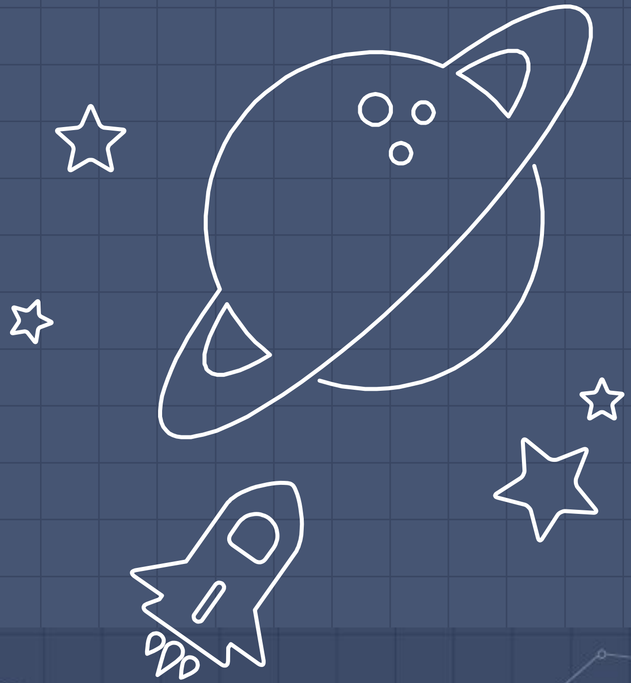
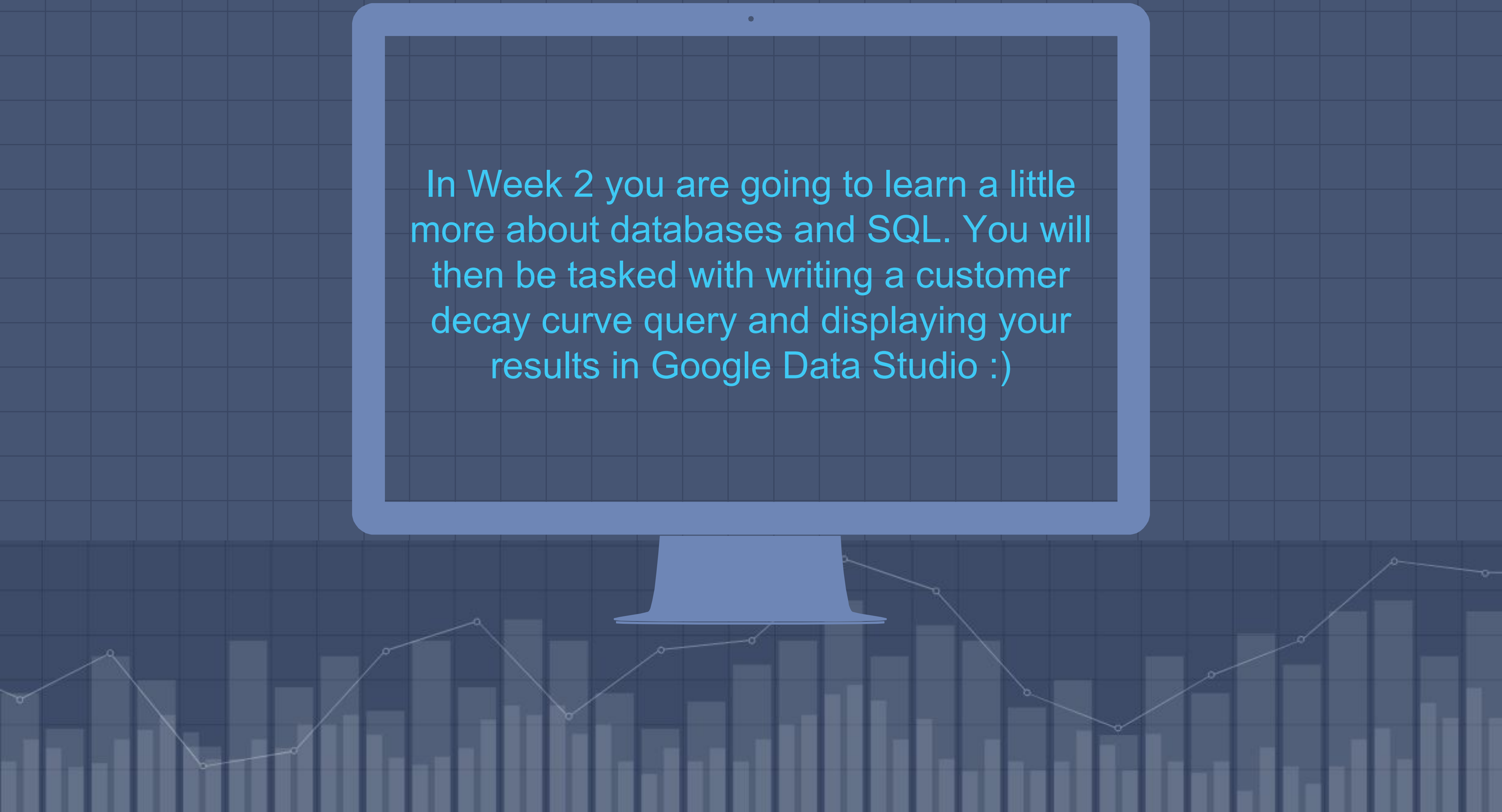
arise:

Data Science Fundamentals

Week 2





In Week 2 you are going to learn a little more about databases and SQL. You will then be tasked with writing a customer decay curve query and displaying your results in Google Data Studio :)

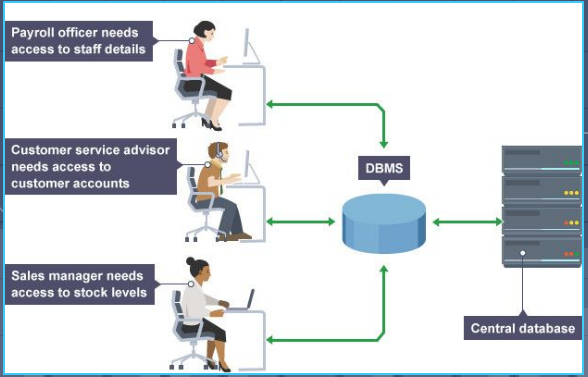
Database Management Systems

Database management systems (DBMS) are software packages that make it possible for users (in Paylater's case our employees) to access, create, read, update and delete the data stored in a database.

DBMS ensure :

1. Data <u>concurrency</u> (many users can access the same data at any given time)
2. Data <u>security</u> (restrict access to all or certain parts of the database)
3. Data <u>integrity</u> (the consistency and quality of data)

The image to the right graphically displays how users interact with a DBMS and how a DBMS interacts with an actual database.


Payroll officer needs access to staff details
Customer service advisor needs access to customer accounts
Sales manager needs access to stock levels
DBMS
Central database

Relational Databases

There are two types of databases, relational and non-relational.

Relational databases are structured in a way that data is stored in seperate tables. Each table can be thought of as an excel spreadsheet where the rows are the data entries and the columns are the features that occur in that table. Each row in a table can be identified by a unique primary key (primary feature). Data across different tables can be merged by joining tables. Tables are joined on foreign keys. A foreign key is a key within a table that is the primary key of another table.

Structured Querying Language (SQL) is used to interact with data stored in a relational database. For a better understanding of relational databases please watch this video.

When designing a relational database, the tables within the database need to be normalised. This means that data repetition must be eliminated and data dependencies within tables must be logical. This is further explained is this video. Relational databases are designed using Entity Relationship (ER) Diagrams. These diagrams show entities that are stored within a database and the relationships between the entities. Please watch part1 and part2 of this video for more information on ER diagrams.

Non-Relational Databases

Non-Relational databases, sometimes referred to as noSQL databases, do not store data in tabular format. The most common types of non-relational databases are key-value, graph, column and document. Each store data in a different way and are best suited to particular cases. Please read this document to better understand the different types of non-relational databases and their use cases.

Non-Relational databases have many advantages over relational databases. Non-Relational databases were created to handle the increasing volume of data that companies are posed with handling. They allow users to stored data as serialized JSON objects which decreases the memory requirements necessary to handle data. Within non-relational databases, records stored in the same collection can have different data types. This allows users to have more freedom when designing databases.

The most common non-relational database is MongoDB. At Paylater we use Google Bigquery as our relational database, which uses Standard SQL as its querying language. We use MongoDB as our backend database.

We are going to spend time this week working with SQL and learning how to use Google Bigquery.

Google Bigquery

Please follow this link to gain access to Google Bigquery, once the linked has opened please do the following:

- Click on `GO TO CONSOLE`, this will take you to the Google Bigquery user interface. You may be asked for credentials, if so please log in with the email address that you used when you applied to the program.
- In the left tab under `Resources` you should see the project titled `propane-highway-202915`. Please expand that dataset.

You should be able to see two tables, one titled `Loans` and the other titled `BillPayments`. These two tables hold all loan disbursements to Paylater clients and bill payments made by Paylater clients in 2018 respectively.

We are now going to learn how to perform selections, joins and aggregations within SQL.

Selections

Before we start learning SQL, please save this link to the Standard SQL functions and operators. This is an incredibly useful resource and will definitely be needed for your challenge!

`Select`, this is the most basic SQL command. This allows you to pull data within a table.

The following line of code will pull all features from the Loans table.

```
SELECT * FROM `propane-highway-202915.arise.Loans`
```

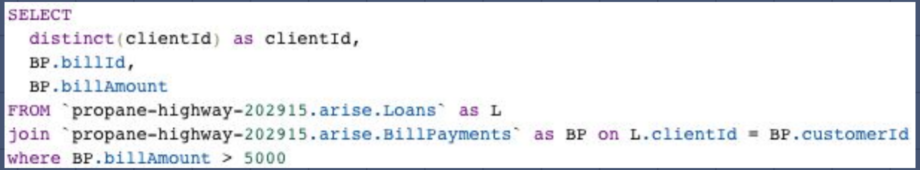
If we replace the * with any of the feature names in the Loans table only those features will be selected. For example, the following line selects on the loanId's and clientId's attached to each loan in the Loans table.

```
SELECT loanId, clientId FROM `propane-highway-202915.arise.Loans`
```

Joins and the `where` clause

We are now going to learn how to join data from two tables. In this example we are going to use a direct join but please read this article explaining the other types of joins. A direct join allows users to select data in the table associated to the `join` where the primary key of those particular rows exist as a foreign key in the tables associated to the `from`.

Here is an example:



```
SELECT
  distinct(clientId) as clientId,
  BP.billId,
  BP.billAmount
FROM `propane-highway-202915.arise.Loans` as L
join `propane-highway-202915.arise.BillPayments` as BP on L.clientId = BP.customerId
where BP.billAmount > 5000
```

In the above example we only extract the clientId's, billId's and bill amounts of bill payments where the customerId of attached to the bill payment exists in the Loans table.

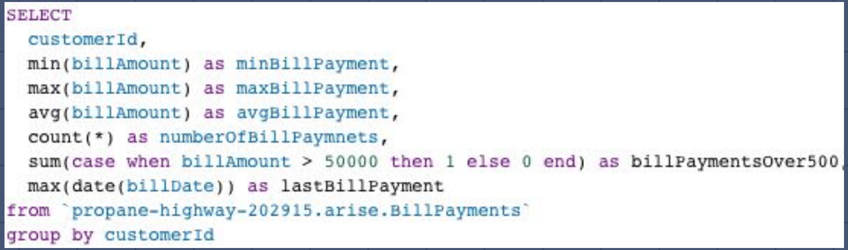
A few key points, you can see after the tables are initialised there is an as followed by a letter of two. For the Loans table it is an `L` and for the BillPayments table it is a `BP`. These are called aliases and they allow you to directly state where a feature comes from in a select statement. This is necessary when you have two features called the same name in two different tables.

Finally the `where` clause at the end of the statement. This is used to exclude unwanted rows from a query result. In the example query we exclude any pill payments that are less than 5000 Kobo.

The `group by` and Aggregations

The group by clause allows you to perform aggregations of certain features with respect to another feature. Standard SQL functions and operators documentation displays what aggregations are possible within Bigquery, but common ones are min and max values, average value, counts of unique identifiers and the sum of particular cases.

Here is an example of a group by clause and aggregations.



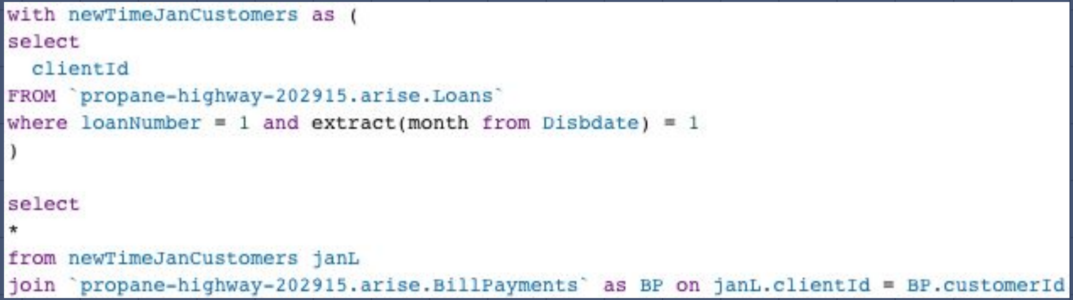
```
SELECT
  customerId,
  min(billAmount) as minBillPayment,
  max(billAmount) as maxBillPayment,
  avg(billAmount) as avgBillPayment,
  count(*) as numberOfBillPaymnets,
  sum(case when billAmount > 50000 then 1 else 0 end) as billPaymentsOver500,
  max(date(billDate)) as lastBillPayment
from `propane-highway-202915.arise.BillPayments`
group by customerId
```

Let's break down the above example. In the above query we are grouping by customerId from the BillPayments table. What does that mean? It means that for each customer in the table we perform the above calculations. The first three calculations return each customer's minimum, maximum and average bill payments. The count(*) function returns the number of accounts of the group by clause, in the example this is the number of times each customerId appears in the table. You can think of it as a total count. The 5th aggregation is an example of a case when, which is SQL's if/else. The statement sums the number of times that each customer had a bill payment greater than 5000 Kobo. Note that you could have exchanged 1 with the bill payment amount to get each customer's total value of the bill payments that are more than 5000 Kobo in value. The final aggregation is an interesting one, please note the date() function in the final aggregation. This function converts timestamps to dates, this will be needed for your assignment. The final aggregation finds the lastest bill payment made by each customer in the table.

Temporary Tables

Temporary tables are key to writing readable and understandable queries. They allow users to select specific data and then be able to access that data at a later point via joins.

Here is an example:



```
with newTimeJanCustomers as (
select
  clientId
FROM `propane-highway-202915.arise.Loans`
where loanNumber = 1 and extract(month from Disbdate) = 1
)

select
*
from newTimeJanCustomers janL
join `propane-highway-202915.arise.BillPayments` as BP on janL.clientId = BP.customerId
```

In the query above the newTimeJanCustomers temporary table is initialised at the top of the query. This query only extracts clients on their first loan (new clients) that got that loan in January. Then in the later selection, those clients are joined to the BillPayments table. That means that the BillPayments of only clients that were first time clients in January 2018 are selected.

Your Challenge

You have been tasked with creating two queries.

The first is to write a query that first extract only clients that received their first loan in January 2018. You then have to find the number of those clients that received/took out a loan in the following months of 2018. If a client took more than one loan in a month, they should only be counted once. This is a measure of client retention. Please display your results in Google Data Studio (GDS), please follow this link to get to GDS.

After you have written your query you can use it as a data resource in a GDS report by following these steps:

- First open a blank GDS report. You can then select `CREATE NEW DATA RESOURCE` in the bottom right corner of your blank report.
- The Google Connectors page then opens. Select the BigQuery connector.
- In the left tab that then pops up, select customer query. You then select OneFI Academy Google Cloud and copy and paste your query into the query field. Make sure `Use Legacy SQL` is unticked.
- Click `CONNECT`. All fields of your query then pop up, check to make sure their types are correct. When they are, click `ADD TO REPORT`.
-

You now need to display your results from your query. This link should help you learn how to best use Data Studio. You can skip the data introduction of this video.

Your Challenge

Your second task is to produce a very similar report but for clients that had a bill payment in March 2018. You have to show the number of those clients that made at least one bill payment in the remaining months of 2018. On top of this, you have to show whether or not the bill payment clients have a loan within 2018 or not. Make sure that the second part is displayed in your results (hint: two category bar plot).

You can add your second query to the same report as another data source, it can be done in the same manner as the first query.

Please display your findings on a new page within your GDS report (you will have a page for loans and a page for bill payments).

Please submit your two queries and the link to your GDS report here. Please rename your report as the email address that you used when you applied to the arise program. Remember to change the sharing permissions of your GDS report to anyone with the link can view!

Your week 2 answers must be submitted by midnight on Sunday the 3rd of March.

Next Week

Next week you are going to learn about unsupervised learning techniques and how to build them in Python!!!