Рабочее окружение

> **JupyterLab**

[JupyterLab](https://jupyterlab.readthedocs.io/en/latest/) — среда для написания и выполнения кода, родственная классическому Jupyter Notebook, но с рядом дополнительных возможностей. В рамках курса мы будем пользоваться его разновидностью под названием JupyterHub — эта версия практически не отличается от обычного JupyterLab, но она адаптирована для использования на сервере.

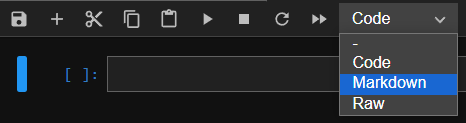
Иногда при длительной работе, а также после выхода из JupyterHub вас может выкинуть из вашего окружения. Это может проявиться в том, что некоторые библиотеки перестанут импортироваться. В таком случае вы можете остановить сервер и войти в него заново следующей последовательностью: File → Hub Control Panel → Stop My Server → Start Server, после чего снова выбрать нужное окружение.

* Python 3 — файл формата .ipynb, он же ноутбук. В основном код вы будете писать там.
* Python File — файл формата .py. Фактически обычный текстовый файл, в котором можно писать код и который будет восприниматься как код на Python. Этот тип файла будет вам полезен в уроках, связанных с использованием Airflow.
* Terminal — командная строка. Она будет полезна для работы с Git (о нём в следующих шагах).

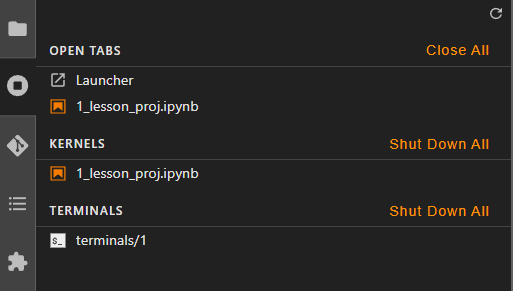
## **Другие полезные вещи**

Основную информацию вы можете найти в документации JupyterLab. Здесь мы расскажем вам о паре вещей, которые вам могут быть полезны в первую очередь.

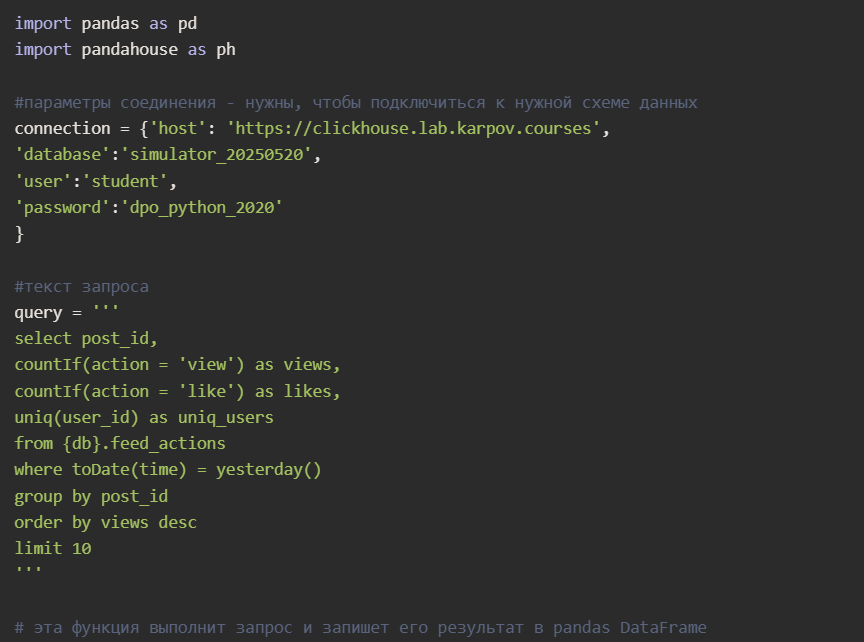
* Settings позволит вам настроить внешнее отображение и некоторые другие интересные вещи. В частности, там можно поменять тему с белой на чёрную (Theme → Dark) и включить автоматическое закрытие скобок (Auto Close Brackets)
* В разделе Kernel вы можете остановить текущую выполняющуюся ячейку (Interrupt Kernel), а также перезапустить ноутбук целиком (Restart Kernel с разными опциями)
* Если выделить ячейку внутри ноутбука и выбрать в качестве её типа Markdown, то вы можете писать в ней текст и форматировать его. Основной синтаксис форматирования Markdown можно подглядеть [тут](https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html).



* Если на левой панели выбрать значок в виде черного квадрата внутри белого кружка, то вместо списка файлов и папок вы увидите текущие открытые вкладки, а также активные ноутбуки и терминалы. С этой страницы любой из них можно остановить (Shut Down), чтобы освободить ресурсы.



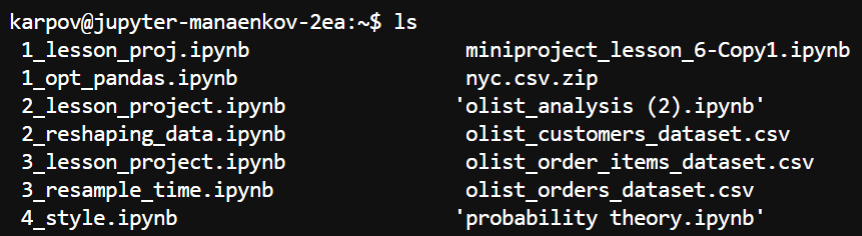
* Вы можете писать запросы к SQL-таблицам прямо из JupyterHub! Для этого вам понадобится пакет [pandahouse](https://github.com/kszucs/pandahouse) (установлен на сервере).



Помимо read\_clickhouse(), вам могут быть полезны команды execute() (выполняет произвольный запрос, полезна для создания пустых табличек) и to\_clickhouse() (записывает датафрейм в SQL-таблицу, присоединяя её снизу).

## **ls**

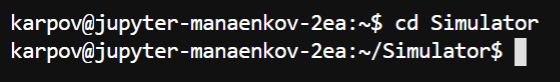
Сокращение от *list* — эта команда перечисляет все объекты, которые есть в текущей рабочей директории (грубо говоря, папке, в которой вы находитесь).



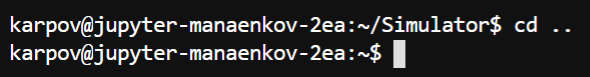
## **cd**

Сокращение от *change directory* — эта команда позволяет менять текущую рабочую директорию. Используется как cd путь\_к\_папке. Пути являются относительными, то есть отсчитываются относительно той директории, где вы находитесь в данный момент.

В примере ниже мы переходим в уже существующую папку под названием Simulator. Обратите внимание, как изменилась отображающаяся директория:



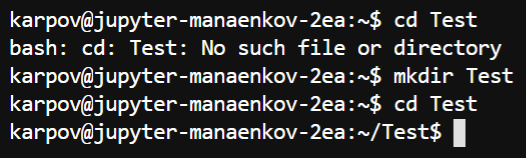
  Чтобы выйти из текущей директории, вам пригодится команда cd ..:



## **mkdir**

Сокращение от *make directory* — эта команда позволяет создать новую рабочую директорию. Аналогична кнопке New Folder в JupyterLab. Используется как mkdir имя\_папки.

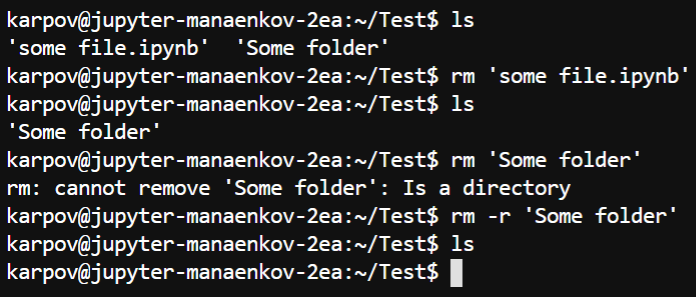
Ниже мы пробуем войти в несуществующую папку Test, на что терминал выдаёт ошибку об отсутствии такой папки. После её создания с помощью mkdir мы можем без проблем войти в неё.



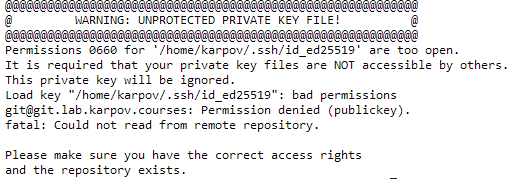
rm

Сокращение от *remove* — эта команда позволяет удалить файл. Используется как rm имя\_файла. Если хотите удалить папку вместе со всем её содержимым, то нужно указать дополнительный модификатор -r — тогда команда будет выглядеть как rm -r название\_папки.

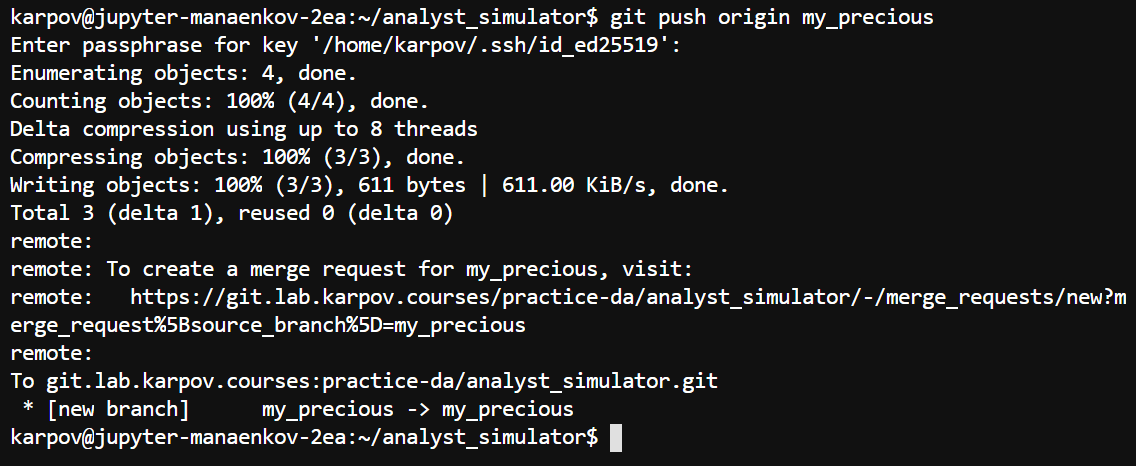
Ниже мы создали внутри папки Test два объекта: файл some file.ipynb и папку Some folder. Обратите внимание, что rm без проблем удаляет файл, но выдаёт ошибку при удалении папки. rm -r удаляет её без проблем.



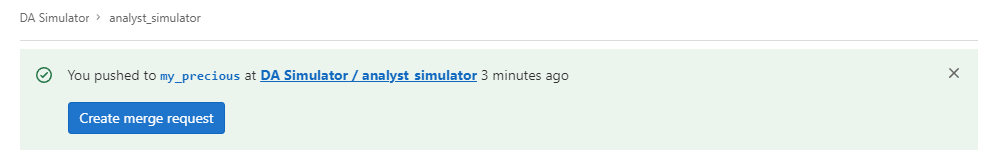
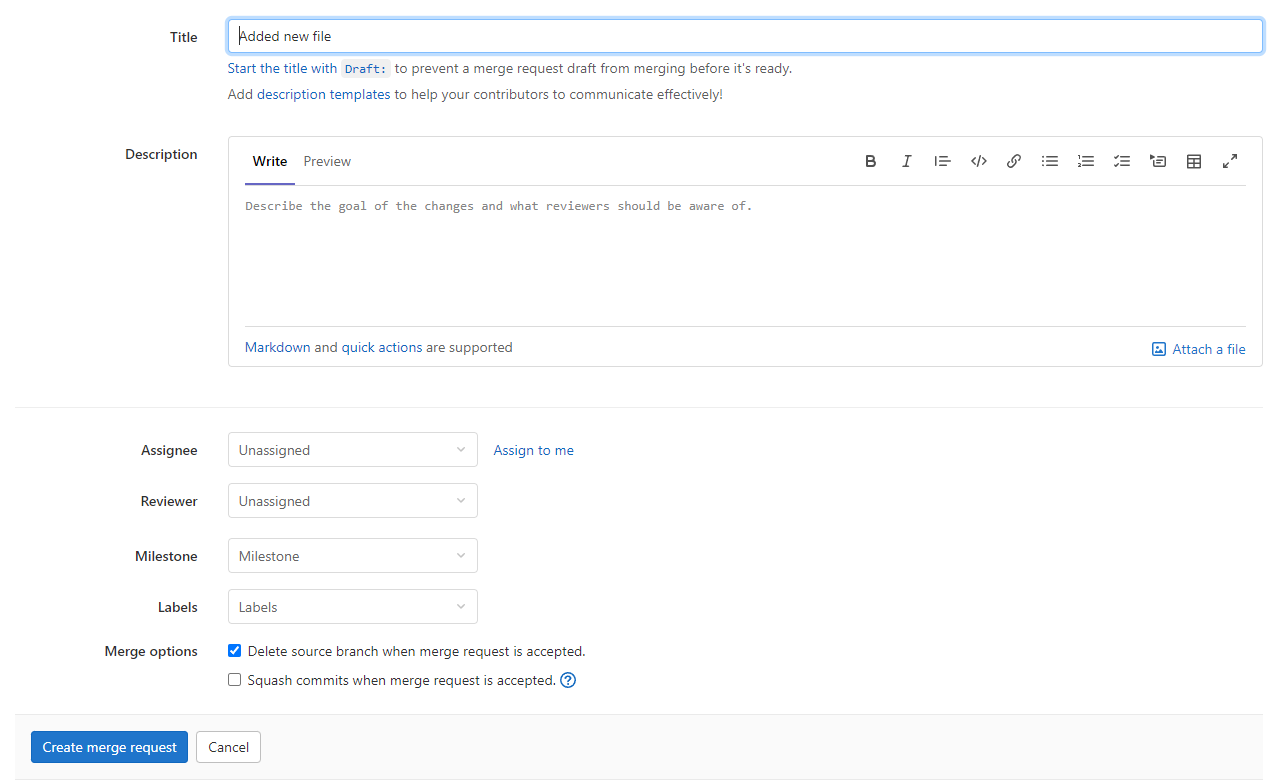
Пара дополнительных замечаний:

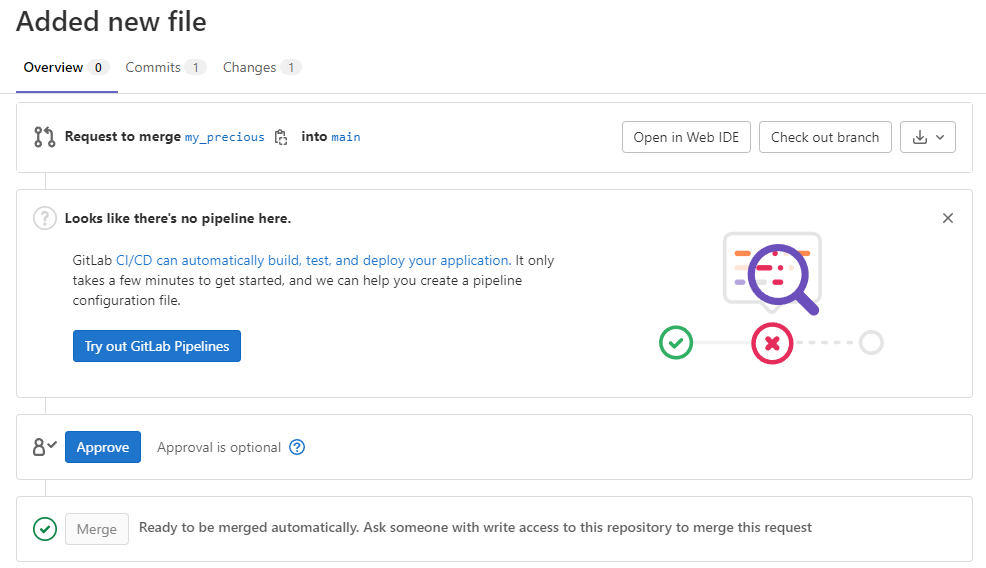
* Обратите внимание, что названия объектов выше обрамлены в кавычки. Это связано с наличием пробела в названиях; если бы мы назвали свои объекты some\_file.ipynb и Some\_folder, то их не пришлось бы ставить. В целом избегать пробелов в названиях — хорошая практика, особенно в контексте Git.
* rm удаляет объекты насовсем. **Если вы что-то удалили, то вернуть это нельзя**.
* Многие команды терминала имеют дополнительные опции, меняющие результат выполнения этой команды — rm -r является одним из таких вариантов. Можете поискать в интернете документацию всех описанных здесь команд и посмотреть, какие опции есть у каждой.
* > **Частые проблемы**
* 1. Первая распространённая проблема, пугающая новичков — git clone или другая команда не выполняется, а вместо этого пишется вот такое сообщение:
* **\*\**\* Please tell me who you are.***
* ***Run***
* ***git config --global user.email "you@example.com"***
* ***git config --global user.name "Your Name"***
* ***to set your account's default identity.***
* ***Omit --global to set the identity only in this repository.***
* Это происходит в том случае, если вы впервые используете Git на этой машине — он хочет с вами познакомиться. Выполните дословно обе команды, которые он вам пишет, но вместо you@example.com укажите свою почту, а вместо Your Name — своё имя латиницей. После этого повторите изначальную команду, она должна заработать. Сделать это нужно только один раз.
* 2. Иногда при попытке выполнить любую команду возникает вот такая ошибка:
* 
* Эта ошибка связана с излишне широкими правами доступа вашего ключа. Чтобы её решить, выполните команду chmod 400 ~/.ssh/id\_ed25519, после чего повторите вашу изначальную команду. Эта ошибка может возникнуть больше одного раза — во всех случаях выполняете команду выше.
* 3.  Вы можете натолкнуться на ошибку fatal: not a git repository. Это означает, что вы забыли зайти внутрь репозитория — сделайте это с помощью команды cd.
* 4. Изредка команды git clone и git push могут отказаться работать без объективных причин. В таком случае помогает удаление ключей из папки .ssh и с GitLab — после удаления создайте ключи заново по схеме из прошлого шага, и всё должно заработать.

# > git push

* Всё, что вы делали в последних трёх шагах, касалось лишь вашей локальной копии репозитория. Осталось поделиться своими наработками с миром и отправить свою ветку в удалённый репозиторий! В этом нам поможет команда git push origin название\_ветки — *origin* в данном случае обозначает оригинальный репозиторий, копию которого мы сделали с помощью git clone и с которой мы работали всё это время.
* После выполнения команды вам опять потребуется ввести пароль — и ваша ветка будет *запушена* в удалённый репозиторий! Остаётся лишь последний шаг.
* 
* **Важно**: вы не можете пушить ветку main, если вы не являетесь хозяином репозитория. Поэтому мы и создавали новую ветку.

## **Merge Request**

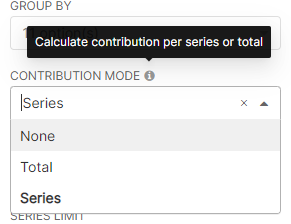
* Последний шаг, который вам остаётся сделать — это создать *запрос на слияние*, он же *Merge Request* (в GitHub это называется *Pull Request*). Для этого вам нужно перейти по ссылке, которую вам даст терминал после git push — но вы также можете просто войти в удалённый репозиторий в GitLab, вверху будет отображаться вот такое сообщение с синей кнопкой.
* 
* В любом случае вы попадёте на страницу создания Merge Request. При желании можете указать что-то в описании, но можете просто сразу нажать на кнопку *Create merge request*.
* 
* Готово! Перед вами будет вот такая страница. Если задание потребует от вас ссылку на Merge Request, то просто берёте ссылку из адресной строки и прикрепляете её на LMS.



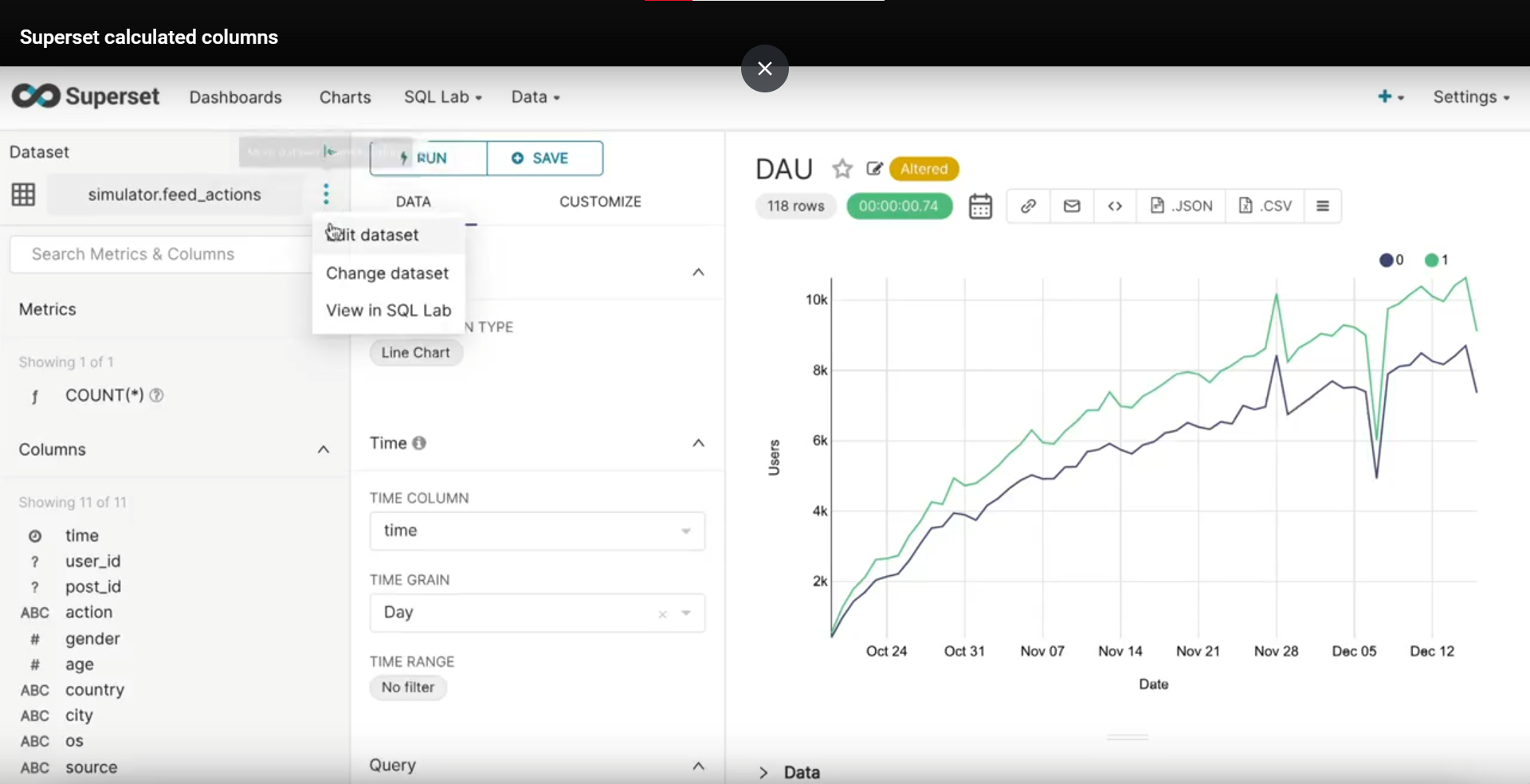
## **Дашборды**

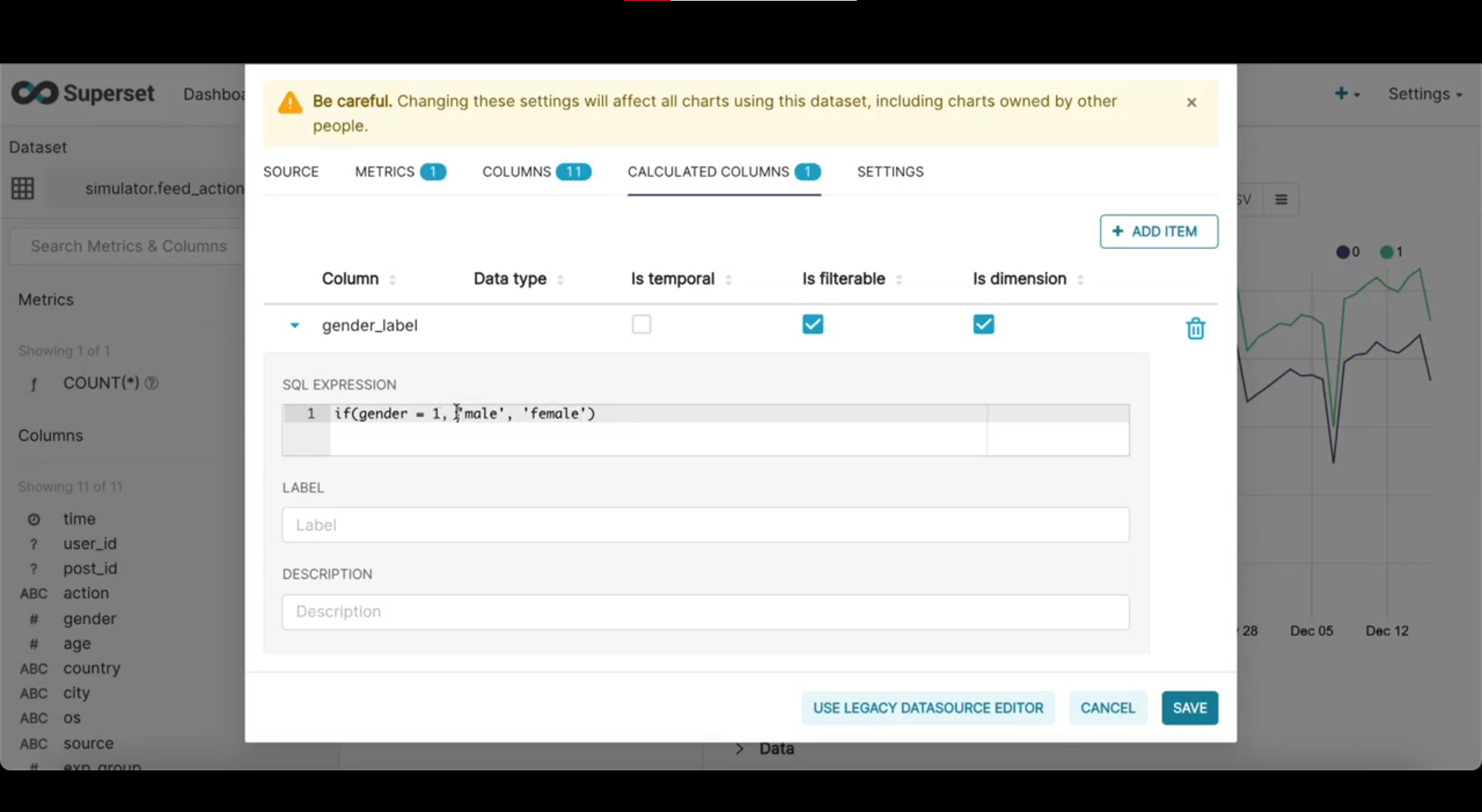
## **Пара дополнительных вещей, которые стоит упомянуть**

* Если мы сравниваем между собой величины на разных временных шкалах, то важно **нормализовать** данные. Это можно реализовать, например, через визуализацию не «чистой» метрики, а **процента вклада** каждого временного шага. Делается это так:



* А ещё в Superset можно создавать вычисляемые колонки, которые можно потом использовать для группировки! Здесь нам пригодится функция Clickhouse [multiIf](https://clickhouse.com/docs/en/sql-reference/functions/conditional-functions/#multiif). Например, если мы хотим раздробить возраста на несколько категорий, то мы можем сделать что-то подобное: multiIf(age < 20, '0 - 19', age >= 20 and age < 30, '20-29', '30+'). Но такого рода изменения могут быть опасными, если вычисляемая колонка представляет собой неоптимальный запрос. Поэтому сейчас (и, возможно, на вашей будущей работе) такие изменения вносятся через админов — присылайте нам код и название колонки, мы добавим. На практике, если вы хорошо себя зарекомендовали, вам могут выдать права на изменение сорса.





**Анализ продуктовых метрик**

# > Зачем нужен Retention?

Мы можем много смотреть на метрики активности пользователей в разные временные промежутки — на MAU, DAU, WAU. Радоваться тому, какие они большие и как они (потенциально) растут со временем, понимая, что это означает большую вовлеченность наших пользователей... Или нет? Почему в интернете можно найти много постов на тему того, что всё это — [«метрики тщеславия»](https://amplitude.com/blog/measuring-active-users), пользы от которых мало?

Проблема в том, что эта «активность пользователей» может не означать **реального интереса** пользователей к нашему продукту. Рост этих метрик может быть искусственным и не отражать **долговременную вовлечённость.**Иными словами, это легко манипулируемые, [«gameable» метрики](https://wrongbutuseful.com/2020/04/30/when-growth-metrics-go-bad/).

Примеры того, как мы можем увеличить активность пользователей, при этом потенциально пустив бизнес под откос:

1. Завлекать пользователей в наш продукт деньгами («установи приложение и получи 1000 рублей в подарок»).
2. Завлекать их обманом или дешёвым эпатажем — по аналогии с [clickbait](https://www.benchmarkone.com/blog/5-reasons-clickbait-negatively-impacts-business/) и [hate clicks](https://contently.com/2019/02/20/hate-clicks/).
3. Накидать ботов в наш продукт, создавая видимую активность.
4. «Подписать» человека на наш продукт без его желания .

Все эти варианты могут привести к (видимой) увеличенной активности, но реально приводить к оттоку наших клиентов и даже гибели бизнеса. Можете также придумать свои варианты, когда такое может случиться. Подумайте в сторону технических проблем и [product-market fit](https://vc.ru/marketing/88908-product-market-fit-chto-eto-takoe-i-pochemu-bez-nego-startap-ne-vzletit).

Если же мы хотим понять, сколько пользователей реально остаётся и продолжает пользоваться продуктом, мы должны посчитать **retention** (он же **retention rate**). Часто можно видеть следующую формулу:

*RR*=количество клиентов в начале периода(количество клиентов на конец периода−количество новых клиентов за период)​⋅100%

Но не все используют именно этот вариант формулы, можно встретить и другие. Например, часто встречается [формулировка](https://amplitude.com/blog/how-to-calculate-retention-rate-b2b-saas) «количество активных пользователей за текущий период/общее число пользователей за прошлый период». Или «число пользователей за n-ый период/число пользователей в когорте». **Эти варианты больше соответствуют тому, как Толя считал retention в лекции** :)

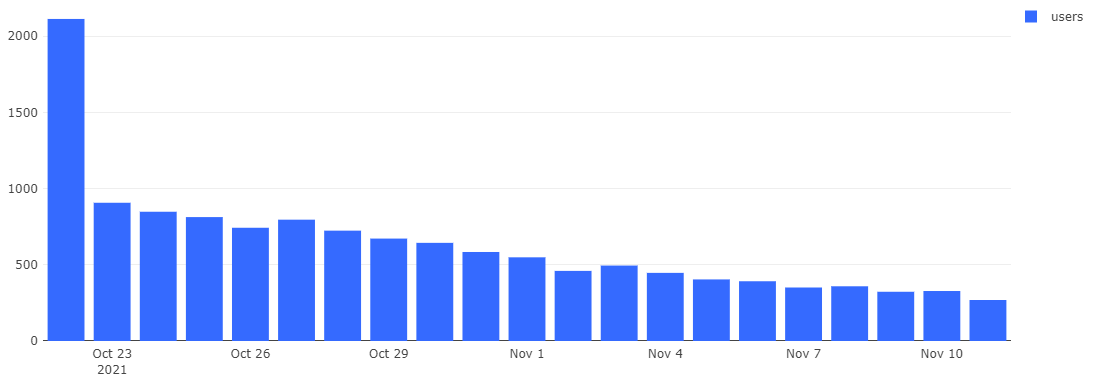
Есть и [другие](https://www.braze.com/resources/articles/calculate-retention-rate) типы retention. Более того, можно считать не остающихся клиентов, а тех, которые уходят — это будет называться [churn rate](https://www.investopedia.com/terms/c/churnrate.asp).

**Внимание!** Потенциально манипулировать можно любой метрикой, retention не исключение. Помните про [закон Гудхарта](https://retireearly.ru/financial-literacy/zakon-gudharta) и будьте осторожны.

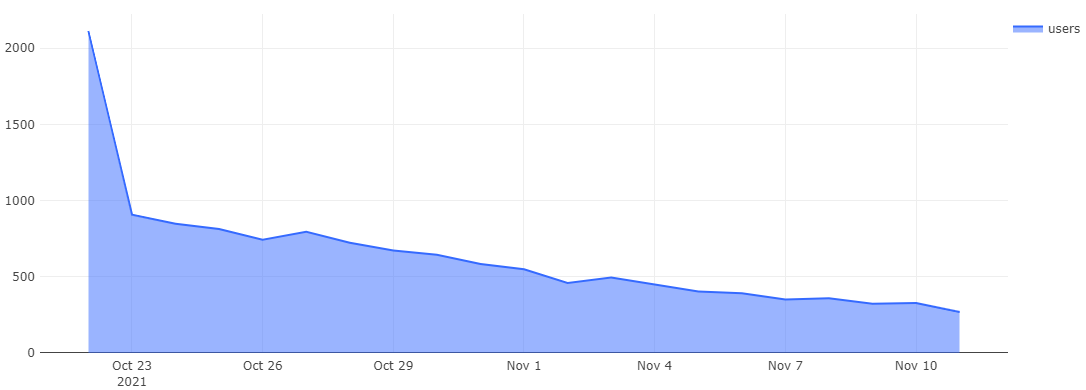
**Внимание (2)!**Retention не всегда бывает полезной метрикой.Например, если мы продаём **товары для маленьких детей**, то подавляющее большинство покупателей к нам не будет возвращаться, и это нормально :)

# > Как изобразить Retention?

На лекции Толя изображал его через столбиковую диаграмму, меняющуюся во времени:

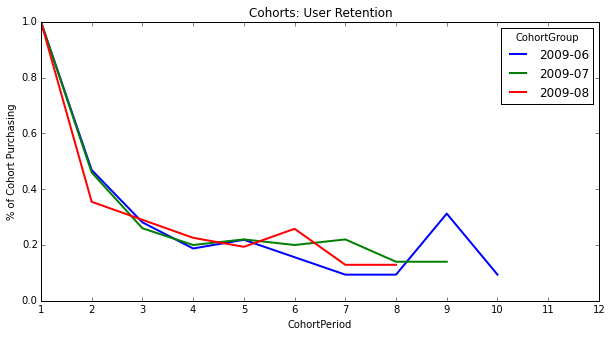


Естественно, вместо столбиков можно рисовать и более традиционные линии. При желании можно даже закрасить пространство под этой линией и сделать area chart:

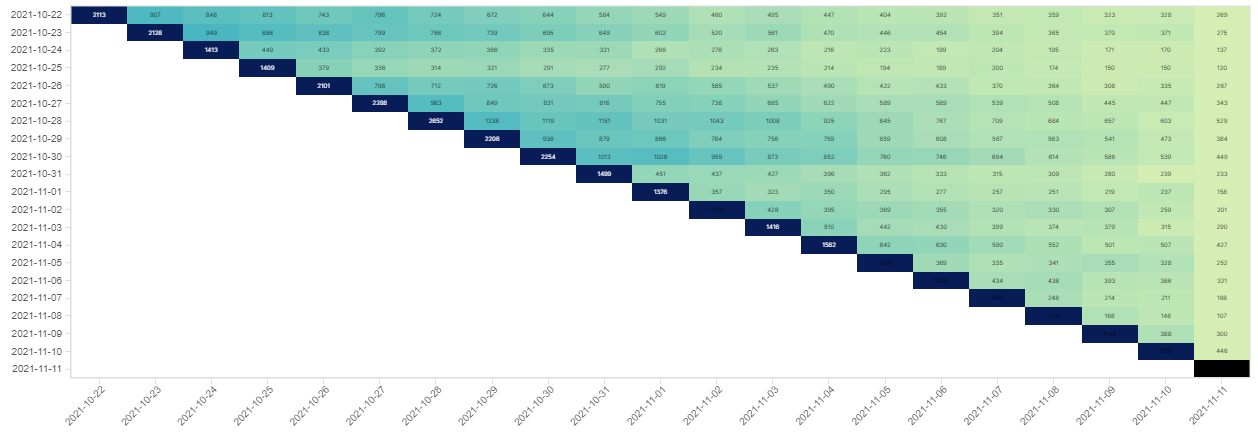


Но отдельные линии без закрашивания удобнее, если мы хотим визуализировать несколько **когорт**. Когорты — это группы наших пользователей, которые совершили определённое действие в определённый отрезок времени.

Иными словами, мы можем визуализировать когорты людей, впервые проявившие активность в определённом месяце, и сравнить их динамику между собой:



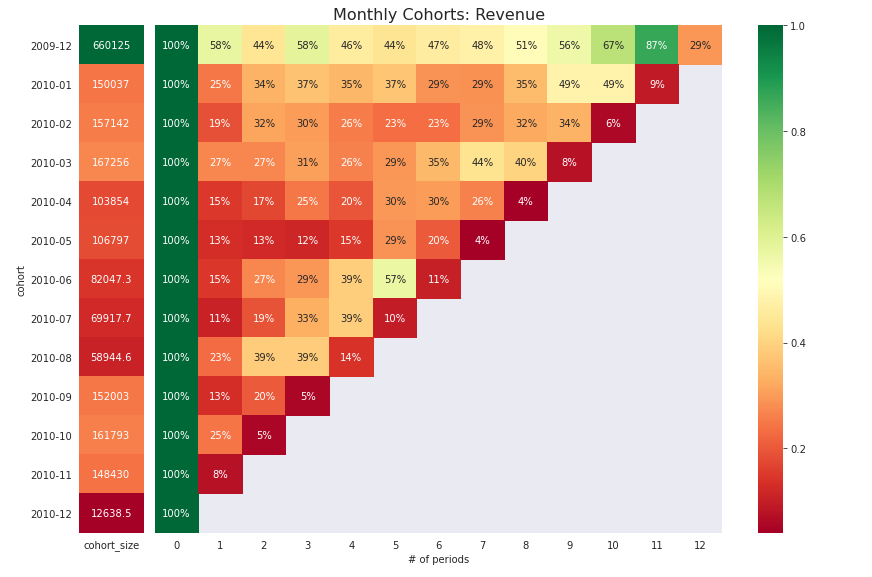
Однако наибольшую популярность в индустрии получила визуализация когорт через тепловые карты:



Как это интерпретировать?

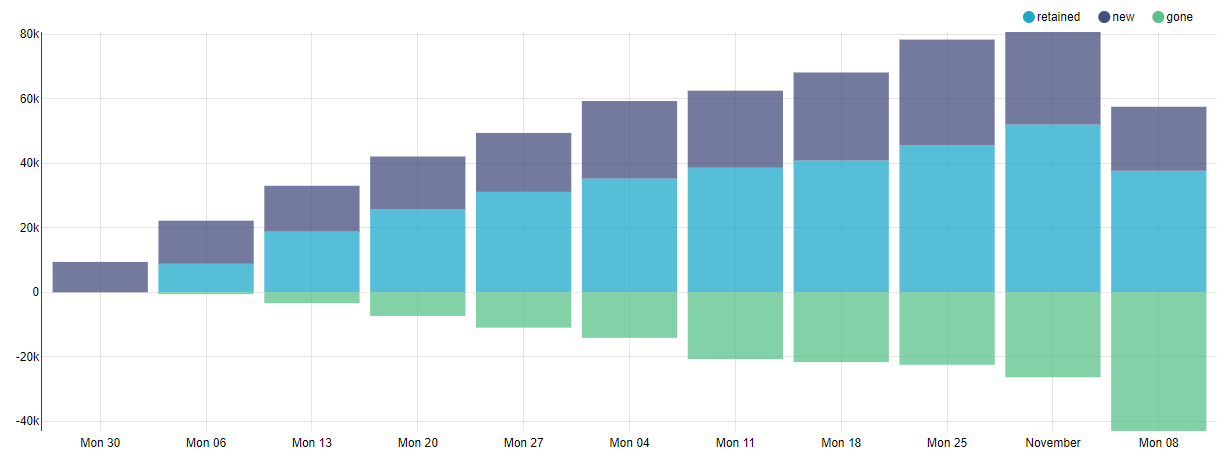
1. По оси x — шкала времени. Здесь мы видим каждый новый день на определённом промежутке.
2. По оси y — день первого появления. По ней мы можем отследить, с какого дня начинается конкретная когорта.
3. Этому же соответствуют темно-синие ячейки на диагонали. Так, в верхнем левом углу мы видим пользователей, впервые начавших пользоваться нашим сервисом **2021-10-22**. Ниже по диагонали — те, кто появился **2021-10-23**. И так вплоть до **2021-11-11**, где наш график заканчивается.
4. По цвету каждой ячейки можно понять, насколько много пользователей «удержалось» по сравнению с первым днём. В данном случае чем **светлее** — тем **меньше** человек осталось в когорте. Естественно, это зависит от выбранной вами цветовой палитры.
5. Если вам нужны конкретные значения, то в каждой ячейке можно посмотреть точное количество человек в конкретный день + их процент от изначального количества. Само собой, в первый день когорты этот процент равен 100 :)
6. Очень важный момент — чтобы всё это было наглядно, нужна **нормировка тепловой карты** по колонкам. Различие между нормировкой по рядам и колонкам можно видеть [тут](https://www.python-graph-gallery.com/94-use-normalization-on-seaborn-heatmap).

Можно также встретить варианты тепловой карты вроде такой:



Из наиболее важных особенностей — карта «отзеркалена» по сравнению с прошлым вариантом, а по оси x указаны **номера периодов**, а не конкретные даты. Можете попробовать сделать что-то похожее в Superset :)

Также у Толи был вот такой график, нарисовать который вам предстоит самостоятельно:



На что стоит обратить внимание:

* Здесь указана недельная динамика, рисуются не все дни (так компактнее);
* В оригинальном запросе использовалось среднее количество юзеров за каждую неделю и каждую **группу,**однако вполне можно считать и просто **количество уникальных пользователей**;
* **Группы** пользователей (оставшиеся, новые, ушедшие) выделялись до рисования графика — вам придётся поработать в SQL;
* «Ушедшие» пользователи имеют отрицательные значения на этой шкале.

# > Дополнительно про анализ

Анализировать данные об «удержании» (retention) и «оттоке» (churn) можно [множеством разных способов](https://www.researchgate.net/publication/2344226_Modelling_Customer_Retention_with_Statistical_Techniques_Rough_Data_Models_and_Genetic_Programming) в зависимости от поставленной гипотезы. Среди более продвинутых вариантов отдельно стоит выделить [анализ выживаемости](https://habr.com/ru/company/newprolab/blog/333628/).

Изначально этот метод применялся в медицине для анализа факторов, влияющих на продолжительность жизни пациентов. При этом он учитывает тот факт, что пациенты могут выпадать из исследования ввиду своей смерти — отсюда и название.

Никто не заставляет использовать именно смерть в качестве возможного исхода! Мы можем анализировать **время до любого события, которое имеет бинарный исход** (произошло или нет).

Например, прекратил ли человек пользоваться сервисом или нет.

Пример подобного анализа можно видеть по ссылке вверху, а также в нашем [открытом вебинаре](https://vk.com/video760719865_456240781?list=ln-MrKS6n0Bw6ReNvzJ7B) ([YouTube](https://www.youtube.com/watch?v=1-ADJNE1WNc)) по **обобщённым линейным моделям**. Модели такого рода начинают обсуждаться примерно с 1:05:00 :)

Использованный пакет: [lifelines](https://lifelines.readthedocs.io/en/latest/)

Запросы, которые мы написали в видео для расчета Retention.

SELECT date, count(user\_id)

FROM

(

SELECT user\_id

FROM simulator\_20250520.feed\_actions

GROUP BY user\_id

HAVING min(toDate(time)) = today() - 30

) t1

JOIN

(

SELECT DISTINCT user\_id, toDate(time) AS date

FROM simulator\_20250520.feed\_actions

) t2

USING user\_id

GROUP BY date

SELECT date, count(user\_id) AS users

FROM

(SELECT DISTINCT user\_id,

toDate(time) AS date

FROM simulator.feed\_actions

WHERE user\_id in

(SELECT DISTINCT user\_id

FROM

(SELECT user\_id,

min(toDate(time)) AS start\_day

FROM simulator.feed\_actions

GROUP BY user\_id

HAVING start\_day = today() - 20)) )

GROUP BY date

SELECT start\_day,

day,

count(user\_id) AS users

FROM

(SELECT \*

FROM

(SELECT user\_id,

min(toDate(time)) AS start\_day

FROM simulator.feed\_actions

GROUP BY user\_id) t1

JOIN

(SELECT DISTINCT user\_id,

toDate(time) AS day

FROM simulator.feed\_actions) t2 USING user\_id

WHERE start\_day >= today() - 10 )

GROUP BY start\_day,

Day

SELECT toString(start\_day) start\_day,

toString(day) day,

count(user\_id) AS users

FROM

(SELECT \*

FROM

(SELECT user\_id,

min(toDate(time)) AS start\_day

FROM simulator.feed\_actions

GROUP BY user\_id) t1

JOIN

(SELECT DISTINCT user\_id,

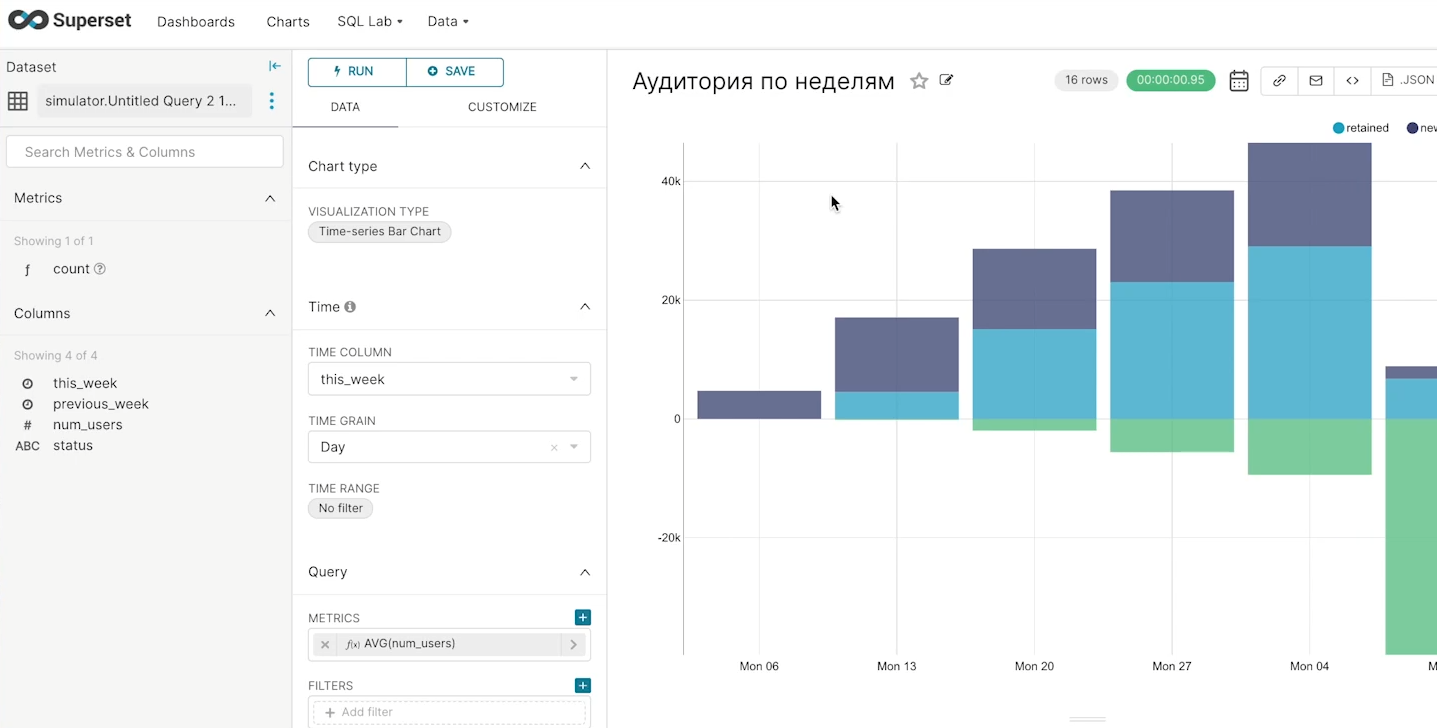
toDate(time) AS day

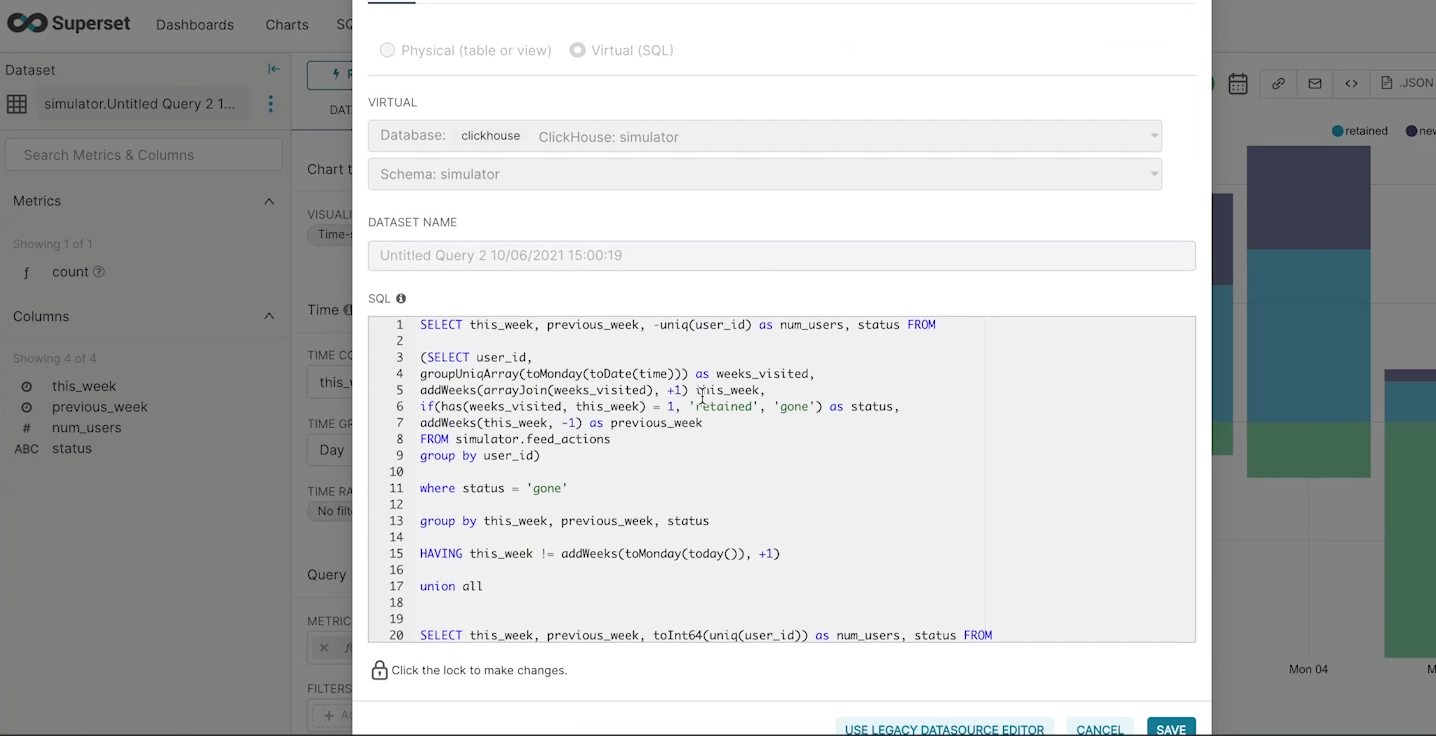
FROM simulator.feed\_actions) t2 USING user\_id

WHERE start\_day >= today() - 20 )

GROUP BY start\_day,

day



­­

