Title: Seattle's Public Library Book Trends based on Precipitation.

Authors: Sean Stevens, Jin Young Park, Matt Choi, Jaewoo Cho

Introduction

Our project centers around finding the correlation (if any exists) between the types of books that are checked out from the Seattle Public Library and the amount of rain accumulation that is recorded each month. This topic seemed interesting because weather can affect moods and behavior, and we wanted to see if more or less rain in a particular month influenced the kinds of books that people wanted to read. Analysis of this kind could be helpful for local libraries or retail booksellers to determine which books they should promote during certain seasons (for instance, if a strong correlation is drawn between checkouts of mystery books during months with heavy rain, a bookseller may wish to run a promotion on this genre of book during the fall or winter seasons).

The goal stated in our original project proposal did not change much throughout the course of this project, although there were several hiccups that we had to overcome (these are detailed in the **Data** and **Setup Challenges** sections). Generally, we found that our datasets were fairly messy and needed to be cleaned up quite a bit to be usable by our primary Pig Latin script. Cleanup involved several helper tasks that removed nulls and duplicates of assumed unique join keys, and conversion of values to allow for simpler join logic.

Data

As for the progress report, briefly describe the data you used, where you got them from,
and if you had to do anything special to make them available for your project.
(For example, if you had to collect data in real-time from Twitter, briefly explain how you did this.
Or if you did something clever to get a big data set uploaded to S3, briefly describe your approach.
This will count as a helper task.) Show a few lines of each input file.

Datasets

1. Seattle Public Library Checkouts by Title (City of Seattle Open Data portal)
2. Book Cover and Genre Classification Dataset (Github)
3. Library Collection Inventory (City of Seattle Open Data portal)
4. Seattle Observed Monthly Rain Gauge Accumulations (City of Seattle Open Data portal)

Data Normalization

- **Seattle Public Library Checkouts by Title**
  This dataset contains the data regarding all the checkout information with details such as checkout year, checkout month, Title, ISBN, and the number of checkouts, and so on. Since this dataset was large(originally 9.21 GB), we had to normalize it such as dropping unnecessary columns and leaving the columns we needed only which were CheckoutYear, CheckoutMonth, Title, Checkouts. We used Python Pandas to normalize this data. We read this csv data using Pandas and put it into the Pandas dataframe with names of columns, then dropped unnecessary columns. We dropped the ISBN column

because most of the ISBNs were empty, and we could add the ISBN column by joining with the Library Collection Inventory dataset using Title. Through this normalization process, we were able to reduce the size from 9.21 GB to 5.81 GB.

- **Book Cover and Genre Classification Dataset**
  This dataset contains the data regarding 137,788 books in 32 classes(Genre) with columns of AMAZON ISBN, FILENAME, IMAGE_URL, TITLE, AUTHOR, CATEGORY_ID, CATEGORY. There were two normalization operations required. First, this dataset was missing the column names, so we used Python Pandas to import this dataset into the Pandas dataframe by giving column names as the original dataset. So that it can clarify the data without misinforming readers or users. Second, since some spreadsheet applications like Excel or Numbers automatically remove the leading zeros in the datacell, the ISBNs starting with zero were missing all the leading zeros which causes data integrity issues. We used the apply function in Pandas to apply a lambda operation which validates if an ISBN is less than 10, we added zeros in front of the ISBN (10 - the size of the ISBN = the number of zeros to be added). Then we wrapped the ISBN with a double quotation mark such as, “1234567890”. In this way, we were able to improve the data integrity issue with ISBN.

- **Library Collection Inventory** This dataset contains the data regarding all the books that Seattle Public Library stores, for example Title, Author, ISBN, and many other columns. The original size of this dataset was extremely huge(34.39 GB). This data was very crucial to this project because the Library Checkouts dataset was missing ISBN data but we were able to get the ISBN of each book by Joining them using the title. Before we normalize the data, we analyzed the data and we found that there were several issues to resolve. First, there are so many duplicates with the same Title. Second, the rows with missing ISBN. Third, some ISBNs with multiple ISBNs. We only needed the first ISBN which is 10 digits Amazon Specialized ISBN not the full 13 digits ISBN. Fourth, we only needed two columns which are Title and ISBN, so we needed to drop all the rest of the columns.

  For this normalization, we used Python Pandas. We read the csv using Pandas first and put it into the Pandas dataframe. First, we operated the ‘drop_duplicates’ function from Pandas for the Title column which removed all the rows with duplicate titles. Second, we filtered out the rows with ISBN containing the value of ‘nan’. Third, we operated Pandas apply function with lambda equation which validates if ISBN with multiple ISBNs, then only assigns the ISBN with the first ISBN which is 10 digits. Lastly, we reframed the dataframe to only contain Title and ISBN columns. Through this normalization, we were able to reduce the size of the data from 34.39 GB to 61.5 MB (Specifically, reduce the rows from 100 million to about 500,000).

- **Seattle Observed Monthly Rain Gauge Accumulations**
  This dataset is provided with a relatively simple schema with the first column containing the date at which the readings were recorded, followed by a series of columns that list

out the number of inches of rain accumulated in a particular gauge, to 1/100th of an inch precision. One of the pre-processing tasks that needed to be done with this data was to convert the date format in the first column to a format that would more easily mesh with the date formats throughout the rest of our data. To do this, a Pig Latin script was written that would parse each date entry in the first column of the dataset, then generate two new columns that contained the year and month values for that row. Since the rest of our data is aggregated on a per-month basis, we could discard the day value from the record date. Then, the rain gauge reading columns would be re-merged with the month and year columns to produce the final output.

Another helper task that needed to be performed on this dataset was pre-aggregating the rain gauge data so that we had one overall rain accumulation value instead of separate readings from across Seattle. This would simplify our primary data processing task, since we would not need to average the accumulation readings in the same script as we are performing joins and applying filters. To accomplish this, each line was read in through the aforementioned Pig script that converted the date values, and every column after the date column to the end were summed up and divided by the number of readings. Then, this new relation was merged with the month and year columns to produce the aggregated output for use in our main Pig script.

Technical Discussion.

Pig Script

To find out the correlation between precipitation and book checkout trends by genre, we collected multiple datasets to find the count of book checkouts per genre from Seattle Public Library. Since the Seattle Public Library only published book checkout dataset by title, we had to join the checkout dataset with library collection inventory dataset which has each book's title and its isbn number. Then with the joined dataset, we added each book's genre by joining it with another dataset that contains each book's isbn and its genre. Finally, we used monthly rain gauge accumulations dataset from Seattle Open Data to join it with our dataset to add average rain gauge for each month.

When we first joined the book checkout dataset by title and library collection inventory dataset, we had to match each book's title. To do so, we removed spaces and special characters and had every letter to lowercase. Rows with no title (empty) and duplicated rows are removed as well. For joining genres and rain gauge datasets, we used isbn and year/month columns.

After joining, we grouped the dataset by year, month, genre, and rain gauge. Then we summed up all the checkouts per group. Below is our full code of our primary Pig Script:

```
// Loading Checkout by Title dataset and change the titles to match with Library Collection Inventory dataset
Checkout = LOAD 's3://data/Checkouts_by_Title.csv' USING CSVLoader(',') as (UsageClass: chararray, CheckoutType: chararray, MaterialType: chararray, CheckoutYear: int, CheckoutMonth: int, Checkouts: int, Title: chararray, isbn: chararray, Creator: chararray, Subjects: chararray, Publisher: chararray,
```

```
PublicationYear: chararray);
c1 = FOREACH Checkout GENERATE CheckoutYear, CheckoutMonth, Title, Checkouts;
c2 = FOREACH c1 GENERATE REPLACE(Title,' ','') as Title, CheckoutYear, CheckoutMonth, Checkouts;
c3 = FOREACH c2 GENERATE REPLACE(Title, '([^a-zA-Z\\s]+)', '') as Title, CheckoutYear, CheckoutMonth, Checkouts;
c4 = FOREACH c3 GENERATE LOWER(Title) as Title, CheckoutYear, CheckoutMonth, Checkouts;
c5 = FILTER c4 BY Title != '';
c6 = DISTINCT c5;

// Loading Library Collection Inventory dataset and change the titles to match with Checkout by Title dataset
l = LOAD 's3://data/normalized_book32_listing.csv' USING CSVLoader(',') as (Rank: int, Title: chararray, isbn: chararray);
l2 = FOREACH l GENERATE REPLACE(Title, '([^a-zA-Z\\s]+)', '') as Title, isbn;
l3 = FOREACH l2 GENERATE REPLACE(Title, ' ', '') as Title, isbn;
l4 = FOREACH l3 GENERATE LOWER(Title) as Title, isbn;
l5 = FOREACH l4 GENERATE REPLACE(isbn,'"','') as isbn, Title;
l6 = FILTER l5 BY (isbn != 'nan') AND (Title != '') AND (isbn != '');
l7 = DISTINCT l6;

// Joining Checkout by Title dataset and Library Collection Inventory dataset
r = JOIN c6 BY (Title), l5 BY (Title);

// Loading book genre dataset and join it with our joined dataset using isbn
Book = LOAD 's3://dominic-bucket/project/input/normalized_book32_listing copy.csv' USING CSVLoader(',') as (order: Int, isbn: chararray, Filename: chararray, ImageURL: chararray, Title: chararray, Author: chararray, CategoryID: chararray, Category: chararray);
b1 = FOREACH Book GENERATE REPLACE(isbn,'"','') as isbn, Category;=
r2 = JOIN r BY (isbn), b1 BY (isbn);

// Loading rain gauge dataset and join it with our joined dataset using year and month
Rain = LOAD 's3://dominic-bucket/project/input/rain_gauge_data_normalized_averaged.csv' USING PigStorage(',') as (year: int, month: int, AvgGauge: float);
f = JOIN r2 BY (CheckoutYear, CheckoutMonth), Rain BY (year, month);

// Group the dataset by year, month, genre, rain gauge and sum up the checkout counts.
f1 = GROUP f BY (Rain::year, Rain::month, r2::b1::Category, Rain::AvgGauge);
f2 = FOREACH f1 GENERATE FLATTEN(group) AS (year, month, cat, gauge), SUM(f.r2::r::c6::Checkouts) AS cnt;
order_by_data = ORDER f2 BY gauge DESC, cnt DESC;
```

There are two versions of pig scripts, joinfirst and filterfirst. Joinfirst joins datasets first and filter out titles that are empty or duplicate, and filterfirst filters out duplicates or empty titles out first and then joins datasets. We used 6 machines (1 master and 5 cores), 11 machines (1 master and 10 cores), and 16 machines (1 master and 15 cores) when we ran through AWS. Joinfirst failed to run since it took too much space and too much time (over 200 GB data output and over 10 hours of running time) despite how many machines or how much additional space we tried. For filterfirst, it successfully ran for all settings and below is a table showing the difference in aspects such as running time from syslog files for each setup. Execution with 16 machines had the lowest running time, mapper output, shuffle bytes disk, and DAG file bytes read and written compared to execution with 6 machines and 11 machines.
Overall, it read 41389466 records from Checkout by Title dataset, 105411020 records from Library Collection Inventory dataset, 207592 records from book genre dataset, and 175 records from rain gauge dataset. After executing the script, it generated 4651 records as output.

Filterfirst	6 machines	11 machines	16 machines
Running time (sec)	286	194	154
Map Output (bytes)	36484782	6511700	5224917
ShuffleBytesDisk	1174463235	520736260	373337115
DAG-FileBytesRead	8396027399	4997460052	4514786456
DAG-FileBytesWritten	11859710967	9612053742	9231296987

Optimizer
To see how the built-in Pig optimizer affected the run time of the script on our datasets, we performed several test runs, turning off one optimization rule at a time. The table of these run time results is below. For the purposes of these tests, each run was performed on a cluster with 1 master node and 4 worker nodes.

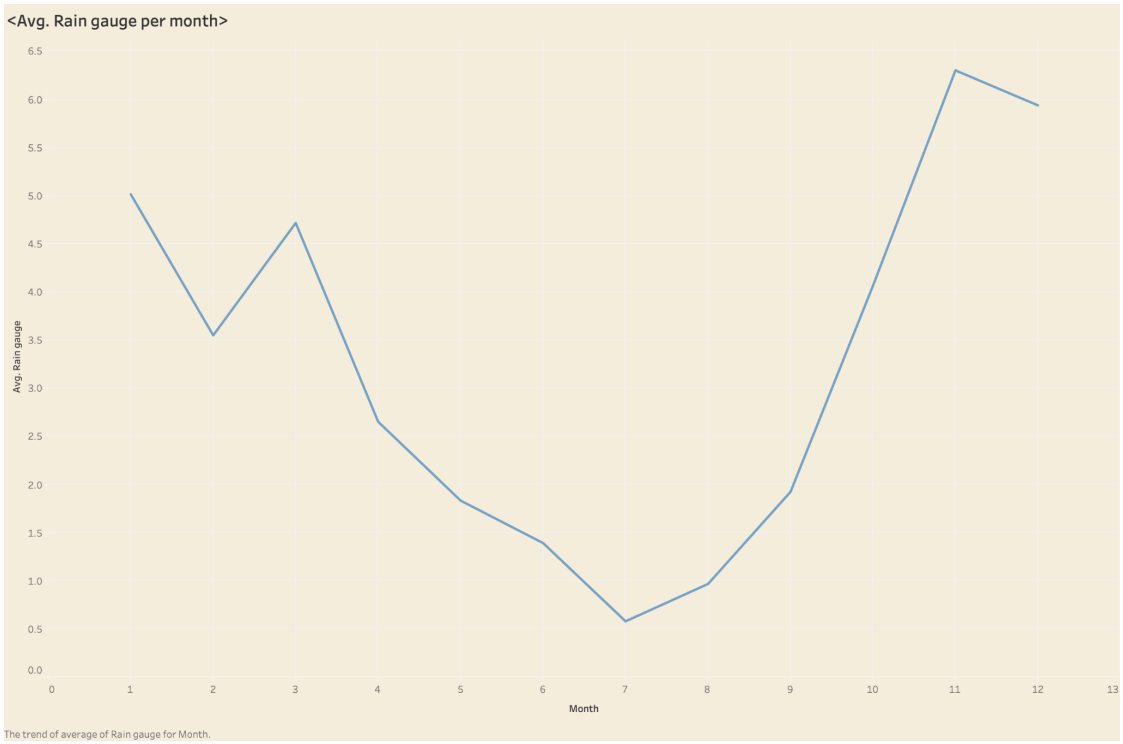
Optimization Rule Removed	**Description**	Running time (sec)
ConstantCalculator	Calculate constants at compile time	358
SplitFilter	Split filter conditions	358
PushUpFilter	Filter as early as possible	364
MergeFilter	Merge filter conditions	364
PushDownForeachFlatten	Join or explode as late as possible	364
LimitOptimizer	Limit as early as possible	364

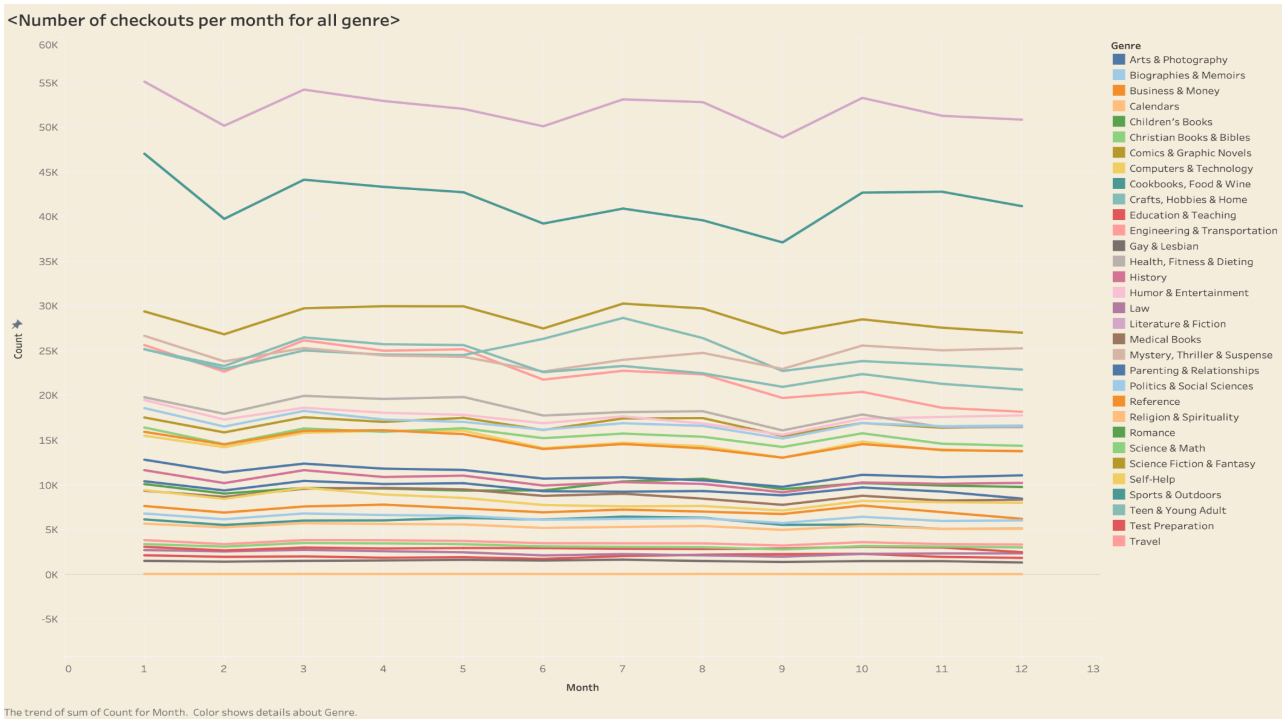
ColumnMapKeyPruneAddForEach	Remove unused data	366
AddForEach	Add ForEach to remove unneeded columns	366
MergeForEach	Merge adjacent ForEach	374
GroupByConstParallelSetter	Force parallel 1 for "group all" statement	366
PartitionFilterOptimizer	Pushdown partition filter conditions to loader implementing LoadMetaData	368
PredicatePushDownOptimizer	Pushdown filter predicates to loader implementing LoadPredicatePushDown	368
No Optimizations	No optimizations applied	374

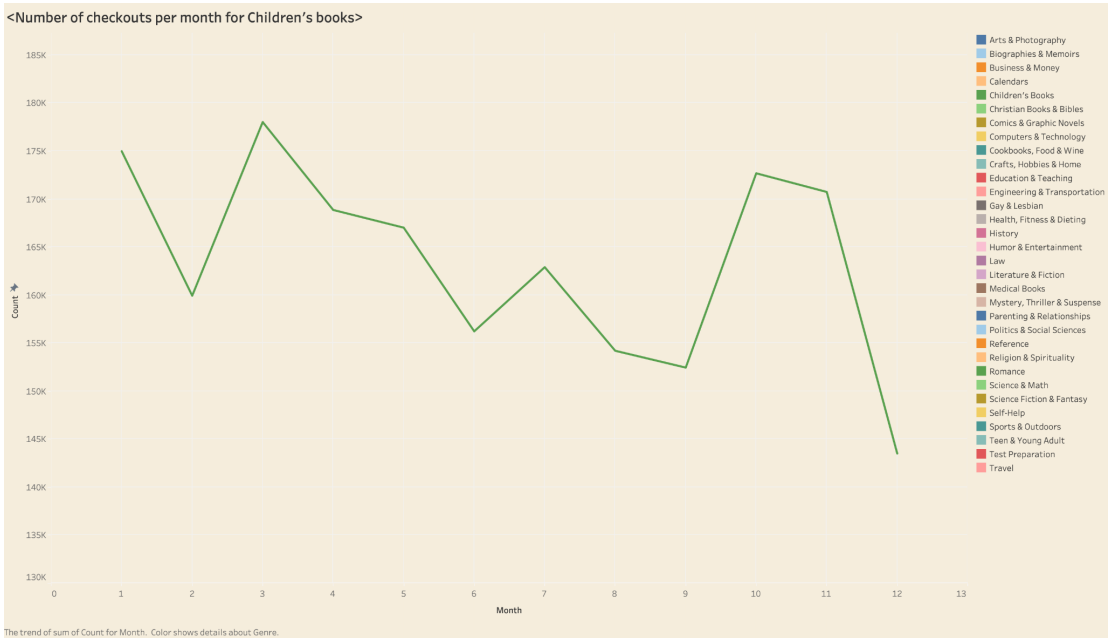
From the above comparison table, we can see that the MergeForEach rule, which merges multiple adjacent ForEach statements into as few as possible, had the greatest impact on the overall run time of the script. Our script heavily relies on ForEach statements to normalize the data as it after loading, and we have many adjacent ForEach statements that occur without further processing occurring between the statements. The MergeForEach rule was thus able to eliminate several iterations of our large data sets by performing a single pass on each of the rows. This rule likely would not have made a significant difference with a smaller data set, but because our data contains roughly 140 million rows, bringing the constant number of iterations down made a significant impact on overall run time.

This finding is further reinforced by noticing that the times of the runs with the MergeForEach optimization turned off and all optimizations turned off are identical. We can infer from this that the other optimization rules do not have a significant impact on the overall run time of the script, since the bulk of our script is concerned with joining and normalizing the data.

Result graphs:







The interesting part of our result was that the number of checkouts for children's books were too big relative to other genres. If we made a graph for all genres in one plot, it made all other genres look like a straight line. So in order for us to show other genres' trends, we made a separate graph for children's books and all other genres. This way, we can also see the trends for other genres.

Setup Challenges

After multiple failures to run joinfirst version, we figured out it kept failing due to lack of space we set up for the cluster which was 64gb for master node and core nodes. We changed the EBS storage setup to 100gb and increased the number of machines. It ran without error but kept running even after 10 hours and was creating more than 200gb of output data as it was joining the Checkout by Title dataset and Library Collection Inventory dataset. After thorough checking of the datasets, we found out that rows with empty title and duplicated rows are joining with each other and creating mass output data. There was no way to run the program efficiently if we join datasets first without filtering out those rows. On the other hand, filterfirst ran in 4 minutes on average without any problems. Therefore, we chose to use filterfirst to explore various optimizers.

Individual Contribution

Sean Stevens:

I had two main tasks for this project: normalization and conversion of the rain gauge data set and comparing the effect that turning off certain parts of the Pig optimizer had on the run time of the main program. I authored a Pig script to convert the dates in the rain gauge data set to a workable format that meshed with the existing date format in our other data sets. The same Pig script also produced an aggregated data set that averaged the rain gauge readings across all the gauges in each row. My other task comparing run times of the primary Pig script with different optimizations turned off required 13 separate steps being run in my EMR cluster and recording the results in our project report. My analysis for this portion required research into what each of the optimization rules does and whether the rule would apply to our program or not, in addition to determining how the rule might change the compiled output of our script contents.

Seungju Choi:

My main task was data processing and normalization. I mainly supported Jin Young Park by providing him the best quality of processed data for his part. I reviewed and analyzed the datasets and made solutions on how to improve data quality via data normalization process, such as removing duplicates and unnecessary data, and filtering data. Specifically, I normalized three different datasets 1. Seattle Public Library Checkouts by Title (City of Seattle Open Data portal) - Data Size: from 9.21 GB to 5.81 GB after Normalization, 2. Book Cover and Genre Classification Dataset (Github), and 3. Library Collection Inventory (City of Seattle Open Data portal) - Data Size: from 34.39 GB to 61.5 MB after Normalization. The tool I used for data normalization is Python Pandas since it is very easy to use(only a few lines of codes were needed) with various Pandas builtin functions and many resources(instructions) available on the internet. If I had more time to work on data normalization, I would try to learn about the Dask framework which is a Python library for parallel computing and apply it for the project. Since I had to handle huge csv files, I believe I could reduce the data processing time with Dask compared to Pandas since Dask is specialized to large parallel data processing.

Jin Young Park:

My task was to create different versions of Pig Latin script for this project. I tried various orders by mixing and matching joins and filters of four datasets. After I got expected output with the script using smaller sample datasets, I used AWS EMR to run the script with actual data. The datasets were so large that it was not runnable locally. While using AWS, I continued to fix the errors and problems by changing the cluster setups. Once I fixed all the errors and problems, I ran the program with 6 machines, 11 machines, and 16 machines and compared the results and performances such as running time, mapper output, and others.

Jaewoo Cho:

My task was to come up with a good visualization that shows the correlation of rain accumulation and the kinds of books that people checked out to read. At first, it was hard to interpret the final data that we've processed. It was hard to find or see any impact of rain accumulation on the kinds of books that people check out with just the data. After trying out

different types of graphs, I decided to make multiple line graphs. I thought the line graph would be the best type of graph to see the trend of the number of checkouts for different kinds of books per month. I created 1. Avg rain gauge per month 2. Number of checkouts per month 3. Number of checkouts per month for Children's books 4. Number of checkouts per month for different genres. 5. Ratio of number of checkouts to rain gauge per month.

Project Conclusion

Result

By looking at the graph, we could see that in general, the number of checkouts for every type of book have increased dramatically from the September-October time frame. Interestingly, the September-October time frame is when the rainy season starts. Also from January to March, the number of checkouts for all genres follow the trend of rain accumulation. The average rain gauge decreases from January to February and then increases again from February to March. At the same time, the number of checkouts also decreases from January to February and increases again from February to March. Other than those two points of time frame, we couldn't find any relationship. Most of the genres followed these trends, however, for some genres of the book that are directly related to the weather such as Travel, sports & outdoors genres, had a different trend of number of checkouts during the rainy season. For these genres, the number of checkouts decreased starting September which makes sense since when the rainy season starts, people tend to stay home more and not really think about outdoor activities.