Data Engineering Case Study

Prompt

Download Data

We will be exploring Lending Club's loan origination data from 2007-2018. Download from Kaggle

Part 1: Data Exploration and Evaluation

Create an exploratory data analysis project. Load the data and perform any necessary cleaning and aggregations to explore and better understand the dataset. Based on your exploration, please describe your high level findings in a few sentences. Please include two data visualizations and two summary statistics to support these findings.

Part 2: Data Pipeline Engineering

Build a prototype of a production data pipeline that will feed an analysis system (data warehouse) based on this dataset. This system will allow data scientists and data analysts to interactively query and explore the data, and will also be used for machine learning model training and evaluation. Assume that the system will receive periodic updates of this dataset over time, and that these updates will need to be processed in a robust, efficient way. For this section, please:

- *Create a data model / schema in a database or storage engine of your choice.*
- *Develop code that will persist the dataset into this storage system in a fully automated way.*
- *Include any data validation routines that you think may be necessary.*

Prioritize simplicity in your data model and processing code. Explain your thought process and document any alternate data models you considered along the way. Finally, wrap up with a discussion of system improvements that could be addressed in the future.

Case Study

Part 1: Data Exploration and Evaluation

As one would expect, the mean interest rate and default rate rise as the grade decreases. Many who look at the Lending Club data are particularly interested in the platform as an alternative investment. Using a simple rate of return, it is clear that the mean rate of return decreases dramatically with lower grade loans, much of which is attributable to low-grade loans that quickly default.

Grade	Ratio	Mean Interest Rate	Default Rate	Mean Rate of Return	Median Rate of Return
A	25.37%	7.06%	3.82%	2.72%	5.96%
B	33.27%	10.67%	9.29%	1.06%	8.63%
C	26.11%	14.07%	15.11%	-2.52%	11.38%
D	11.43%	18.08%	20.51%	-7.49%	13.89%
E	3.14%	22.03%	25.50%	-13.40%	15.37%
F	0.54%	25.34%	34.03%	-24.15%	14.14%
G	0.13%	27.98%	40.47%	-40.44%	14.96%

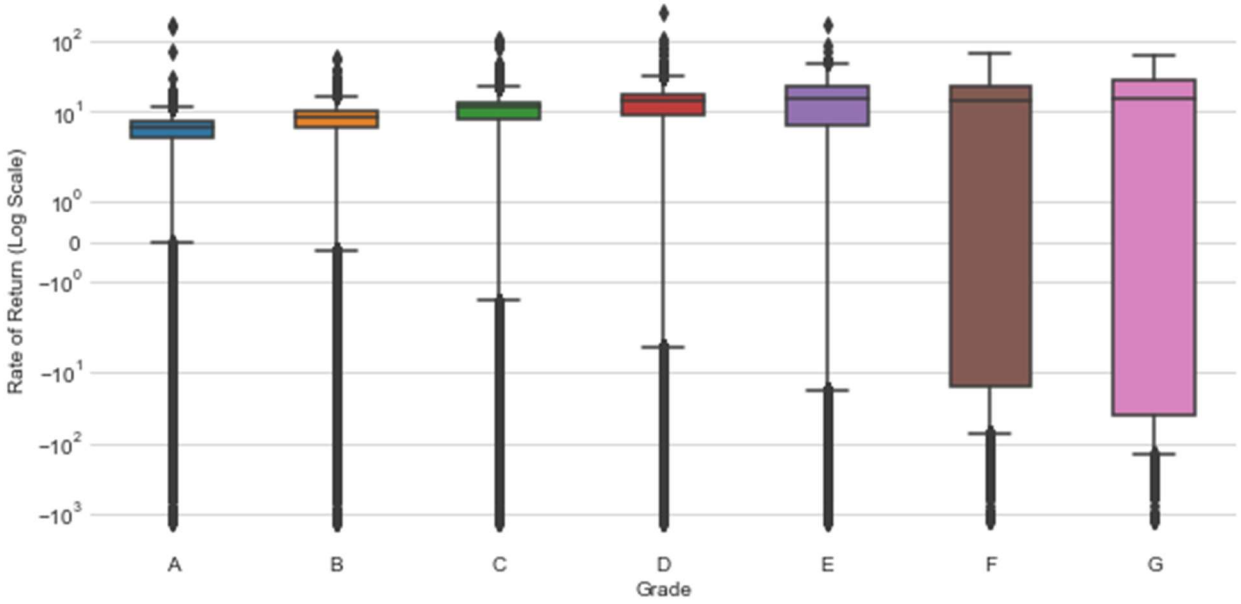


Figure 1: Distribution of Simple Annualized Rate of Return by Loan Grade

Equally important is to understand how the loan statistics change year to year to give insight into the strictness of Lending Club's underwriting process. Clearly, the Mean Debt-to-Income ratio is increasing across all grades while Lending Club is also inconsistent about verifying borrowers income. This can imply that Lending Club is issuing riskier loans. Excluding 2017 and 2018 (as those years are too recent for the loans to play out) the default rate has increased every year from 2010 to 2016.

Year	Avg. Annual Income	Verified Income	Avg. FICO	Mean DTI	Default Rate
2007	$60,524.57	0.00%	690	10.90x	26.54%
2008	$62,925.76	14.71%	699	13.24x	20.55%
2009	$66,357.08	40.56%	717	12.51x	13.47%
2010	$65,452.83	45.20%	716	12.94x	10.68%
2011	$65,474.57	61.26%	718	13.50x	10.49%
2012	$65,815.82	54.69%	703	16.45x	13.50%
2013	$69,813.13	61.15%	697	16.85x	12.26%
2014	$70,746.86	63.41%	694	17.59x	13.70%
2015	$72,015.66	70.46%	696	18.53x	14.91%
2016	$73,943.50	72.09%	697	18.61x	15.67%
2017	$73,804.82	63.17%	702	18.73x	10.46%
2018	$74,698.31	56.44%	708	19.17x	3.62%

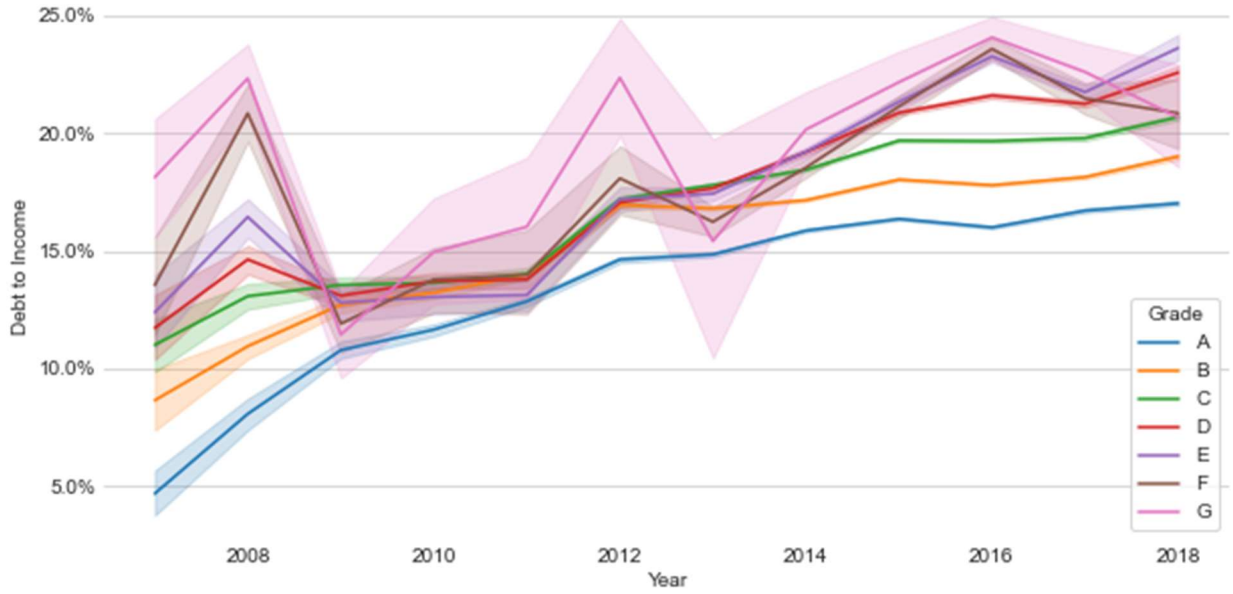


Figure 2: Debt-to-Income per Year. Shaded Region Represents 95% Confidence Interval

Part 2: Data Pipeline Engineering

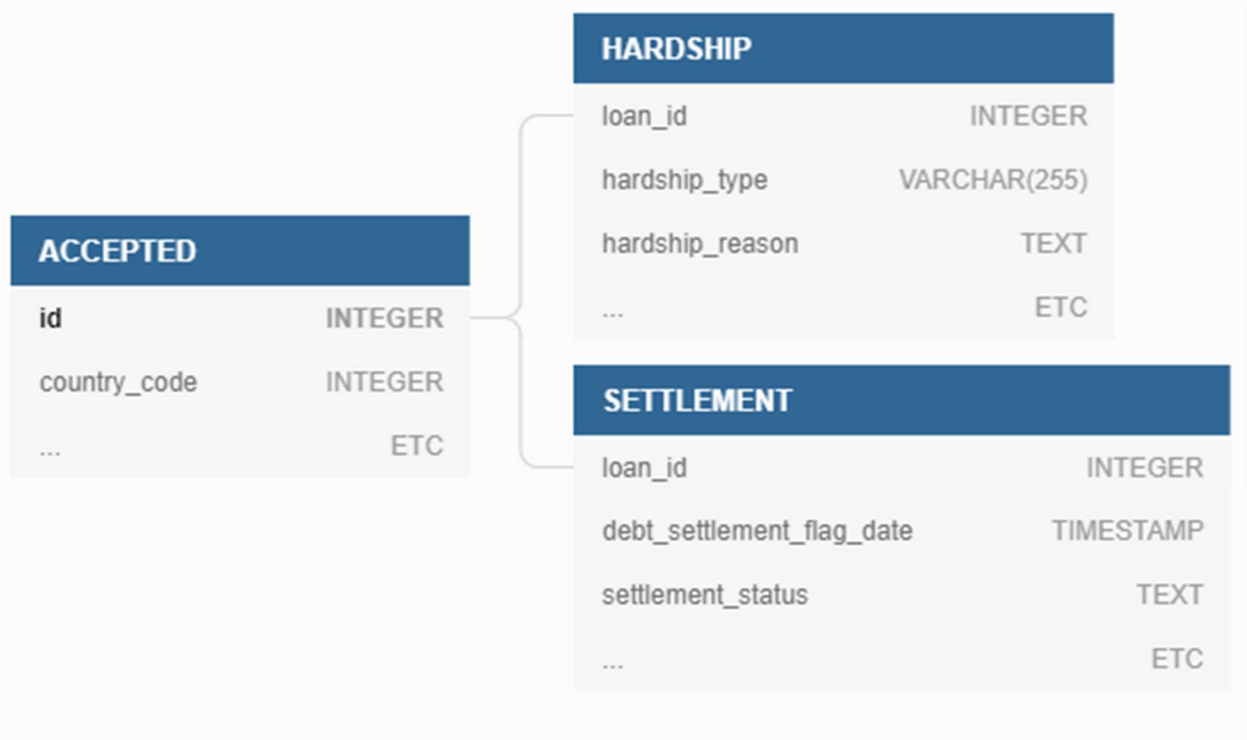
The Storage Engine

We will be using Snowflake for the storage engine as it is a fully managed cloud-based data warehouse which supports infinite scaling in both compute and storage and is highly optimized for analytics tasks. Although our Lending Club dataset is only ~400MB, Snowflake would be an excellent choice for larger systems to any size.

Though PostGres provides excellent storage capabilities, it does not scale for analytics workloads. Microsoft SQL Server is much better as it offers columnar database options but still requires the company to manage it themselves. Going with Spotify's take on Business Intelligence, that a company should focus as little as possible on supporting its infrastructure technologies. Snowflake fulfills all these requirements.

The Data Model

The data model follows a semi-normalized structure with a central fact table ("ACCEPTED") and dimension tables in order for the database to contain the entire dataset from Lending Club. The data is semi-normalized as most columns that would be analyzed are found in the "ACCEPTED" table and therefore don't require any joins. A view is included to reconstruct the original Lending Club table. The additional dimensions are features that may be analyzed on their own but not usually with the fact table.



I considered moving the member profile information to another dimension (using `member_id` as the foreign key) but ultimately decided against it. My reasoning twofold: Much of the profile information is very useful for analytics, and therefore should be accessed without joins. Additionally, unlike a user profile which is constantly updated with the most current information, the member information here is a snapshot in time as of the acceptance of the loan.

Automated Pipeline Between Data Lake and Warehouse

The entire ETL pipeline is constructed in Python from start to finish. I considered using SQL but decided on Python for its flexibility and CPython based table/matrix operations. Automatic periodic updates are achieved through the `schedule` module, set to update every day at `00:00`. Alternatively, this could be handled through CRON.

The module extracts the data from the application (in this case assumed to be PostGres) and loading into the Snowflake data warehouse through the `Pipe` class which is initialized with all the information for transforming, validating, and dimensioning the table. See README-Pipe.md for details on the class.

The order of operations is as follows:

1. Initialize the pipes that the module will schedule.
    - This includes requesting the most recent update date from the Snowflake database.
2. Schedule the table updates job.
3. Run table updates once upon initializing.
4. Pull new data from application.
5. Transform -> validate -> dimensionize.
6. Upload data to each table in Snowflake warehouse.
    1. This uses a `MERGE` on the table's `id` column to upsert by inserting if no match and updating if the `id` already exists in the database. This is important because we are generally only increasing the number of records and updating loans as details change.
7. Update the pipe's `last_updated` property to the `today` variable.

Discussion of Future System Improvements

Future improvements to the system should include full support for catching drop rows to notify the upstream application about invalid rows, which may allow the upstream application to attempt to recover some data.

We would also want to automate updating the pipeline when the application or target schema change which would greatly reduce the upkeep costs of the pipeline. An

alternative would be to leverage the expertise of a fully managed pipeline system such as Striim or Stitch Data to free internal personnel for business-oriented goals, similar to the benefits of using Snowflake in the first place.

We could leverage Snowflakes internal tools to support raw file storage, whether structured or semi-structured which allows for raw data to be stored in the warehouse for access if necessary. Also allows for ETL processes to happen on the warehouse rather than within scripts or connectors. The data warehouse also does not need to rely on the producing application or data lake for any structure in the data. Additionally, we could stage the raw or JSON files and then use pipes to ETL the data in near real-time.

Finally, should we decide to run the pipeline ourselves, or want to integrate additional services into the pipeline Kafka provides a robust method for updating in near real-time or even to simply allow the pipeline to know if data has been created or updated.