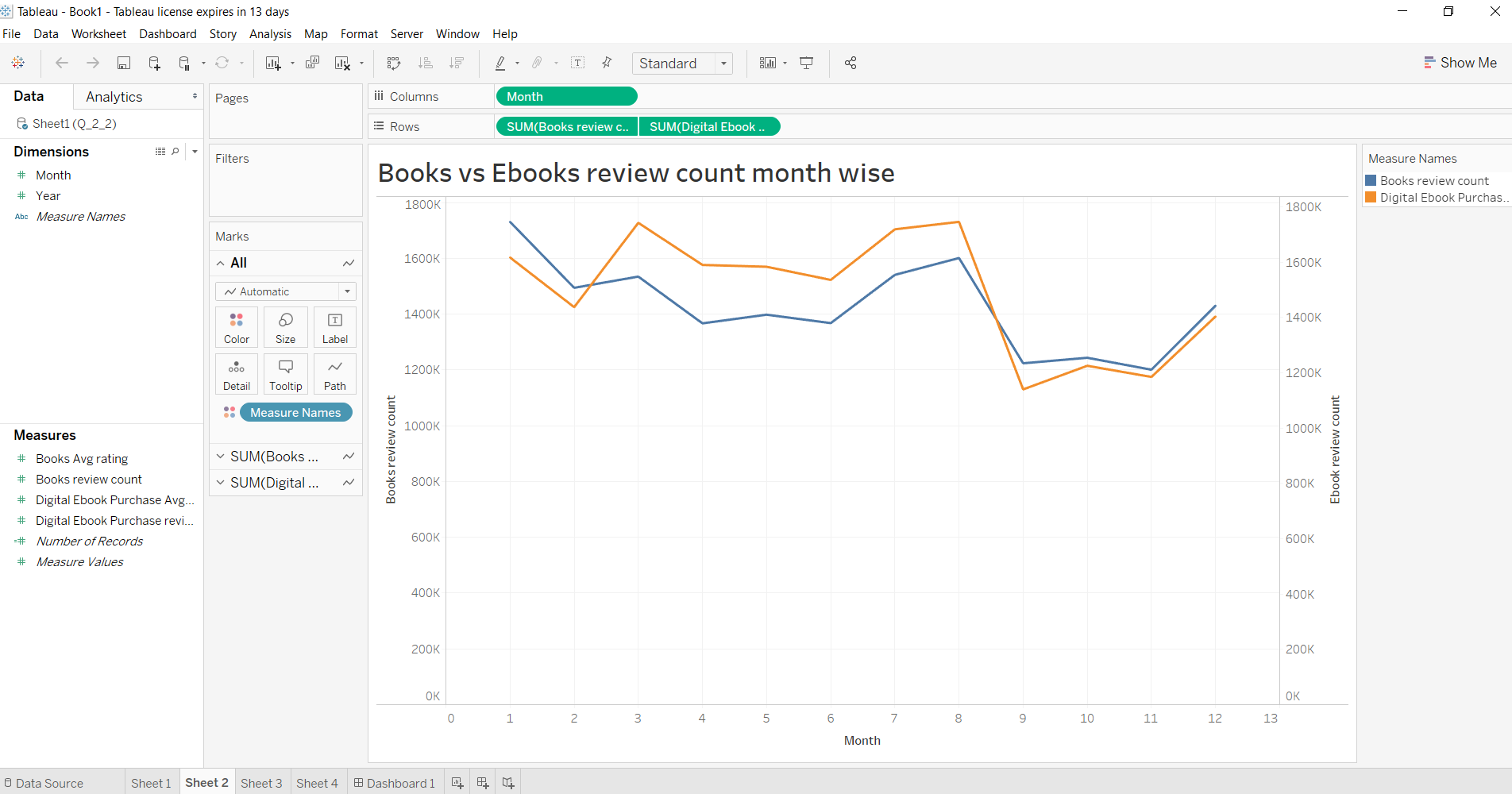
**2. Produce two graphs to demonstrate aggregations from #1:**

**1. Number of reviews**

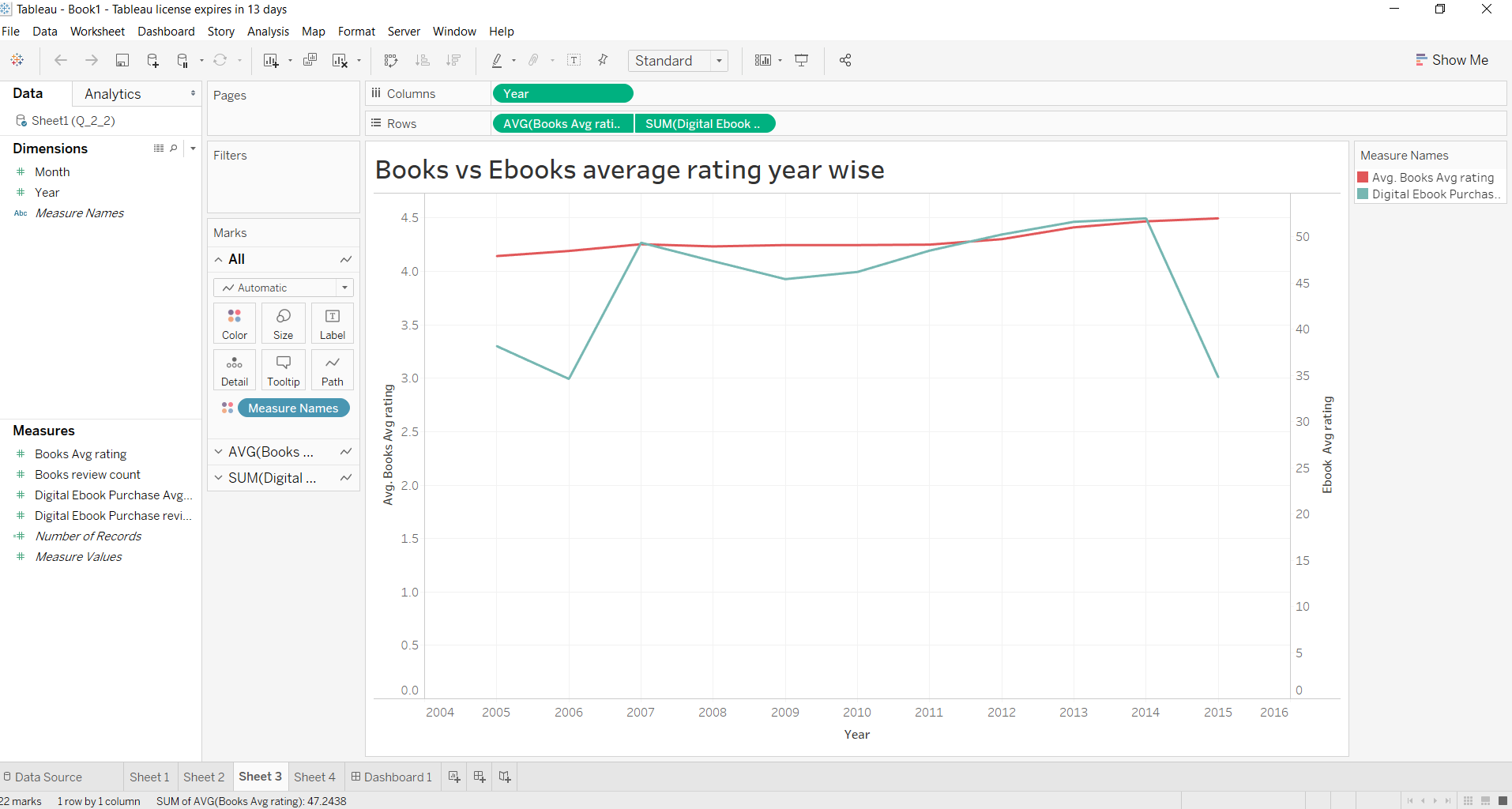
**2. Average stars**

****

**Interpretation:** The above graph between Books and Digital Books review count per year shows that there has been rapid growth in purchase of products in both categories. For Digital books we can see that between 2005 to 2010 there was almost no to very less purchase. In 2014 we can see that both the categories are sharing the same peak. Thus, we can say that there has been a higher growth rate in Digital books than Books category.

****

**Interpretation:** The above graph between Books and Digital Books review count per month shows that there has been a higher purchase from months between March and August.

****

**Interpretation:** The above graph between Books and Digital Books average rating per year shows that there for Books category the average rating is between 4 and 4.5, whereas for Digital books the average rating started from 3-3.5 during 2004 to 2006 and increased to 4 afterwards.

**3. Identify similar products (books) in both categories. Use "product\_title" to match products. To account for potential diﬀerences in naming of products, compare titles after stripping spaces and converting to lower case.**

**Code:-**

#Q2.3

#df12 will have df2 dataset with condition product\_category is "Digital\_Ebook\_Purchase" or "Books" and star\_rating is 4 or 5

category\_to\_filter=["Digital\_Ebook\_Purchase","Books"]

Rating=[4,5]

df12=df2.select("product\_category","product\_title","star\_rating")\

.filter((F.col("product\_category").isin(category\_to\_filter)) & (F.col("star\_rating").isin(Rating)))

#Q231a is a dataframe with product\_category Digital\_Ebook\_Purchase

Q231a= df12.groupby("product\_category", F.lower(F.trim(F.col("product\_title"))).alias("product\_title"))\

.agg(F.round(F.avg("star\_rating"),2).alias("Ebook\_Avg\_rating"))\

.filter(F.col("product\_category")=="Digital\_Ebook\_Purchase")

#Q231b is a dataframe with product\_category Books

Q231b= df12.groupby("product\_category", F.lower(F.trim(F.col("product\_title"))).alias("product\_title"))\

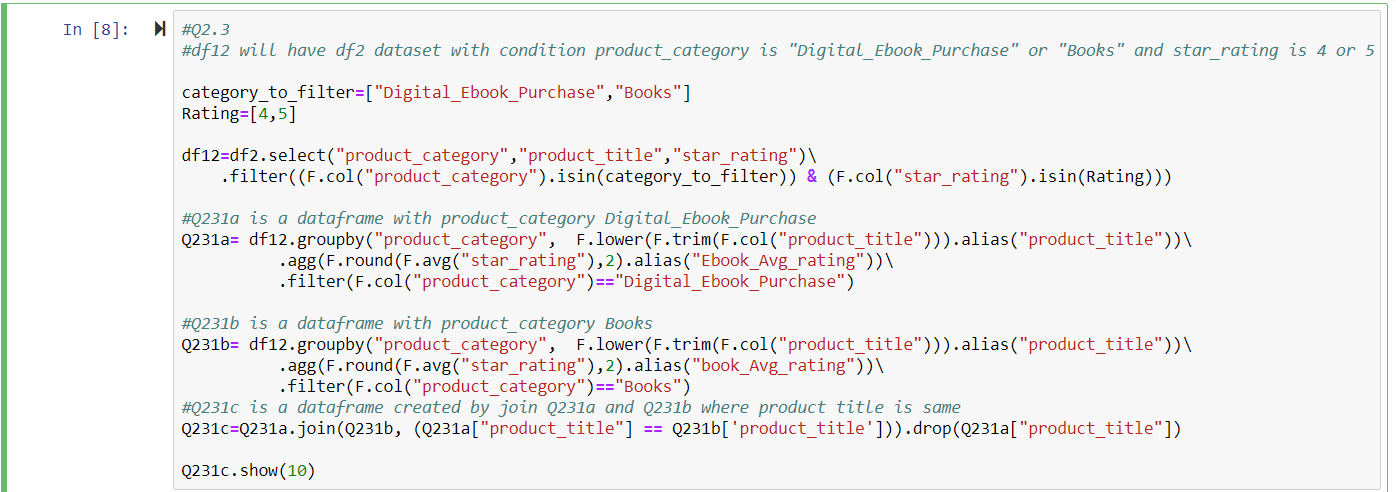
.agg(F.round(F.avg("star\_rating"),2).alias("book\_Avg\_rating"))\

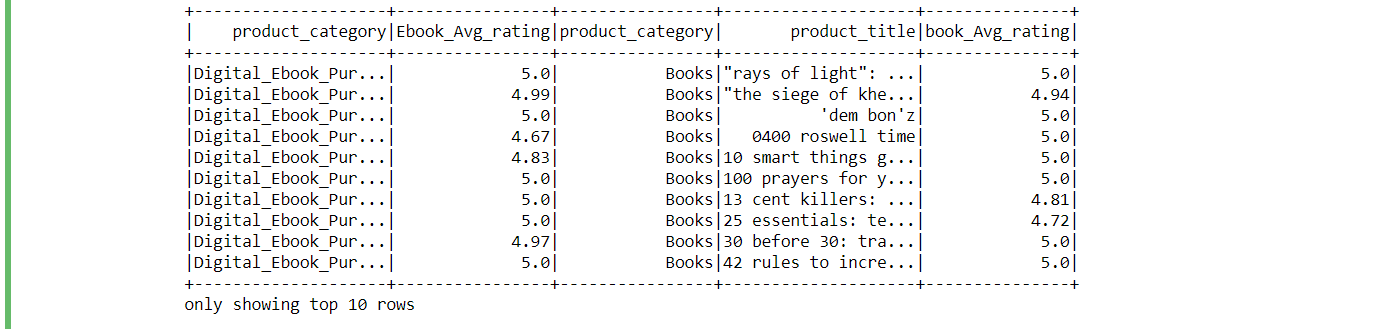
.filter(F.col("product\_category")=="Books")

#Q231c is a dataframe created by join Q231a and Q231b where product title is same

Q231c=Q231a.join(Q231b, (Q231a["product\_title"] == Q231b['product\_title'])).drop(Q231a["product\_title"])

Q231c.show(10)

****

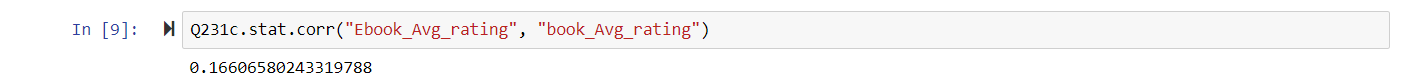
****

**1. Is there a diﬀerence in average rating for the similar books in digital and printed form?**

**2. To answer #1, you may calculate number of items with high stars in digital form versus printed form, and vise versa. Alternatively, you can make the conclusion by using appropriate pairwise statistic.**

**Code:-**

Q231c.stat.corr("Ebook\_Avg\_rating", "book\_Avg\_rating")

****

**Interpretation:** We have performed correlation between Ebooks and Books average rating for each product title to find any significant value. But the correlation of 0.166 indicates there is no significant correlation between them.

**4. Using provided LDA starter notebook, perform LDA topic modeling for the reviews in Digital\_Ebook\_Purchase and Books categories.**

**1. Perform LDA separately for reviews with 1/2 stars and reviews with 4/5 stars.**

**2. Add stop words to the standard list as needed. In the example notebook, you can see some words like 34, br, p appear in the topics.**

**3. Identify 5 top topics for each case (1/2 versus 4/5)**

**4. Does topic modeling provides good approximation to number of stars given in the review?**

**Imported ML libraries:-**

**from** **pyspark.mllib.clustering** **import** LDA, LDAModel

**from** **pyspark.mllib.linalg** **import** Vectors

**from** **pyspark.ml.feature** **import** CountVectorizer, IDF,RegexTokenizer, Tokenizer

**from** **pyspark.sql.types** **import** ArrayType

**from** **pyspark.sql.types** **import** StringType

**from** **pyspark.sql.types** **import** \*

**from** **pyspark.sql.functions** **import** udf

**from** **pyspark.sql.functions** **import** struct

**import** **re**

**from** **pyspark.ml.feature** **import** StopWordsRemover

**from** **pyspark.ml.clustering** **import** LDA

**from** **pyspark.ml.feature** **import** CountVectorizer

**Part1:-**

**Loading Dataframes with product category Digital\_Ebook\_Purchase and Books and Rating 4 or 5:**

*#df\_ml = df.filter((F.col("product\_category")=="Digital\_Ebook\_Purchase") | (F.col("product\_category")=="Books"))*

df\_ml = df.filter((F.col("product\_category")=="Digital\_Ebook\_Purchase") \

& (F.col("year")==2015) \

& (F.col("review\_date")<'2015-02-01')

& (F.col("star\_rating")>3))

**Creating Dataframe with only narrative and unique ID:**

*#from pyspark.sql.functions import monotonically\_increasing\_id, concat*

df1 = df\_ml.withColumn('review\_text',

F.concat(F.col('review\_headline'),F.lit(' '), F.col('review\_body')))

corpus =df1.select('review\_text')

*# This will return a new DF with all the columns + id*

corpus\_df = corpus.withColumn("id", F.monotonically\_increasing\_id())

*# Remove records with no review text*

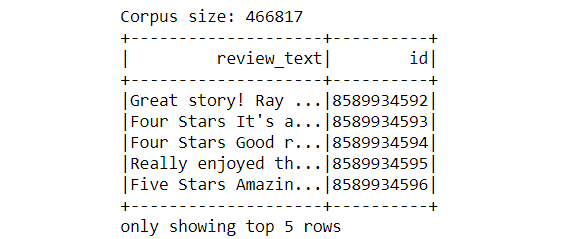
corpus\_df = corpus\_df.dropna()

**Persisting and finding the size of the dataset:**

corpus\_df.persist()

print('Corpus size:', corpus\_df.count())

corpus\_df.show(5)

****

**Tokenizing narrative text:**

tokenizer = Tokenizer(inputCol="review\_text", outputCol="words")

countTokens = udf(**lambda** words: len(words), IntegerType())

*'''*

*tokenized\_df = tokenizer.transform(corpus\_df)*

*tokenized\_df.select("review\_text", "words").withColumn("tokens", countTokens(col("words"))).show()*

*'''*

regexTokenizer = RegexTokenizer(inputCol="review\_text",

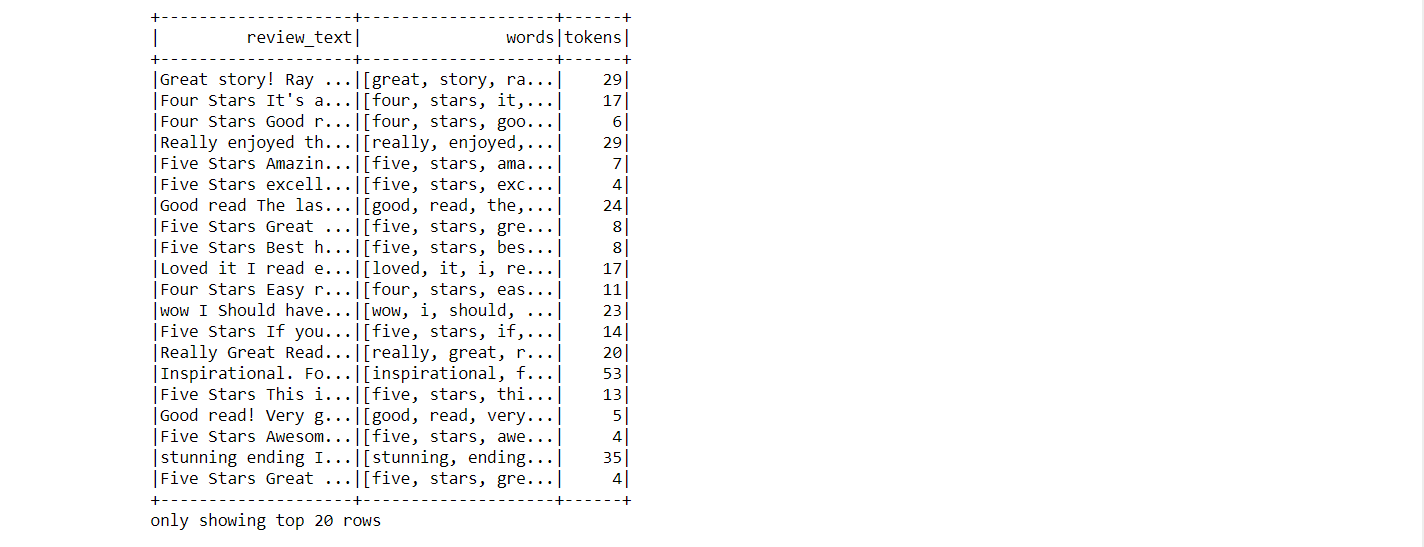
outputCol="words",pattern="**\\**w+", gaps=**False**)

*# alternatively, pattern="\\w+", gaps(False) pattern="\\W"*

tokenized\_df = regexTokenizer.transform(corpus\_df)

tokenized\_df.select("review\_text", "words") \

.withColumn("tokens", countTokens(F.col("words"))).show()

****

**Making stop words:**

stop\_words = ['a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'amoungst', 'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both', 'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'computer', 'con', 'could', 'couldnt', 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due', 'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen', 'fify', 'fill', 'find', 'fire', 'first', 'five', 'for', 'former', 'formerly', 'forty', 'found', 'four', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'hasnt', 'have', 'he', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i', 'ie', 'if', 'in', 'inc', 'indeed', 'interest', 'into', 'is', 'it', 'its', 'itself', 'keep', 'last', 'latter', 'latterly', 'least', 'less', 'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'sincere', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'system', 'take', 'ten', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'thick', 'thin', 'third', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'un', 'under', 'until', 'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves', '']

stop\_words = stop\_words + ['br','book','34']

**Removing stop words from the tokens:**

remover = StopWordsRemover(inputCol="words", outputCol="filtered")

tokenized\_df1 = remover.transform(tokenized\_df)

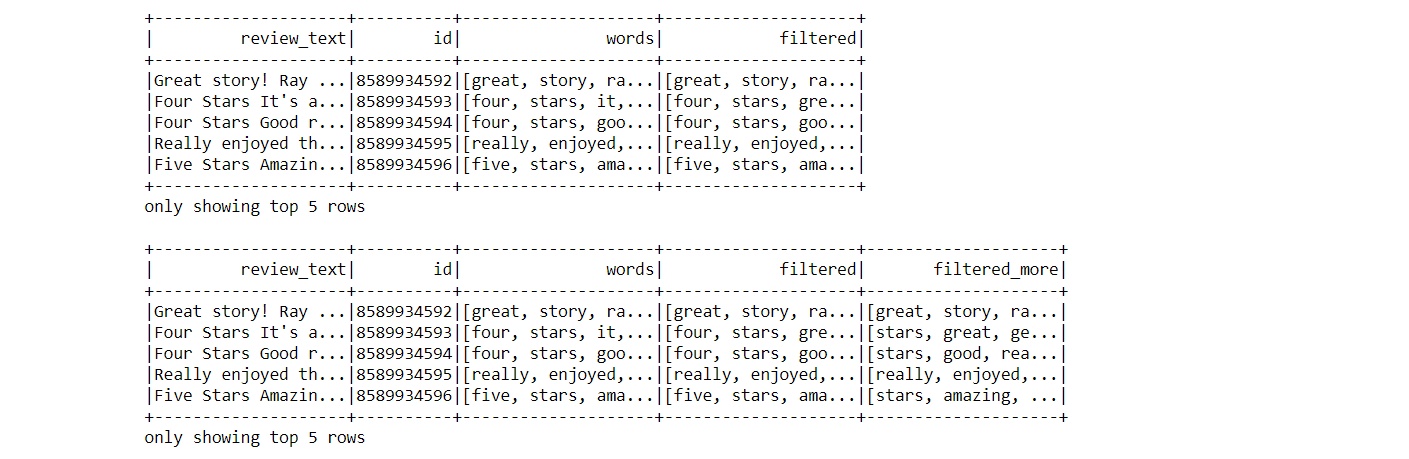
tokenized\_df1.show(5)

stopwordList = stop\_words

remover=StopWordsRemover(inputCol="filtered", outputCol="filtered\_more" ,stopWords=stopwordList)

tokenized\_df2 = remover.transform(tokenized\_df1)

tokenized\_df2.show(5)

****

**Vectorizing(converting the words into numbers)**

*# Term Frequency Vectorization - Option 2 (CountVectorizer) :*

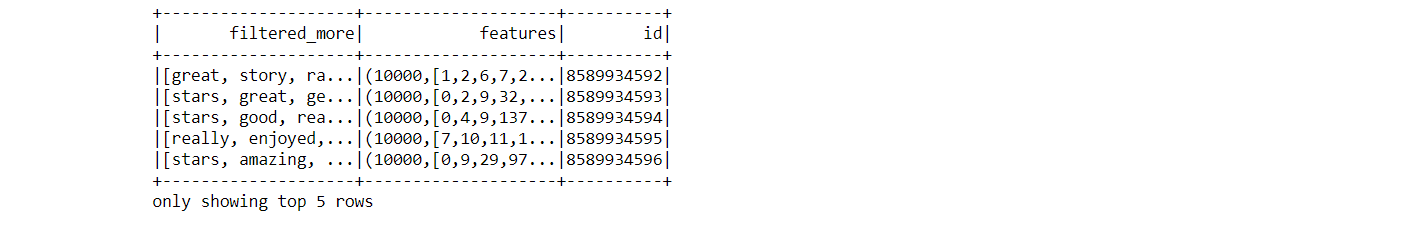
cv = CountVectorizer(inputCol="filtered\_more", outputCol="features", vocabSize = 10000)

cvmodel = cv.fit(tokenized\_df2)

featurized\_df = cvmodel.transform(tokenized\_df2)

vocab = cvmodel.vocabulary

featurized\_df.select('filtered\_more','features','id').show(5)

****

**Making the dataframe to train LDA model**

countVectors = featurized\_df.select('features','id')

countVectors.persist()

print('Records in the DF:', countVectors.count())

**Taining LDA Model:**

*#k=10 means 10 words per topic*

lda = LDA(k=5, maxIter=10)

model = lda.fit(countVectors)

**Displaying words for top 5 topics**

topics = model.describeTopics()

topics\_rdd = topics.rdd

topics\_words = topics\_rdd\

.map(**lambda** row: row['termIndices'])\

.map(**lambda** idx\_list: [vocab[idx] **for** idx **in** idx\_list])\

.collect()

**for** idx, topic **in** enumerate(topics\_words):

print ("topic: ", idx)

print ("----------")

**for** word **in** topic:

print (word)

print ("----------")

**Output:-**

topic: 0

----------

story

characters

love

read

series

author

good

great

reading

books

----------

topic: 1

----------

read

good

story

great

really

like

stars

love

enjoyed

little

----------

topic: 2

----------

read

series

books

great

stars

reading

love

loved

story

like

----------

topic: 3

----------

story

love

read

life

like

way

time

really

family

know

----------

topic: 4

----------

read

great

good

like

author

reading

story

time

books

people

----------

**Part2:-**

**Loading Dataframes with product category Digital\_Ebook\_Purchase and Books and Rating 1 or 2 :**

*#df\_ml = df.filter((F.col("product\_category")=="Digital\_Ebook\_Purchase") | (F.col("product\_category")=="Books"))*

df\_ml = df.filter((F.col("product\_category")=="Digital\_Ebook\_Purchase") \

& (F.col("year")==2015) \

& (F.col("review\_date")<'2015-02-01')

& (F.col("star\_rating")<3))

**Creating Dataframe with only narrative and unique ID:**

*#from pyspark.sql.functions import monotonically\_increasing\_id, concat*

df1 = df\_ml.withColumn('review\_text',

F.concat(F.col('review\_headline'),F.lit(' '), F.col('review\_body')))

corpus =df1.select('review\_text')

*# This will return a new DF with all the columns + id*

corpus\_df = corpus.withColumn("id", F.monotonically\_increasing\_id())

*# Remove records with no review text*

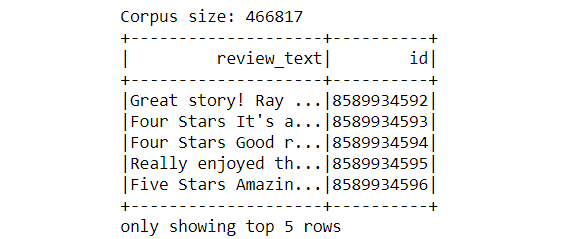
corpus\_df = corpus\_df.dropna()

**Persisting and finding the size of the dataset:**

corpus\_df.persist()

print('Corpus size:', corpus\_df.count())

corpus\_df.show(5)

****

**Tokenizing narrative text:**

tokenizer = Tokenizer(inputCol="review\_text", outputCol="words")

countTokens = udf(**lambda** words: len(words), IntegerType())

*'''*

*tokenized\_df = tokenizer.transform(corpus\_df)*

*tokenized\_df.select("review\_text", "words").withColumn("tokens", countTokens(col("words"))).show()*

*'''*

regexTokenizer = RegexTokenizer(inputCol="review\_text",

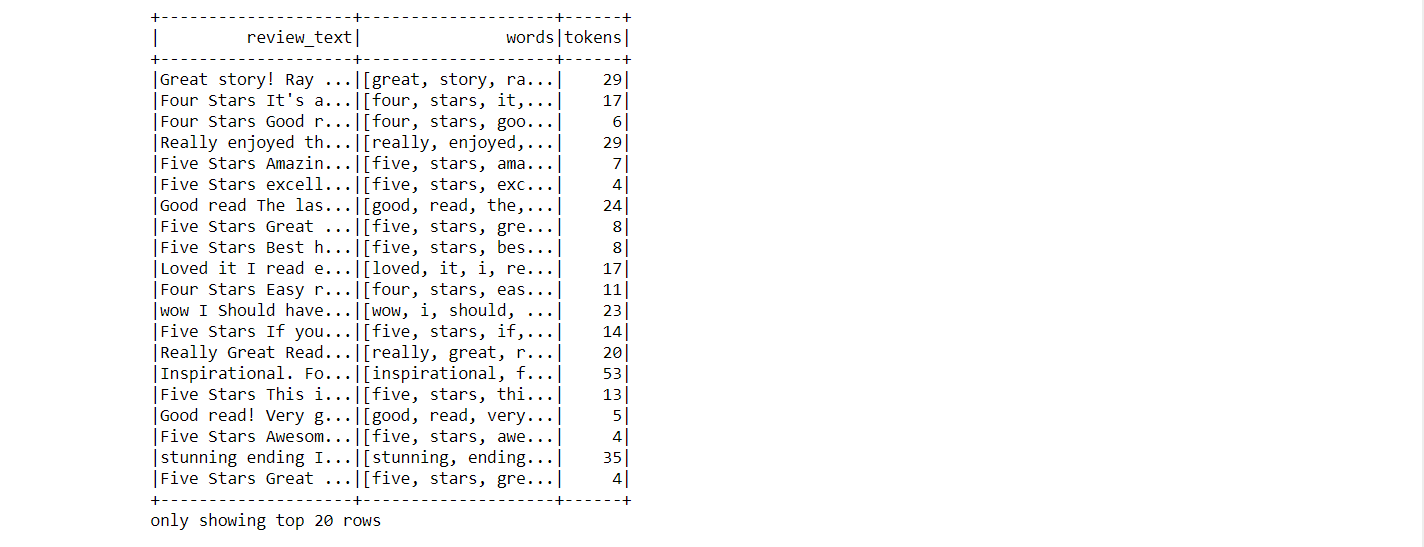
outputCol="words",pattern="**\\**w+", gaps=**False**)

*# alternatively, pattern="\\w+", gaps(False) pattern="\\W"*

tokenized\_df = regexTokenizer.transform(corpus\_df)

tokenized\_df.select("review\_text", "words") \

.withColumn("tokens", countTokens(F.col("words"))).show()

****

**Making stop words:**

stop\_words = ['a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'amoungst', 'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both', 'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'computer', 'con', 'could', 'couldnt', 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due', 'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen', 'fify', 'fill', 'find', 'fire', 'first', 'five', 'for', 'former', 'formerly', 'forty', 'found', 'four', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'hasnt', 'have', 'he', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i', 'ie', 'if', 'in', 'inc', 'indeed', 'interest', 'into', 'is', 'it', 'its', 'itself', 'keep', 'last', 'latter', 'latterly', 'least', 'less', 'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'sincere', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'system', 'take', 'ten', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'thick', 'thin', 'third', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'un', 'under', 'until', 'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves', '']

stop\_words = stop\_words + ['br','book','34']

**Removing stop words from the tokens:**

remover = StopWordsRemover(inputCol="words", outputCol="filtered")

tokenized\_df1 = remover.transform(tokenized\_df)

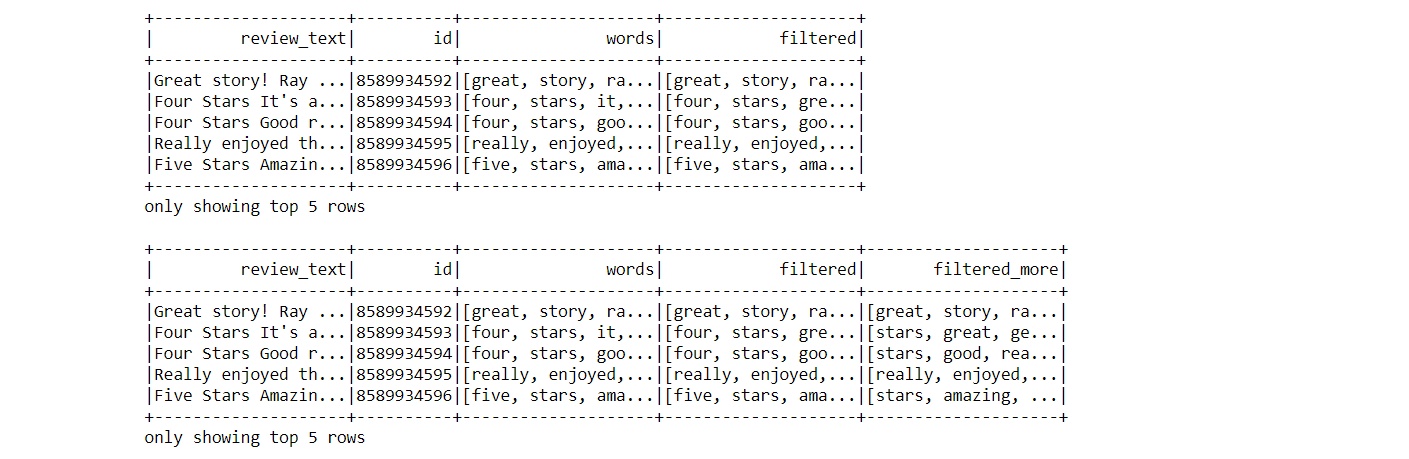
tokenized\_df1.show(5)

stopwordList = stop\_words

remover=StopWordsRemover(inputCol="filtered", outputCol="filtered\_more" ,stopWords=stopwordList)

tokenized\_df2 = remover.transform(tokenized\_df1)

tokenized\_df2.show(5)

****

**Vectorizing(converting the words into numbers)**

*# Term Frequency Vectorization - Option 2 (CountVectorizer) :*

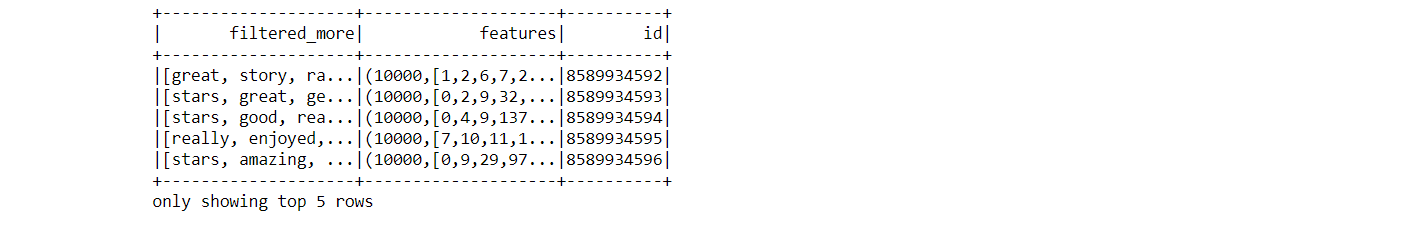
cv = CountVectorizer(inputCol="filtered\_more", outputCol="features", vocabSize = 10000)

cvmodel = cv.fit(tokenized\_df2)

featurized\_df = cvmodel.transform(tokenized\_df2)

vocab = cvmodel.vocabulary

featurized\_df.select('filtered\_more','features','id').show(5)

****

**Making the dataframe to train LDA model**

countVectors = featurized\_df.select('features','id')

countVectors.persist()

print('Records in the DF:', countVectors.count())

**Taining LDA Model:**

*#k=10 means 10 words per topic*

lda = LDA(k=5, maxIter=10)

model = lda.fit(countVectors)

**Displaying words for top 5 topics**

topics = model.describeTopics()

topics\_rdd = topics.rdd

topics\_words = topics\_rdd\

.map(**lambda** row: row['termIndices'])\

.map(**lambda** idx\_list: [vocab[idx] **for** idx **in** idx\_list])\

.collect()

**for** idx, topic **in** enumerate(topics\_words):

print ("topic: ", idx)

print ("----------")

**for** word **in** topic:

print (word)

print ("----------")

**Output:-**

topic: 0

----------

story

love

read

characters

series

good

great

author

loved

reading

----------

topic: 1

----------

read

good

story

great

really

like

love

stars

time

didn

----------

topic: 2

----------

read

great

stars

books

series

story

reading

like

characters

loved

----------

topic: 3

----------

story

love

read

life

like

way

family

time

characters

written

----------

topic: 4

----------

read

good

great

author

like

reading

story

time

books

people

----------

**Interpretation:**We have performed LDA for both low and high ratings but haven’t observed any significant difference between the reviews. Some of the common topics are good, great, read, love and like, though we expected for bad rating the top words will be bad, verybad. Thus, we can say that LDA is not suitable for this review dataset.

**Conclusion:**

After handling the big data and performing exploratory analysis on the Amazon review dataset, I came out with various hidden patterns. The features I focused on are review count, product id, star rating and reviews. Observing them I found that only 2 categories i.e. Books and Digital Ebooks Purchase category constitutes more than 53% of the dataset. All the product categories are showing a growth in number of customers, but Digital Ebook Purchase has observed the maximum growth rate followed by Books and Wireless categories. There is a growth in average rating for all product categories but the Mobile apps experienced the maximum growth rate in average rating. A similar pattern is shown while observing the medians of the product categories. The main reason behind rapid increase in average and median rating of Mobile Apps might be improve in technologies, making apps more user friendly with no bugs. I used various sparks functionalities like percentiles and pivots and found interesting information like there are only 5% of the reviews have length greater than 51,019 words. One might think that as the as technologies improving, people are shifting from paper to online, but the data shows the other way there are still more number of people who prefer buying books than Ebooks, though there is close competition between them. From 2005 to 2010 there was almost no to very less purchase for Digital ebooks but in 2014 both the categories are sharing the same peak. Thus, we can say that there has been a higher growth rate in Digital books than Books category. On observing review count per month between Books and Digital Books I found that there is a higher purchase between March and August. I also performed correlation test between Digital Ebooks Purchase and Books average rating for each product title to find any significant value, but the value of 0.166 indicates there is no a significant correlation between them. I performed a LDA model for both low and high ratings for top 2 categories to find any significant pattern, but haven’t observed any significant difference between the reviews. Some of the common topic words are good, great, read, love and like, though I expected for bad rating the top words will be bad, very bad. Thus, I can say that LDA is not suitable for this review dataset.

**References:-**

<https://searchbusinessanalytics.techtarget.com/definition/big-data-analytics>

<https://www.ibm.com/analytics/hadoop/big-data-analytics>

<https://aws.amazon.com/emr/faqs/>