statsbot

Estimating LTV using SQL

WHAT IS LTV?

Customer lifetime value, or LTV, is the **amount of money that a customer will spend with your business in their "lifetime,"** or at least, in the portion of it that they spend in a relationship with you. It's an important indicator of how much you can spend on acquiring new customers.

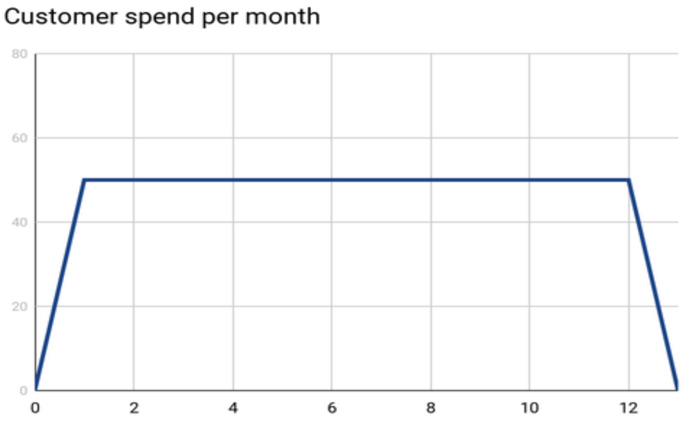
Imagine, your customer acquisition cost (CAC) is $150, and LTV is $600. You would be able to increase the budget to get more people and grow your business. The balance between CAC and LTV allows you to check any business for market survival.

ESTIMATING LTV

In order to understand how to estimate LTV, it is useful to first think about evaluating a customer's lifetime value at the end of their relationship with us. So, say a customer stays with us for 12 months, and spends $50 per month.

The revenue that they generated for our business over their lifetime is then $50*12 = $600. Simple!

We can consider the basic definition of LTV as a sum of payments from a specific user.



This same principle applies to the group. If we want to see the average LTV for the group we can look at total spend divided by the number of customers. When we're talking about estimating LTV for the group, or predictive LTV, we need to take into account how long customers stay with us. To get that, we actually look at it "backwards": we look at how many customers we lose over time, or the churn rate.

HOW DO YOU CALCULATE LTV FOR SAAS?

At a group level, the basic formula for estimating LTV is this: $LTV = \frac{ARPU}{Churn\ rate}$

Where ARPU is average monthly recurring revenue per user and the churn rate is the rate at which we are losing customers (so the inverse of retention).

This basic formula can be obtained from assumption:
Next Month Revenue = (Current Month Revenue) * (1 - Churn Rate)

Note: When we're estimating LTV for SaaS we can neglect Gross Margin, because costs are minor and don't affect the accuracy of a result. But when we calculate predictive LTV for ecommerce later in this article, we'll include COGS in our formula.

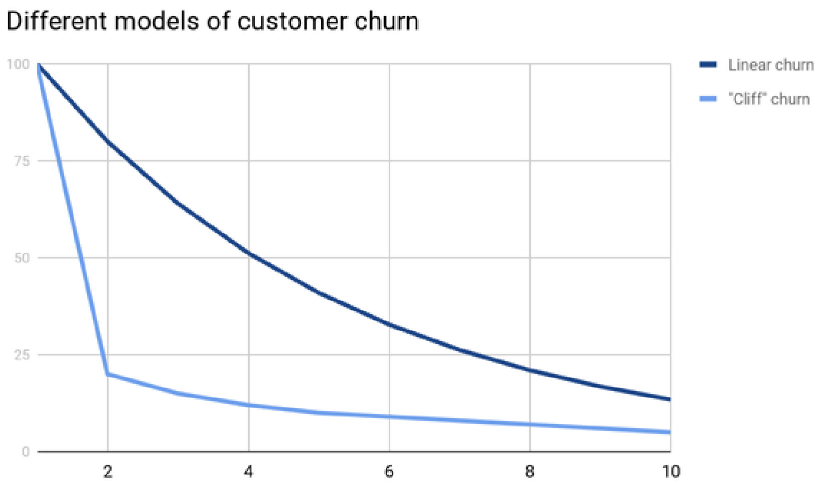
The **main limitation of the LTV formula** above is that it assumes that churn is linear over time, as in: we are as likely to lose a customer between the first month of membership of our service and the second, as we are to lose them much later on. Going deeper into the nature of predictive LTV, we can say that it's a sum of a geometric series, and linear churn doesn't look like a straight line (as is shown in many articles about LTV).

In fact, we know that **linear churn is increasingly not the case**: in a flexible subscription model, we lose many people at the very beginning, when they are "testing out" a service, but once they have been with us for a long time, they are less likely to leave.

Ultimately, it depends on the type of contract that exists between customers and the business: for example, annual renewals, where churn is more linear, will result in LTV that is very close to the formula above.

Services which do not have any contracts may lose a high percentage of their new customers, but then churn may slow down.

If the LTV of the group is the area under the line, we can very clearly see that the rate at which we lose customers **will impact our LTV estimates very significantly**. So we will need to take this into account when we are making our calculations. For a first estimate of LTV, however, it makes sense to go with the simplest formula. After that, we will add levels of complexity.



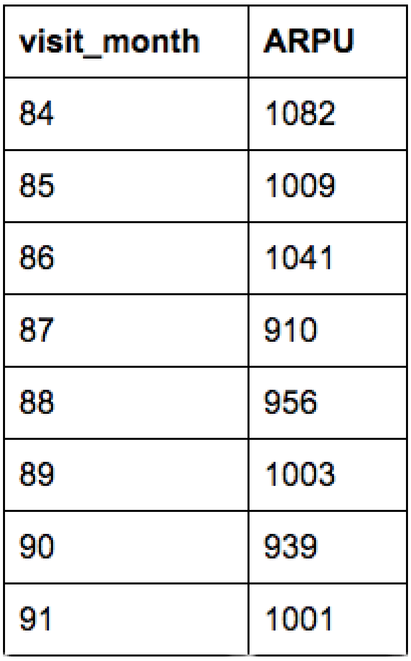
EXTRACTING ARPU AND CHURN USING SQL

In order to make the most basic estimate of LTV, we need to look at our transaction history. Then, we can establish the average revenue per customer as well as the churn rate over the period that we are looking at. For simplicity, we're going to look at the last year, and use a scenario when you have **only monthly payments and no proration**.

You can calculate ARPU in 2 steps:

```
month_ARPU AS
(SELECT visit_month,
    Avg(revenue) AS ARPU
FROM
    (SELECT
         Cust_id,
         Datediff(MONTH, ‘2010-01-01’, transaction_date) AS visit_month,
         Sum(transaction_size) AS revenue
     FROM    transactions
     WHERE  transaction_date > Dateadd(‘year’, -1, CURRENT_DATE)
     GROUP BY
         1,
         2)
GROUP BY 1)
```

The results will look like this:



visit_month	ARPU
84	1082
85	1009
86	1041
87	910
88	956
89	1003
90	939
91	1001

You can then either do a simple average in whatever software you are using, or just add an average to the code above:

```
SELECT Avg(ARPU) AS ARPU
FROM  month_ARPU
```

In the case above, that would give us an **average monthly spend of $987.33**.

Calculating churn rate is a bit more complicated, as we need the percentage of people not returning from one month to the next, taking each group of customers according to the month of their first visit, and then checking if they came back or not in the following month.

> The problem is always that, in a transactional database, we have customers' visits on separate lines, rather than all on the same line. The way to fix that problem is to join the transactional database to itself, so that we can see a customer's behavior on one single line.

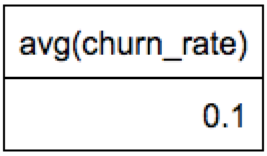
In order to isolate those who churned, we take the visits from month 1, and left join the visits from month 2 on the cust_id. The lines where visits from month 2 have a cust_id that is null are the ones where the customer has not returned.

Let's say that we average this over the year:

```
WITH monthly_visits AS
(SELECT
    DISTINCT
    Datediff(month, '2010-01-01', transaction_date) AS visit_month,
    cust_id
FROM transactions
WHERE
    transaction_date > dateadd('year', -1, current_date)),
(SELECT
    avg(churn_rate)
FROM
    (SELECT
        current_month,
        Count(CASE
            WHEN cust_type='churn' THEN 1
            ELSE NULL
        END)/count(cust_id) AS churn_rate
    FROM
```

```
        (SELECT
            past_month.visit_month + interval ‘1 month’ AS
            current_month,
            past_month.cust_id,
            CASE
                WHEN this_month.cust_id IS NULL THEN 'churn'
                ELSE 'retained'
            END AS cust_type
         FROM
            monthly_visits past_month
            LEFT JOIN monthly_visits this_month ON
                this_month.cust_id=past_month.cust_id
                AND this_month.visit_month=past_month.visit_month +
            interval ‘1 month’
         )data
     GROUP BY 1)
)
```

Say this gives us a result of 0.1, just for simplicity.



avg(churn_rate)
0.1

It is a simple calculation, then, to estimate LTV: we have monthly ARPU and monthly churn, so we just divide one by the other!

$987.33/0.1 = $9873.3

As stated earlier, there are limits to this formula, mostly because it makes a series of assumptions that may not hold in the real world. The main one is that retention and churn rates are stable both across cohorts and across time.

Stability across cohorts implies that early adopters of your service act in similar ways to late adopters, while stability across time implies that customers’ likelihood of churning out is the same at the beginning of their relationship with you as it is, for example, 2 years in.

Depending on how close to the truth these assumptions are, you may need to revise your LTV estimate downwards.

*Check out how **30% more Giphy** employees got involved with analytics after deploying Statsbot

CALCULATING LTV FOR COHORTS

Over time, and as our set of customers becomes larger, we may want to gain more precision in our estimate of LTV. One way to do this is to split our clients into groups, or cohorts.

> A good reason to operate using cohorts is that acquisition date can often reflect something deeper than just timing: early adopters of a product or a service are frequently more engaged and higher spending, while as the product gains traction in the market, larger number of customers may start using it but each spend less.

So then, we need to categorize our customers according to the first time that they purchased from us, both for the calculation of ARPU and retention. Calculating retention per cohort is important as well, because if we suppose that there is something qualitatively different between the groups in terms of their engagement and their spending, it follows that their retention rates will be different.

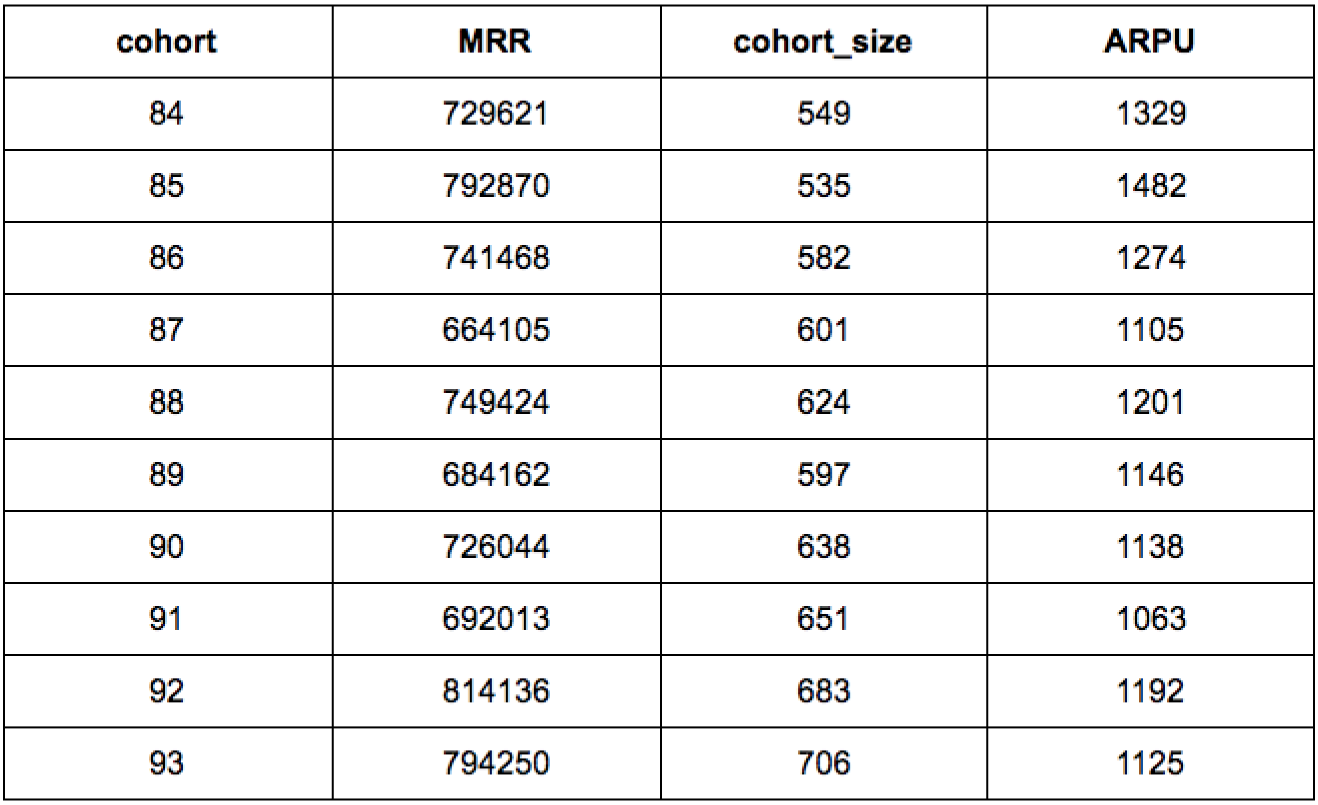
Let’s group customers according to the **month of their first visit**, for simplicity. You can go down to the week, or even the day, if that makes sense for your business: all you need to do is replace “month” by “week” or “day” in the cohort definition.

We can then join the cohort identifier to the query that we wrote above.

```
WITH cohorts AS
(SELECT
     cust_id,
     min(visit_month) AS cohort
FROM
     (SELECT
          cust_id,
          Datediff(month, ‘2000-01-01’, transaction_date) AS visit_month
     FROM   transactions) data
     GROUP BY 1),

cohort_ARPU AS
(SELECT
    cohort,
    Sum(revenue)/Count(DISTINCT visit_month) AS MRR,
    Count(DISTINCT cust_id) AS cohort_size,
    (Sum(revenue)/Count(DISTINCT visit_month))/Count(DISTINCT cust_id) AS ARPU
FROM
    (SELECT
         cust_id,
         cohort,
         Datediff(month, ‘2010-01-01’, transaction_date) AS visit_month,
         Sum(transaction_size) AS revenue
    FROM
         transactions
         LEFT JOIN cohorts ON
              cohorts.cust_id=transactions.cust_id
    WHERE transaction_date > Dateadd(‘year’, -1, CURRENT_DATE)
    GROUP BY
              1,
              2,
              3)
GROUP BY 1)
```

This will give us, by month of first visit, the MRR, the number of customers included in the cohort, and the ARPU. It’s not really necessary to have the MRR or the size of the cohort (or at least, it’s not necessary for calculating LTV), but these can be interesting and useful pieces of information nonetheless.



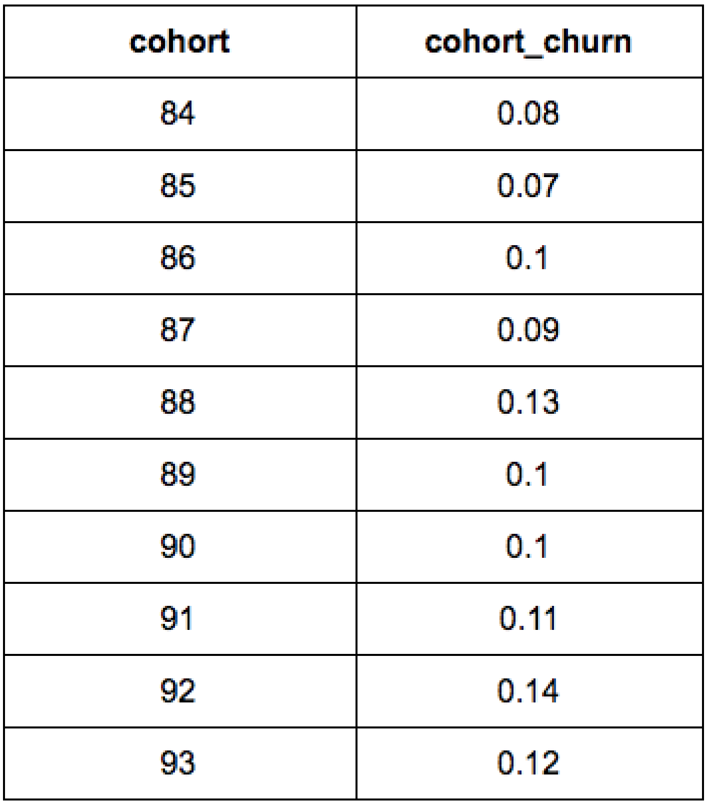
cohort	MRR	cohort_size	ARPU
84	729621	549	1329
85	792870	535	1482
86	741468	582	1274
87	664105	601	1105
88	749424	624	1201
89	684162	597	1146
90	726044	638	1138
91	692013	651	1063
92	814136	683	1192
93	794250	706	1125

Then, we need to calculate the retention rate per cohort. Again, it is fairly simple to just add the cohort as an identifier using a simple left join in the code from before.

```
WITH monthly_visits AS
(SELECT
    DISTINCT
    Datediff(month, ‘2010-01-01’, transaction_date) AS visit_month,
    cust_id
FROM      transactions
WHERE   transaction_date > Dateadd(‘year’, -1, CURRENT_DATE)),

Cohort_churn AS
(SELECT
    cohort,
    avg(churn_rate) AS churn
FROM
    (SELECT
        current_month,
        cohort,
        Count(CASE
            WHEN cust_type='churn' THEN 1
            ELSE NULL
         END)/count(cust_id) AS churn_rate
    FROM
        (SELECT
            past_month.visit_month + interval ‘1 month’ AS current_month,
            past_month.cust_id,
            CASE WHEN this_month.cust_id IS NULL THEN 'churn'
               ELSE 'retained'
            END AS cust_type
        FROM
                monthly_visits past_month
                LEFT JOIN monthly_visits this_month ON
                    this_month.cust_id=past_month.cust_id
                    AND this_month.visit_month=past_month.visit_month + interval ‘1
month’
        )past_month
        LEFT JOIN cohorts ON
            cohorts.cust_id=past_month.cust_id
        GROUP BY
                    1,
                    2)
GROUP BY 1)
```

This will give us something that looks like this:

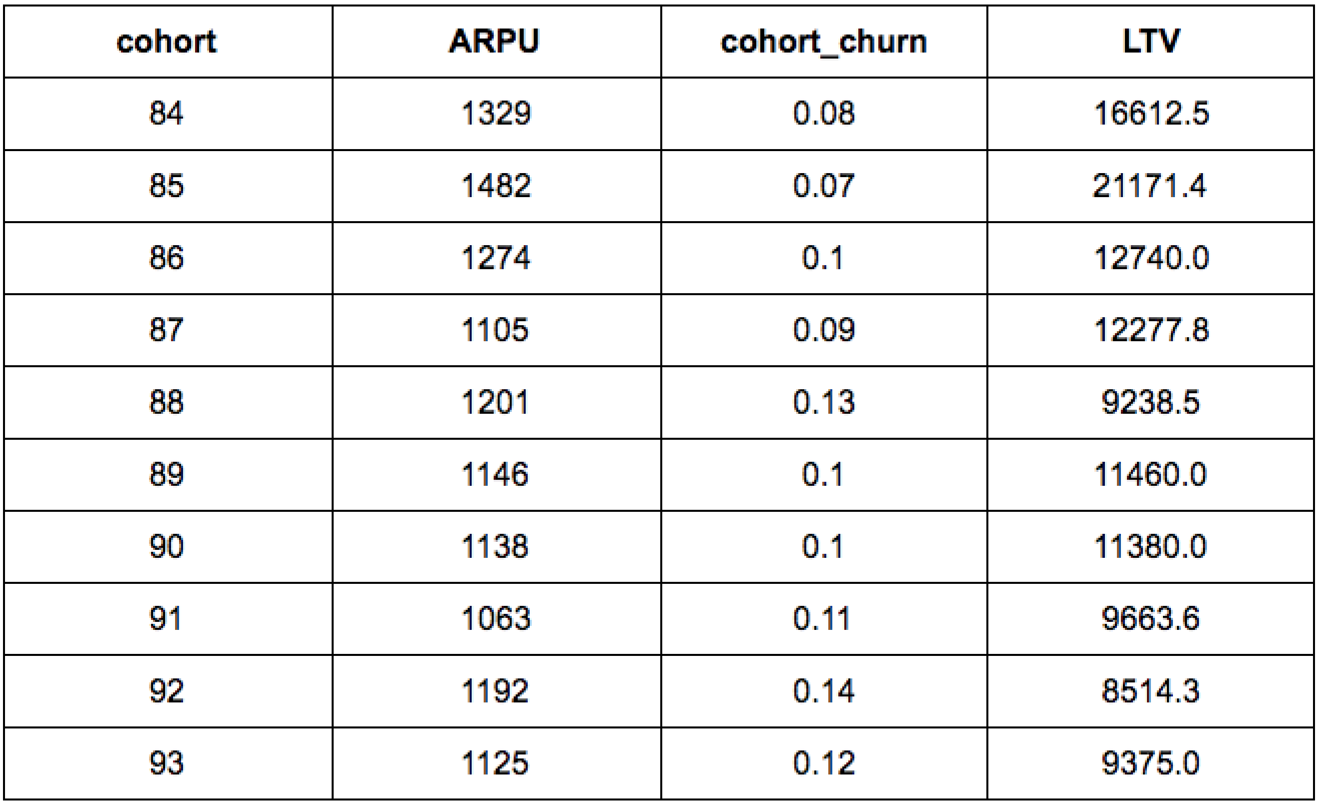


cohort	cohort_churn
84	0.08
85	0.07
86	0.1
87	0.09
88	0.13
89	0.1
90	0.1
91	0.11
92	0.14
93	0.12

To make life easier for ourselves, we can then do a left join to get ARPU, churn rate, and LTV in the same table.

```
SELECT
     cohort_arpu.cohort,
     ARPU,
     churn,
    ARPU/churn AS LTV
FROM
    cohort_arpu
    LEFT JOIN cohort_churn ON
         cohort_arpu.cohort = cohort_churn.cohort
```

Now, for each cohort, we have the **average revenue per account, the churn rate, and estimated the LTV per customer**.



cohort	ARPU	cohort_churn	LTV
84	1329	0.08	16612.5
85	1482	0.07	21171.4
86	1274	0.1	12740.0
87	1105	0.09	12277.8
88	1201	0.13	9238.5
89	1146	0.1	11460.0
90	1138	0.1	11380.0
91	1063	0.11	9663.6
92	1192	0.14	8514.3
93	1125	0.12	9375.0

PREDICTIVE LTV FOR ECOMMERCE

As we said earlier, when estimating LTV for ecommerce we can’t ignore Gross Margin because COGS can significantly influence our results. If you keep things simple, you can use the following formula:

$$LTV = \frac{Orders \times AOV \times AGM}{Churn\ rate}$$

Where:
Orders = Average monthly orders
AOV = Average order value
AGM = Average gross margin

While we can consider churn rate in SaaS as roughly linear, in ecommerce it highly depends on different time factors, for example, seasonality and sales. So ecommerce businesses prefer using machine learning for calculating accurate churn rate, or getting it automatically with business intelligence tools, such as *Statsbot*.

Check out how **30% more Giphy employees got involved with analytics after deploying Statsbot*

CALCULATING LTV FOR EACH INDIVIDUAL CUSTOMER

Customer relationship management is a big buzzword in industry at the moment, and of course, it makes sense to want to personalize service to the highest extent possible, and to the extent that it brings value to your customers. And from a business standpoint, it makes sense to focus your energies on maintaining positive relationships with the customers who are going to bring you the most income. After all, according to Pareto's principle, 20% of your customers generate 80% of your revenue.

> The "building blocks" of a customer's value to a company are frequency, average size, and recency of purchase, which is another way of saying their spending pattern.

Frequency and average size are pretty intuitively important: how often does a customer visit, and how much do they spend when they're here? The reason that we also include recency, in a business model where there are no long-term contracts, is that it will give us an indication of whether the customer is still loyal to our business or not.

For example, if we run a coffee shop, and a customer visited us twice a week for 3 months, spending $10 each time, we might properly assess that this was a loyal and high-value customer. But if we add the information that their last visit was 6 months ago, we might reassess, and consider them to be a former high-value customer, whose loyalty we have now lost.

To keep consistency with the example given above, I'll look at transactions over the past year.

```
WITH cust_spending_pattern AS
(SELECT
     cust_id,
    Datediff('day', Min(transaction_date), CURRENT_DATE) AS first,
    Datediff('day', Max(transaction_date), CURRENT_DATE) AS last,
    Count(DISTINCT order_id) AS freq,
    Avg(order_total) AS avg_spend
FROM     transactions
WHERE  transaction_date > Dateadd('year', -1, CURRENT_DATE)
GROUP  BY 1
ORDER  BY 4 DESC)
```

In each line, you will then have the **key information for each customer, with your highest-spend customers at the top**. Earlier we saw how to calculate churn: this can give us a way to weight the value of a customer who has not visited in a while.

For example, if we saw that churn was 10% per month, we can estimate (roughly) that a customer who has not visited in a month is worth 10% less to us. The aim is not to calculate exactly how much a customer is worth, but rather, to assign priority in terms of marketing efforts.

Monthly spend can be calculated like this:

```
WITH cust_monthly AS
(SELECT
    cust_id,
    first,
    last,
    freq/((last-first)/30)*avg_spend AS monthly_spend
FROM   cust_spending_patterns)
```

We can then integrate recency into our estimate of how much they are likely to spend next month:

```
SELECT
     cust_id,
     Datediff(‘day’, last, CURRENT_DATE) / 30 AS months_since_visit,
     Datediff(‘day’, last, CURRENT_DATE) / 30 *(1- 0.1) * monthly_spend AS
next_month_estimate
FROM    cust_monthly
ORDER  BY 3 DESC
```

Generally speaking, it is more profitable to focus marketing efforts on customers who are engaged with your business rather than trying to win back the loyalty of those who have dropped off. So, we have ordered the extraction by the estimate of next month’s spend, which takes into account not only how much people spend, but the churn rate of the group.

CONCLUSION

Estimating LTV, both for groups and for individuals, for SaaS and ecommerce, is a key way of managing the risks that a business undertakes, and of deciding how much it is acceptable to spend in order to acquire or retain customers.

Using SQL and a simple transactional database, you can go a long way towards calculating LTV, without needing to use sophisticated statistical models. Or you can always rely on *Statsbot* to deal with this work.

**Get more useful resources for business analysis at statsbot.co/blog.*