Adrien Protzel
3/17/15

Final Project

Problem Statement:

I have compiled records of my financial transactions from multiple bank accounts over the past few years. These records encompass various spending categories, but the data is not standardized and features inconsistent headers, columns, and vague descriptions. Despite these issues, the data includes crucial information such as amounts, descriptions, and dates. The aim of this analysis is to gain a comprehensive understanding of my spending habits. Specifically, I intend to identify spending patterns across different accounts and categories to understand where my money is going, detect anomalies or outliers in transaction amounts to identify any unusual spending behavior, analyze trends over time to uncover significant changes or patterns in my spending habits, assess the necessity of each spending category to determine if there are non-essential expenses that can be reduced or eliminated, and predict future spending patterns using historical data to develop a budget that ensures responsible spending and savings for future financial goals. By addressing these objectives, I hope to make informed decisions about my finances, optimize my spending, and achieve my long-term financial goals. Note, all data has been gathered from accounts by downloading .csv files but the data has been scrambled and randomized to keep privacy. More accounts have been added from my own to have more data diversity.

OSEMN Process:

Obtain:

Downloaded .csv files from various bank websites.

Scrub:

The data was cleaned by converting all files to .csv format, rewriting headers, merging them into a single file, and then standardizing the descriptions and categories into a final normalized dataset.

Explore:

The cleaned data was processed using tools such as Python, BigQuery, and Power BI for filtering and cleaning. Due to Dataprep's inability to export data correctly, it was not utilized in this process. The clean data was directly imported into BigQuery, where Spark computes the data in a parallelized fashion to retrieve specific information.

Model:

Power BI was used to create more engaging visuals, including charts, graphs, and dashboards, to illustrate spending patterns, trends over time, and category-wise expenditures. BigQuery and Spark were utilized to filter specific data using parallelization.

Interpret:
The visualizations created in Power BI were analyzed to draw meaningful conclusions about spending habits. High spending categories were identified, and the necessity of these expenses was assessed.

Description of data:

The data originates from various .csv files directly downloaded from bank websites. These files have inconsistent headers and differing structures, making them challenging to work with. Some of the data contains problematic lines due to commas that disrupt the rows, leading to broken entries. Additionally, the descriptions within the data are often unreadable and lack detail, making it difficult to understand the context of each transaction. Furthermore, the files include unnecessary or unusable data that needs to be filtered out.

The complexity of this data set is rated a 7, as indicated by the assigned scores. The lack of a standardized format (3 points) increases the difficulty of processing the data. The initial split across multiple files (1 point) necessitates additional steps to consolidate the information. The presence of strings with punctuation (1 point) further complicates data parsing, leading to broken entries that require manual processing. The data set comprises more than one type of related data (2 points). Overall, these factors contribute to a complex and time-consuming data preparation process.

.csv files

```
Description,,Summary Amt.
Beginning balance as of 09/14/2023,,"500.05"
Total credits,,"28,781.74"
Total debits,,"-28,681.79"
Ending balance as of 03/13/2025,,"600.00"

Date,Description,Amount,Running Bal.
09/14/2023,Beginning balance as of 09/14/2023,,"500.05"
09/18/2023,"Zelle payment from   Conf# kw8849x1d","1.00","501.05"
09/25/2023,"Zelle payment from   for "Kindle + Cover"; Conf#
a069qwzrj","210.00","711.05"
01/02/2024,"Zelle payment from   for "RV space rental for Jan. 2024."; Conf#
rh1j71cia","99.00","1,300.03"
01/02/2024,"Zelle payment from   for "Test of Zelle transfer. Hi AJ, let me know if
y"; Conf# owjcgtyu8","1.00","1,301.03"
01/05/2024,"Interest Earned","0.01","1,301.04"
```

```
"03/11/2025","-5.99","*","","APPLE.COM/BILL 866-712-7753 CA"
"03/08/2025","-34.99","*","","APPLE.COM/BILL 866-712-7753 CA"
"03/07/2025","-25.00","*","","VISIBLE 8663313527 CO"
"03/07/2025","-25.00","*","","VISIBLE 8663313527 CO"
"03/05/2025","-1.99","*","","APPLE.COM/BILL 866-712-7753 CA"
```

```
Details,Posting Date,Description,Amount,Type,Balance,Check or Slip #
DEBIT,03/12/2025,"Wealthfront      EDI PYMNTS F5458E284B1745  WEB ID:
4271967207",-806.52,MISC_DEBIT,1902.00,,
CREDIT,03/12/2025,"PAYPAL           TRANSFER                  PPD ID:
PAYPALSD11",9.00,MISC_CREDIT,2708.52,,
CHECK,03/11/2025,"CHECK 141  ",-107.00,CHECK_PAID,2699.52,141,
```

Data Wrangling Process:

The initial state of the data was characterized by inconsistent headers, varying structures, and problematic lines due to commas that disrupted the rows. The descriptions within the data were often unreadable and lacked detail, making it difficult to understand the context of each transaction. Additionally, the files included unnecessary or unusable data that needed to be filtered out.

Conversion to .csv Format: All files were converted to .csv format to ensure consistency.
Rewriting Headers: Headers were standardized across all files to create a uniform structure.
Merging Files: The individual files were merged into a single file to facilitate comprehensive analysis.
Standardizing Descriptions and Categories: Descriptions and categories were cleaned and standardized to improve readability and context.
Filtering Unnecessary Data: Unnecessary or unusable data was filtered out to focus on relevant information.
Tools Used
Python: Used for initial filtering and cleaning of the data.
BigQuery: Utilized for loading the cleaned data into a table with an appropriate schema.
Spark: Employed for parallelized computation to retrieve specific information from the dataset.

After the Python cleaning process, the data was standardized and formatted as follows:

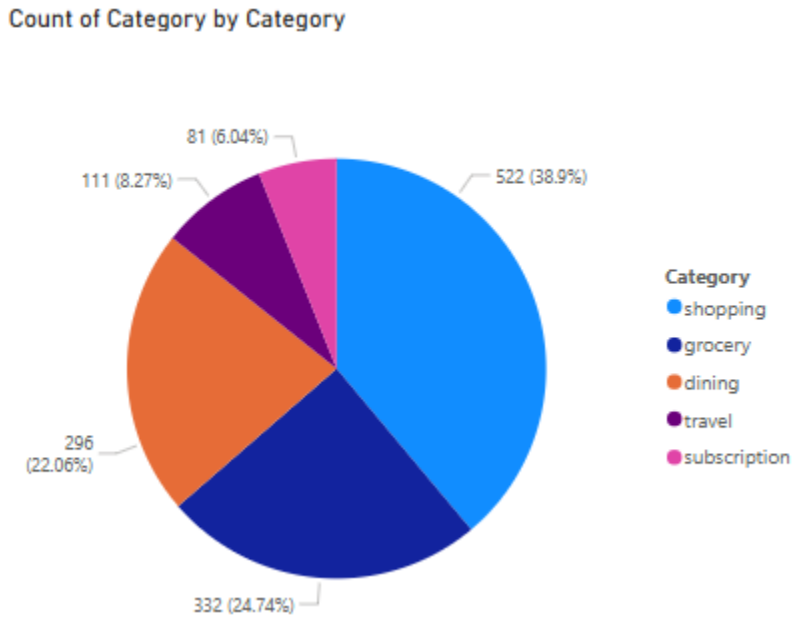
```
Year,Month,Date,Description,Category,Amount,Type,Bank,Card
2025,March,03/11/2025,apple,shopping,-5.99,Credit,Bilt,Bilt
2025,March,03/08/2025,apple,shopping,-34.99,Credit,Bilt,Bilt
2025,March,03/07/2025,visible,utility,-25.0,Credit,Bilt,Bilt
2025,March,03/07/2025,visible,utility,-25.0,Credit,Bilt,Bilt
2025,March,03/05/2025,apple,shopping,-1.99,Credit,Bilt,Bilt
2025,March,03/03/2025,pge,utility,-91.76,Credit,Bilt,Bilt
2025,March,03/03/2025,prose rent,utility,-2255.94,Credit,Bilt,Bilt
2025,February,02/25/2025,xfinity internet,utility,-43.0,Credit,Bilt,Bilt
2025,February,02/21/2025,apple,shopping,-9.99,Credit,Bilt,Bilt
2025,February,02/20/2025,apple,shopping,-12.99,Credit,Bilt,Bilt
2025,February,02/15/2025,transfer,transfer,2458.52,Credit,Bilt,Bilt
2025,February,02/15/2025,apple,shopping,-4.99,Credit,Bilt,Bilt
2025,February,02/11/2025,apple,shopping,-5.99,Credit,Bilt,Bilt
2025,February,02/07/2025,visible,utility,-25.0,Credit,Bilt,Bilt
2025,February,02/07/2025,visible,utility,-25.0,Credit,Bilt,Bilt
```

Data Analysis Questions:

Question 1: What are the top 5 categories of expenses based on the category field?

Description: This question aims to identify the top 5 categories of expenses by analyzing the category field in the transaction data.

Process:
Data Cleaning: Standardize headers and clean the data for consistency.
Filtering: Extract categories and amounts from the data.
Aggregation: Sum the transaction counts and amounts for each category.
Ranking: Identify the top 5 categories based on transaction counts.
Visualization: Create a pie chart to represent the data.



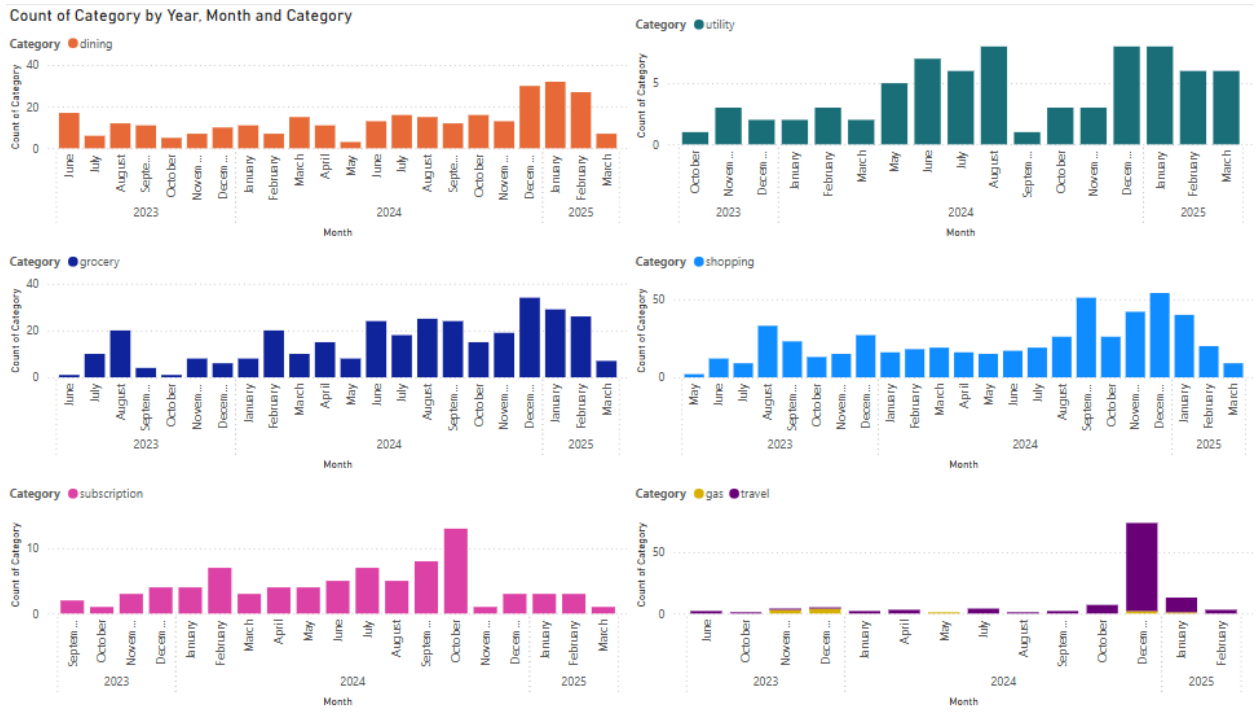
Pie Graph

Conclusion: I used a pie chart to display each category of spending, excluding "Transfer" (balance transfers between accounts) and "Expense" (miscellaneous expenses that do not fit typical categories) to avoid skewing the data. The pie chart reveals that "Shopping" is the largest category, accounting for 38.9% of transactions. The other categories are "Grocery" (24.74%), "Dining" (22.06%), "Travel" (8.27%), and "Subscriptions" (6.04%). This analysis focuses on the count of transactions per category rather than the total amount spent, highlighting the frequency of each category in the data.

Question 2: What is the correlation between transaction types (category) and the frequency of transactions (over time)?

Description: This question aims to determine the correlation between different transaction categories and the frequency of transactions over time.

Process:
Data Cleaning: Standardize headers and clean the data for consistency.
Filtering: Extract data based on date and category.
Correlation Analysis: Use statistical methods to analyze the correlation between transaction categories and their frequencies.
Visualization: Create small multiples stacked bar graphs to depict each category separately.



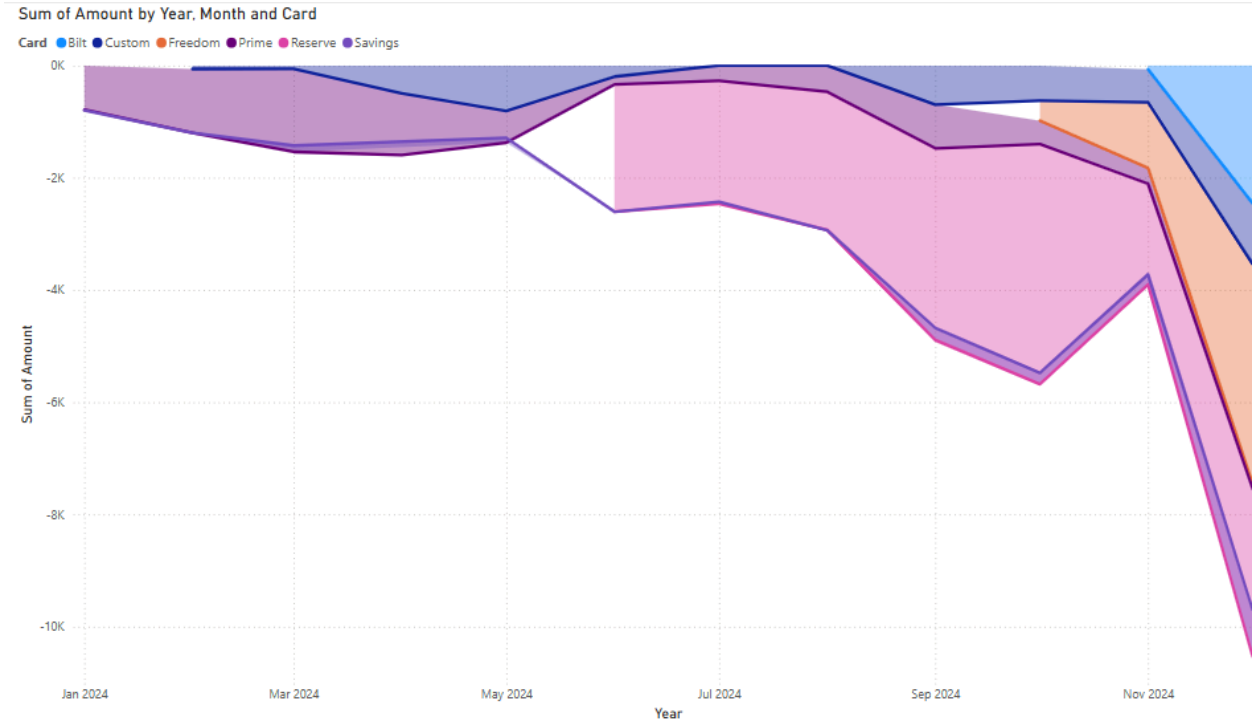
Small Multiples Stacked Bar Graph

Conclusion: I used small multiples stacked bar graphs to depict each category separately. The graphs show a trend of increased spending in the later months, starting in October. The "Gas" and "Travel" categories were combined because "Gas" transactions were too infrequent to compare directly with other categories. This visualization helps to identify patterns and correlations between transaction types and their frequencies over time. Additionally, "Expenses," "Transfer," and "Utilities" were excluded due to their consistency, randomness, or lack of informativeness in the data.

Question 3: How does the spending pattern vary across different accounts over the last year?

Description: This question aims to analyze how spending patterns differ across various bank accounts (cards) over the past year.

Process:
Data Cleaning: Standardize headers and clean the data for consistency.
Filtering: Extract data based on account, amount, and date.
Monthly Aggregation: Group transactions by month and within each month, group by account and sum their amounts.
Comparison: Compare the monthly spending patterns across different accounts.
Visualization: Create a stacked area chart to display the data.

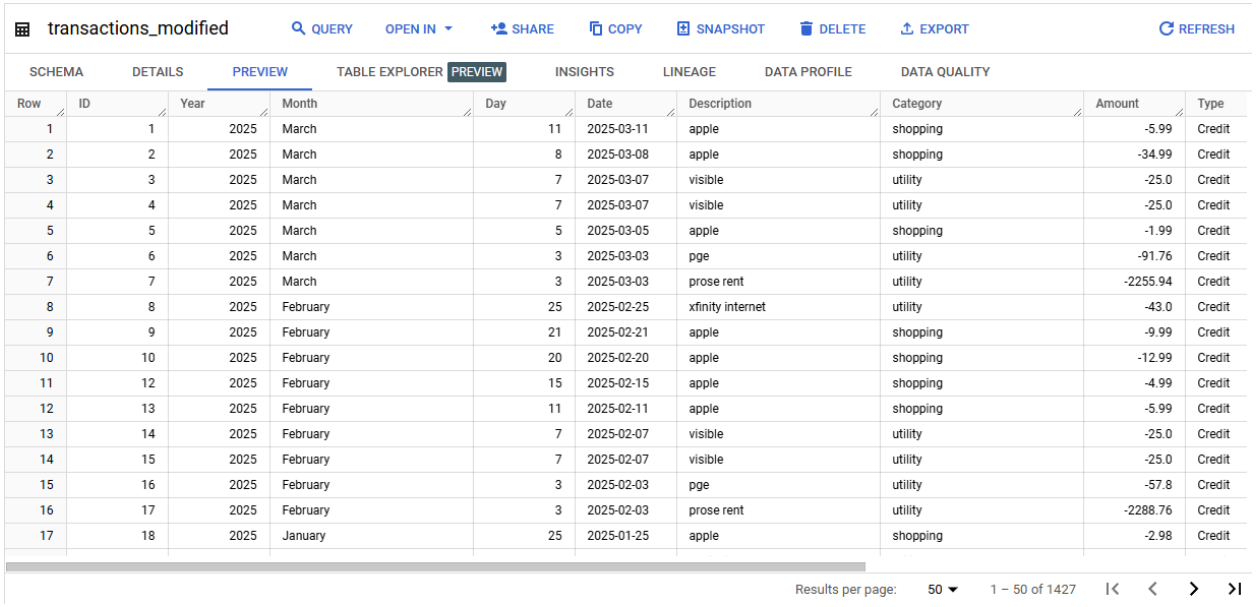


Stacked Area Chart

Conclusion: I used a stacked area chart to display the amounts used for each card. "Transfer" and "Expenses" were excluded due to their potential to skew the data and their unreliability. This analysis covers the year 2024, as it provides a complete year of data. The data shows that most expenses occurred in December, with a smaller peak in October and a dip in November. The increase in expenses in the later months correlates with the trends observed in the previous question's graphs. However, the decrease in November is unusual but is also slightly reflected in the "Subscriptions" category from Question 2.

Data Loading and Processing:

I needed to create a new pandas Python script to modify data and add an ID column. As a side effect, I added a Day column just because I could. I manually loaded the data into BigQuery and created a new dataset called transaction_data. I then created a new table with auto-detect schema named transactions. I executed a DELETE statement to remove rows from transaction-pipeline.transaction_data.transactions where the category was either 'transfer' or 'expense'. I created a bucket named cs512-protzela-transaction-data and added clean.csv and pyspark.py to it. I also created a VM instance called transaction-vm in the us-west1-a zone. Following that, I created a new cluster named cluster-2779 with 2 workers, both workers and master nodes having 2 cores, located in us-east1-b.



transactions_modified

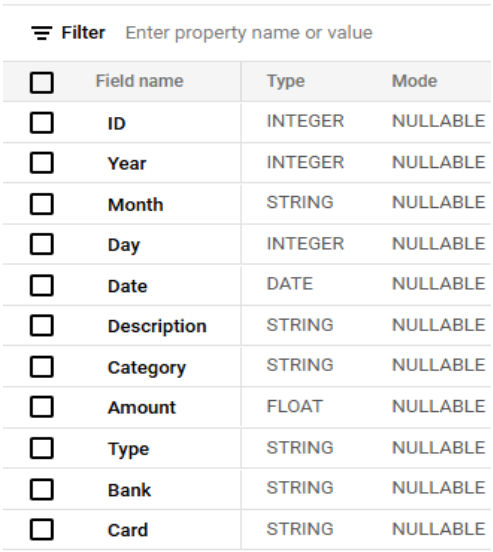
QUERY | OPEN IN | SHARE | COPY | SNAPSHOT | DELETE | EXPORT | REFRESH

SCHEMA | DETAILS | PREVIEW | TABLE EXPLORER PREVIEW | INSIGHTS | LINEAGE | DATA PROFILE | DATA QUALITY

Row	ID	Year	Month	Day	Date	Description	Category	Amount	Type
1	1	2025	March	11	2025-03-11	apple	shopping	-5.99	Credit
2	2	2025	March	8	2025-03-08	apple	shopping	-34.99	Credit
3	3	2025	March	7	2025-03-07	visible	utility	-25.0	Credit
4	4	2025	March	7	2025-03-07	visible	utility	-25.0	Credit
5	5	2025	March	5	2025-03-05	apple	shopping	-1.99	Credit
6	6	2025	March	3	2025-03-03	pge	utility	-91.76	Credit
7	7	2025	March	3	2025-03-03	prose rent	utility	-2255.94	Credit
8	8	2025	February	25	2025-02-25	xfinity internet	utility	-43.0	Credit
9	9	2025	February	21	2025-02-21	apple	shopping	-9.99	Credit
10	10	2025	February	20	2025-02-20	apple	shopping	-12.99	Credit
11	12	2025	February	15	2025-02-15	apple	shopping	-4.99	Credit
12	13	2025	February	11	2025-02-11	apple	shopping	-5.99	Credit
13	14	2025	February	7	2025-02-07	visible	utility	-25.0	Credit
14	15	2025	February	7	2025-02-07	visible	utility	-25.0	Credit
15	16	2025	February	3	2025-02-03	pge	utility	-57.8	Credit
16	17	2025	February	3	2025-02-03	prose rent	utility	-2288.76	Credit
17	18	2025	January	25	2025-01-25	apple	shopping	-2.98	Credit

Results per page: 50 | 1 – 50 of 1427

Screenshot ot the data loaded into BigQuery, with row count



Filter Enter property name or value

Field name	Type	Mode
ID	INTEGER	NULLABLE
Year	INTEGER	NULLABLE
Month	STRING	NULLABLE
Day	INTEGER	NULLABLE
Date	DATE	NULLABLE
Description	STRING	NULLABLE
Category	STRING	NULLABLE
Amount	FLOAT	NULLABLE
Type	STRING	NULLABLE
Bank	STRING	NULLABLE
Card	STRING	NULLABLE

Screenshot of the correct schema types

Results and Discussion:

I wrote a new PySpark Python script and created a new job with the project ID transaction-pipeline, dataset ID transaction_data, and table ID transactions_modified. The PySpark URI was gs://cs512-protzela-transaction-data/solution.py, and I used the JAR file gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar. The expected output of the PySpark code was to give the top 5 months with the most expenses.

However, the PySpark program was unable to run due to a type error, and I couldn't figure it out. I made two versions of the code to output, but neither worked. I tried type converting and printing types and just printing records without doing any work, but nothing seemed to change the output. The error message was pyspark.errors.exceptions.base.PySparkTypeError: [CANNOT_ACCEPT_OBJECT_IN_TYPE] IntegerType() cannot accept object 2025 in type str.