

Прикладные инструменты визуализации

Цель занятия

После освоения темы:

- вы узнаете об инструментах для визуализации данных, используемых в работе аналитика;
- изучите функциональные возможности сервиса Yandex DataLens для построения и настройки графиков;
- сможете строить некоторые виды графиков в Yandex DataLens;
- сможете строить графики и диаграммы с помощью библиотеки `Altair`, добавлять в графики интерактив и выполнять их настройку;
- сможете строить статистические диаграммы с использованием библиотеки `Seaborn`.

План занятия

1. [Yandex DataLens: построение диаграмм без программирования](#)
2. [Использование библиотек Python для визуализации. Библиотека Altair](#)
3. [Использование библиотек Python для визуализации. Библиотека Seaborn](#)

Конспект занятия

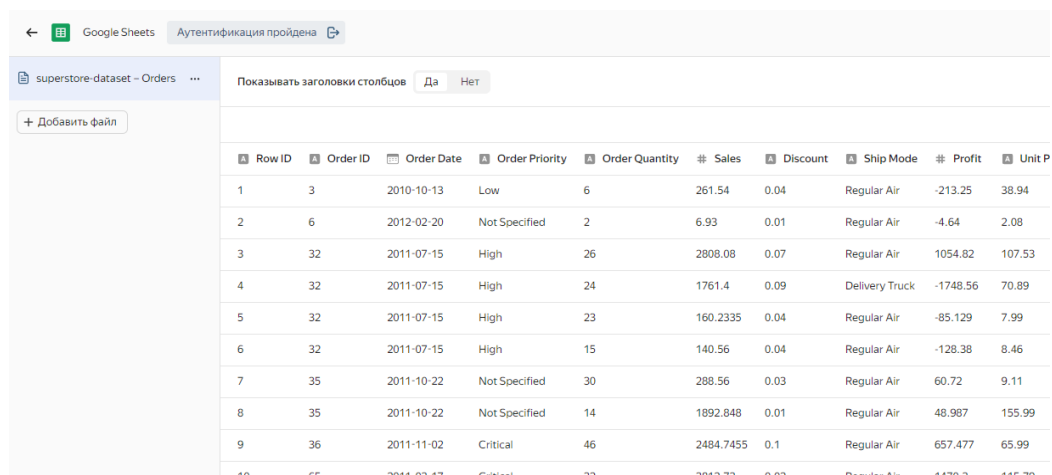
1. Yandex DataLens: построение диаграмм без программирования

[Yandex DataLens](#) — один из самых доступных среди бесплатных инструментов создания графиков. Инструмент можно использовать онлайн, не надо ничего устанавливать на компьютер. С помощью Yandex DataLens можно строить большинство популярных видов графиков, подключаться к различным источникам данным.

Рассмотрим процесс создания графика.

Шаг 1. Подключаемся к источнику данных. Нажимаем «Создать подключение». Yandex DataLens работает со многими популярными базами данных. Также можно подключиться к файлу — загрузить его с расширением `csv`.

В примере мы берем данные из Google-таблицы. Выберем подключение к Google-документам и загрузим заранее заготовленную таблицу `superstore-dataset`. Она содержит данные магазина офисной техники:



The screenshot shows a Google Sheets interface with a table titled 'superstore-dataset - Orders'. The table has 10 rows of data. The columns are: Row ID, Order ID, Order Date, Order Priority, Order Quantity, Sales, Discount, Ship Mode, Profit, and Unit Price. The data is as follows:

Row ID	Order ID	Order Date	Order Priority	Order Quantity	Sales	Discount	Ship Mode	Profit	Unit Price
1	3	2010-10-13	Low	6	261.54	0.04	Regular Air	-213.25	38.94
2	6	2012-02-20	Not Specified	2	6.93	0.01	Regular Air	-4.64	2.08
3	32	2011-07-15	High	26	2808.08	0.07	Regular Air	1054.82	107.53
4	32	2011-07-15	High	24	1761.4	0.09	Delivery Truck	-1748.56	70.89
5	32	2011-07-15	High	23	160.2335	0.04	Regular Air	-85.129	7.99
6	32	2011-07-15	High	15	140.56	0.04	Regular Air	-128.38	8.46
7	35	2011-10-22	Not Specified	30	288.56	0.03	Regular Air	60.72	9.11
8	35	2011-10-22	Not Specified	14	1892.848	0.01	Regular Air	48.987	155.99
9	36	2011-11-02	Critical	46	2484.7455	0.1	Regular Air	657.477	65.99
10	65	2011-03-17	Critical	32	3812.73	0.02	Regular Air	1470.3	115.79

Чтобы мы могли экспортировать данные таблицы, необходимо разрешить доступ на просмотр по ссылке. Нужно скопировать ссылку на таблицу и вставить ее в Yandex DataLens.

После загрузки таблицы изучим, как Yandex DataLens распознал поля:

- A — строковый тип данных;
- картинка календаря — дата;
- # — числовой тип данных.

Можно зафиксировать подключение: нажимаем кнопку «Создать подключение» и даем имя — `Superstore`.

Шаг 2. Создаем датасет. На основе этого подключения создаем датасет — конкретный набор данных для построения графиков.

Выберем, какие поля хотим включить в датасет и какого типа они будут. При экспорте не все поля были распознаны правильно, изменим их тип:

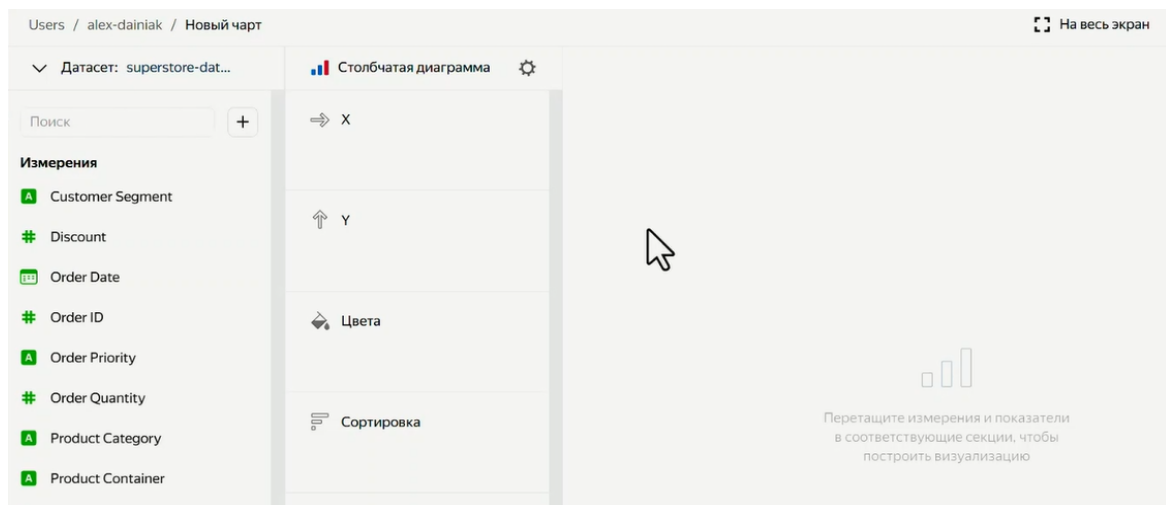
- поле `Row ID` (строка → целое число);
- поле `Order ID` (строка → целое число);
- поле `Order Quantity` (строка → целое число);
- поле `Sales` (строка → дробное число);
- поле `Discount` (строка → дробное число);
- поле `Profit` (строка → дробное число);
- поле `Unit Price` (строка → дробное число);
- поле `Shipping Cost` (строка → дробное число).

Далее скрываем поля, которые нам не понадобятся, например, `Customer Name`.

На этом шаге можно добавить параметры датасета — некоторые константы, которые относятся ко всему датасету.

Зафиксируем датасет, назовем его `Superstore-dataset`.

Шаг 3. Создаем график. Теперь на основе датасета создадим график с помощью кнопки «Создать чарт». Перейдем в интерфейс графика:



Создаем таблицу. Для этого достаточно переместить в область столбцов таблицы параметры, которые у нас есть. Добавим даты (`Order Dates`) и продажи (`Sales`). Получаем таблицу, которая содержит два столбца исходной базы данных.

Видно, что даты дублируются. Чтобы каждая дата появлялась один раз, мы должны агрегировать продажи по дням года. Добавляем функцию и даем полю новое название `Total Sales`:

```
SUM([Sales])
```

Функцию выбираем из справочника. Названия функций похожи на те, что используются в Excel и SQL.

После выполнения описанных действий в таблице появился новый показатель — `Total Sales`, а также остались только уникальные даты.

Поле `Order Date` мы тоже можем агрегировать — просуммируем продажи по дням недели:

```
DAYOFWEEK([Order Date])
```

Результат:

Таблица

Столбцы

Day of Week

Total Sales

Цвета

Day of Week	Total Sales
1	
2	
3	2 168 291,48
4	1 863 484,41
5	2 426 249,07
6	2 255 133,91
7	2 237 338,01

Экспорт

Инспектор

В таблице можно задать дополнительную строку, вставить заголовок, установить количество видимых строк. Если информации очень много, то можно сделать таблицу с пагинацией, то есть с разбиением на страницы.

В данных часто бывают внутренние иерархические поля или пары полей, образующие внутреннюю иерархию. У нас это: категория продукта и подкатегория продукта. Yandex DataLens не знает об иерархичности. Добавим эту иерархию:

Настройка иерархии

Category-Subcategory

Доступно

Поиск

Order ID

Order Priority

Order Quantity

Product Container

Product Name

Profit

Province

Выбрано: 2

Очистить

Поиск

Product Category

Product Sub-Category

Отменить

Сохранить

Иерархии появляются отдельно. Мы можем «перетащить» иерархию в нашу таблицу, и таблица станет интерактивной. По щелчку для данной категории будет таблица распределения продаж по подкатегориям:

На весь экран Сохранить

Таблица

↑ ↓ / Product Category

Product Category	Total Sales
Furniture	5 178 590,54
Office Supplies	3 752 762,10
Technology	5 984 248,18
Total	14 915 600,82

Category-Subcategory

Total Sales

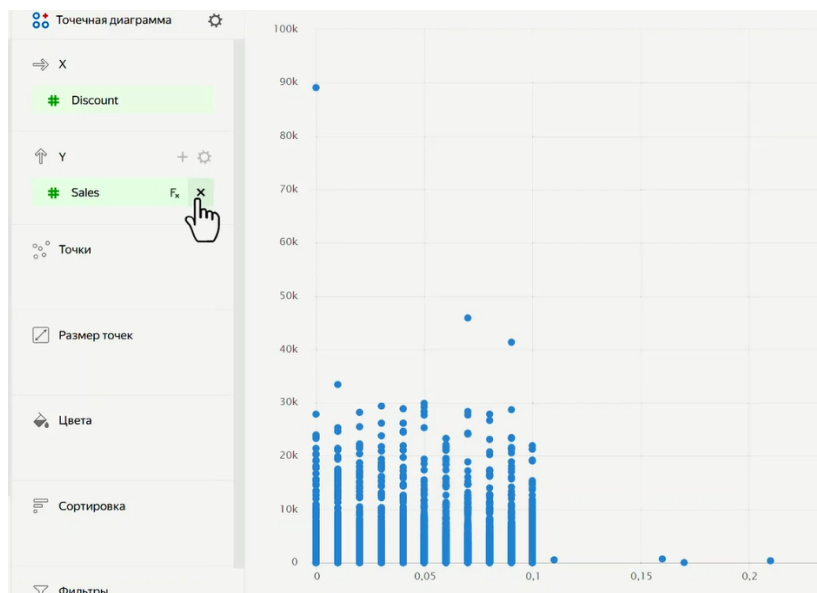
Цвета

Созданную таблицу можно открыть в новой вкладке, экспортировать или открыть для чтения. При открытии публичного доступа к таблице, необходимо опубликовать и данные, на основе которых была создана таблица.

Создаем точечную диаграмму. Создадим график на основе имеющегося датасета. Например, точечную диаграмму, в которой можно посмотреть, как продажи коррелируют со скидкой. При построении диаграммы укажем:

- в поле X значение `Discount`;
- в поле Y значение `Sales`.

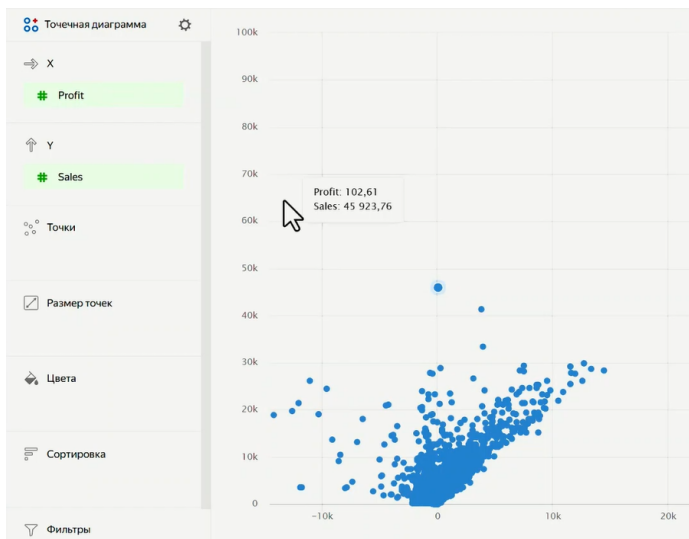
Результат построения:



Далее рассмотрим корреляцию продаж с прибылью, для этого укажем:

- в поле X значение `Profit`;
- в поле Y значение `Sales`.

Результат построения:

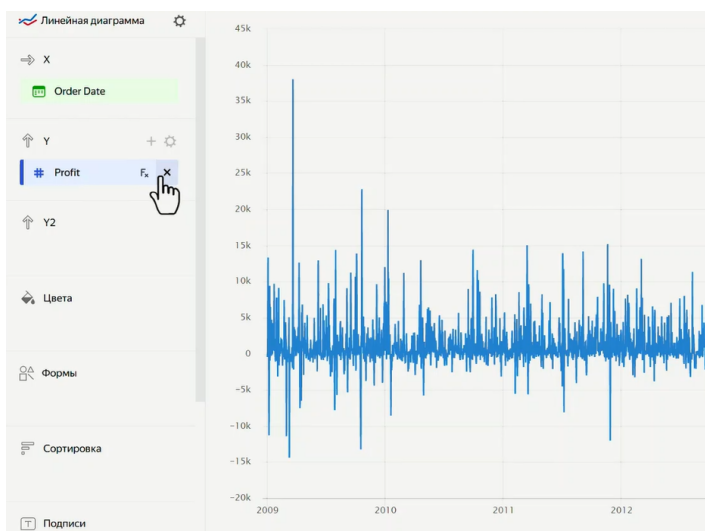


При изменении вида диаграммы «на ходу» мы по возможности сохраняем те поля, которые DataLens считает актуальными.

Создадим линейчатую диаграмму, показывающую зависимость прибыли от даты, прописав:

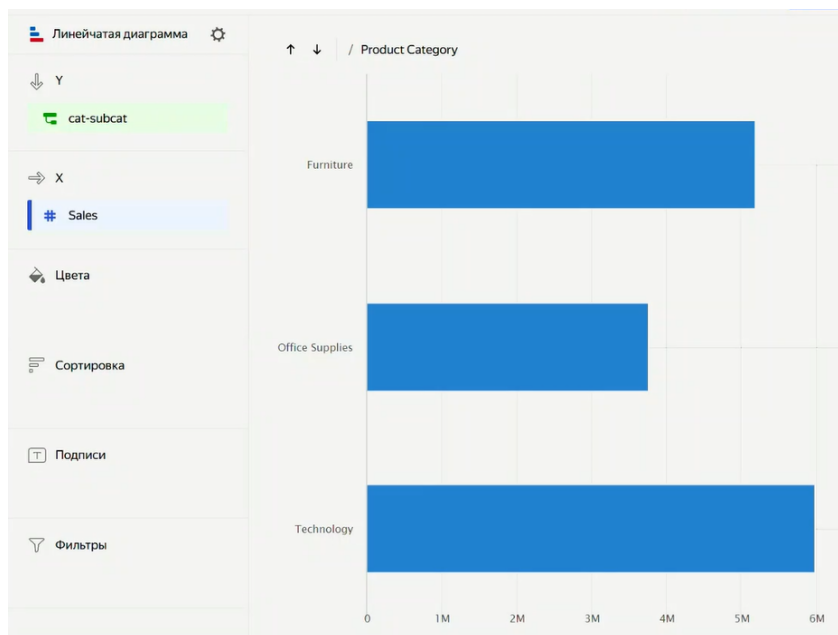
- в поле X значение `Order Date`;
- в поле Y значение `Profit`.

Результат построения:

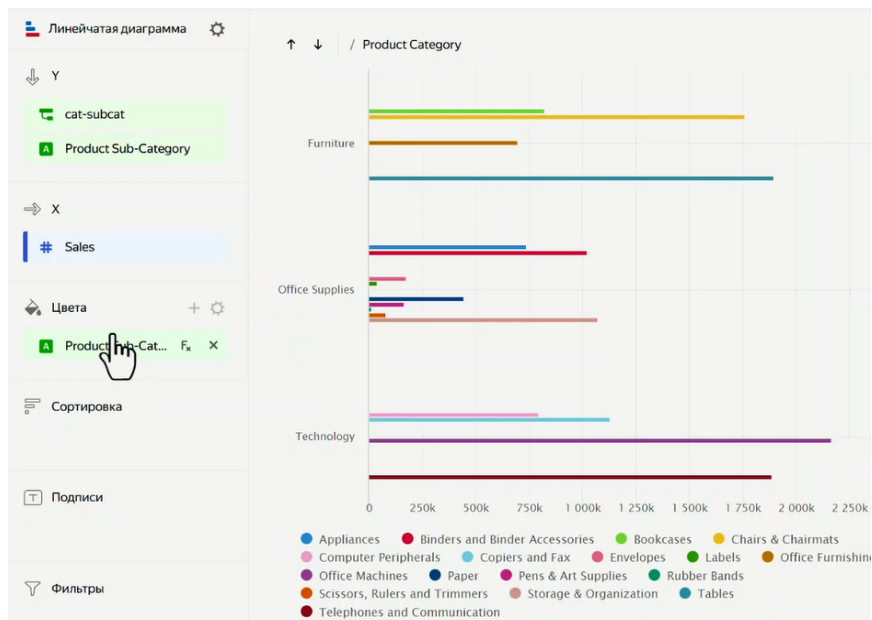


Сделаем линейчатую диаграмму с категориями. Зададим:

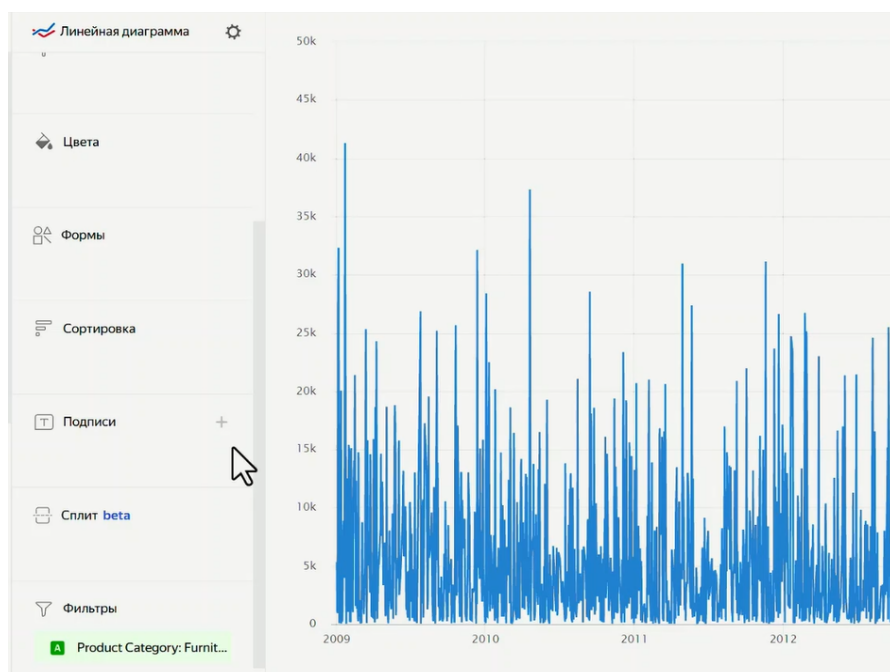
- в поле Y иерархию cat-subcat;
- в поле X значение Sales.



Теперь добавим подкатегории и выделим их цветом:



В графиках есть возможность фильтрации. Построим линейный график продаж по дате и добавим фильтр по категории продукта:



Функционал дашбордов позволяет создавать инфопанели, где можно встраивать несколько графиков на один экран, добавлять фильтры и настраивать взаимодействие фильтров с графиками и графиков друг с другом.

2. Использование библиотек Python для визуализации. Библиотека `Altair`

Библиотека `Altair` использует возможности библиотеки `Vega` для отображения графиков. Галерею графиков можно найти на официальной странице [библиотеки Altair](#).

Рассмотрим пример базы данных магазина `superstore-dataset`. Подгрузить ее можно из Google-документов в стандартную библиотеку Python для работы с данными — в библиотеку `Pandas`. Скопируем ссылку на таблицу в соответствующую функцию загрузки данных. Для этого воспользуемся библиотекой `gsread`¹:

¹ Если вы работаете в Google Collab, для корректной работы кода вам необходимо установить

```
import gspread
import pandas as pd

gc = gspread.service_account('service_account_key.json')
wks = gc.open_by_url(
    'https://docs.google.com/spreadsheets/d/1NObIGEC9ZxjOSs44KHLn-ftbs
    xdFHGVKB8Z-KzjPzYs/edit?usp=sharing' # загрузка данных с
    Google-диска
).worksheet(
    'Orders'
)
```

Далее подгружаем данные в датафрейм Pandas. Практически все библиотеки для визуализации предполагают, что данные им подаются в виде датафрейма Pandas или низкоуровневых массивов NumPy.

При загрузке данных в качестве индекса устанавливаем поле Row ID и столбцы, которые содержат даты, конвертируем в дату. Остальные данные останутся изначального типа:

```
superstore_data = pd.DataFrame(wks.get_all_records())
superstore_data = superstore_data.set_index('Row ID')
superstore_data['Order Date'] =
pd.to_datetime(superstore_data['Order Date'])
superstore_data['Ship Date'] =
pd.to_datetime(superstore_data['Ship Date'])
superstore_data.head()
```

Посмотрим, какие типы данных получились:

```
superstore_data.dtypes
```

Построение графиков в Altair. Попрактикуемся использовать библиотеку Altair. Она хороша тем, что построение графиков в ней выполняется на декларативном уровне. Мы описываем практически в терминах теории визуализации данных, какие мы будем использовать глифы, и как мы будем кодировать при помощи визуальных атрибутов атрибуты данных.

как минимум версию 5.4.7 библиотеки gspread и получить [ключ для сервисного аккаунта в формате JSON](#). Вы можете пропустить этот шаг и загрузить файл с данными непосредственно в Google Collab.

Для начала работы с библиотекой воспользуемся кодом — импортируем библиотеку и выполним предварительные настройки для работы:

```
import altair as alt
alt.themes.enable('dark')
alt.data_transformers.disable_max_rows();
alt.warnings.simplefilter(action='ignore')
```

Создадим наш первый график. Для этого используем конструктор, который называется `Chart`.

В качестве первого параметра указываем датасет, на основе которого построим график. Далее используем цепочечную нотацию и укажем вид используемых глифов. Все функции по выбору типа глифов начинаются со слова `mark`, мы используем `mark_point`.

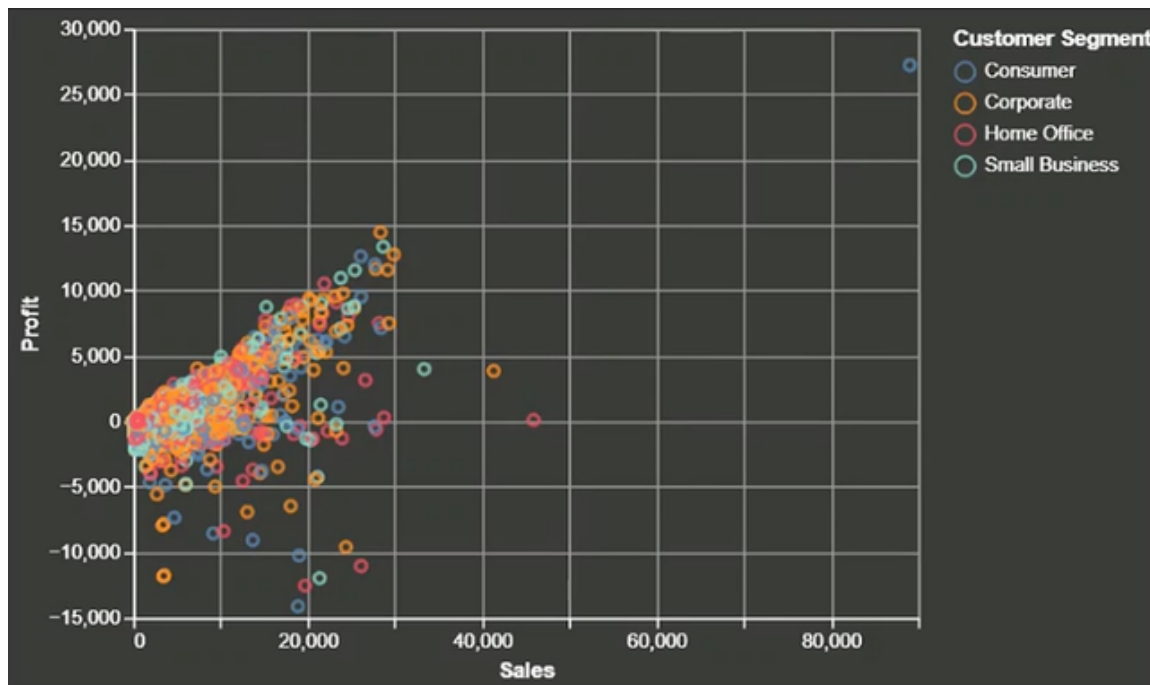
Построим диаграмму рассеяния — график корреляции `Sales` и `Profit`. Далее укажем, какие атрибуты данных будем кодировать. У точек есть горизонтальные и вертикальные координаты, цвет, прозрачность, внешний вид (треугольник, квадрат и т. п.). В примере мы выставим только горизонтальную и вертикальную координаты:

```
chart = alt.Chart(
    superstore_data
).mark_point(
).encode(
    x='Sales',
    y='Profit'
)
chart
```

Увеличим количество параметров в графике. Например, категоризируем данные по сегменту покупателей. Для этого добавим `Customer Segment` как поле данных, по которому выбирается цвет точек:

```
chart = alt.Chart(
    superstore_data
).mark_point(
).encode(
    x='Sales',
    y='Profit',
    color='Customer Segment'
)
chart
```

Теперь на графике появилась легенда и точки:



Можно сделать график минимально интерактивным. Чтобы добавить интерактивность в график по умолчанию, используем соответствующую функцию:

```
chart = alt.Chart(  
    superstore_data  
) .mark_point(  
) .encode(  
    x='Sales',  
    y='Profit',  
    color='Customer Segment'  
) .interactive()  
chart
```

Теперь с помощью колесика мышки можно увеличивать или уменьшать график, а курсором — перемещаться по графику.

Легенда графика не является интерактивной, и нельзя смотреть отдельные категории. В Altair мы можем создать некое выделение — интерактивный элемент.

Далее созданное выделение `selection` необходимо связать с графиком — как пользователь должен взаимодействовать с графиком, чтобы выделение произошло.

Мы создадим мультिवыбор, чтобы пользователь мог выбирать сразу несколько сегментов. В параметрах укажем, по каким полям будет выполняться фильтрация — `Customer Segment`. С помощью параметра `bind` выбираем, что выделение будет привязано к легенде:

```
selection = alt.selection_multi(fields=['Customer Segment'],
bind='legend')
```

После создания выделения его необходимо добавить на график — прописать метод `add_selection`. В примере мы будем менять прозрачность точек, для этого воспользуемся инструментом библиотеки Altair — `alt.condition`:

```
chart = alt.Chart(
    superstore_data
).mark_point(
).add_selection(
    selection
).encode(
    x='Sales',
    y='Profit',
    color='Customer Segment',
    opacity=alt.condition(selection, alt.value(1), alt.value(0))
).interactive()
chart
```

Теперь точки, не попавшие в выборку, «исчезают» с графика.

График можно сохранять в стандартных форматах (например, `svg`, `png`) или в некотором исходном виде — формате `json`:

```
print(chart.to_json())
```

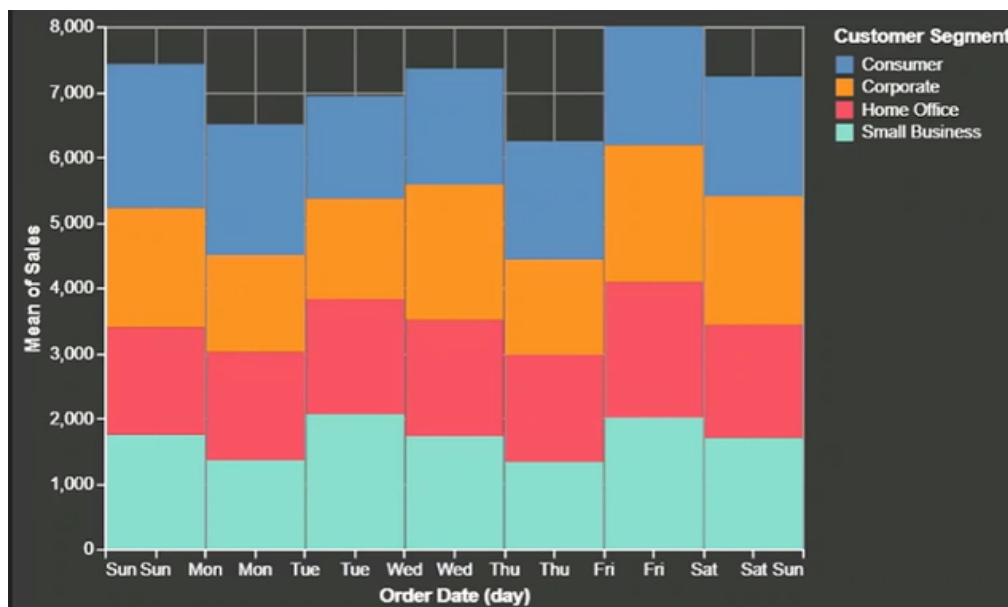
В формате `json` указывается полностью набор данных (поэтому получается довольно объемный файл) и то, что мы программировали в Python.

Этот файл можно вставить в специальный [редактор](#), где возможно изменить исходный `json`-файл и посмотреть результаты. Конечно, если данных много, то редактировать такой файл неудобно.

Создадим еще один график – столбчатую диаграмму распределения продаж по дням. Мы будем использовать функцию `day`, чтобы получить день недели, и функцию `mean` для получения среднего значения продаж:

```
chart = alt.Chart(
    superstore_data
).mark_bar(
).encode(
    x='day(Order Date)',
    y='mean(Sales)',
    color='Customer Segment'
)
chart
```

Результат:



На созданную диаграмму можно добавить интерактив. Например, сделать кликабельной легенду и отображать в столбцах диаграммы только выбранные категории.

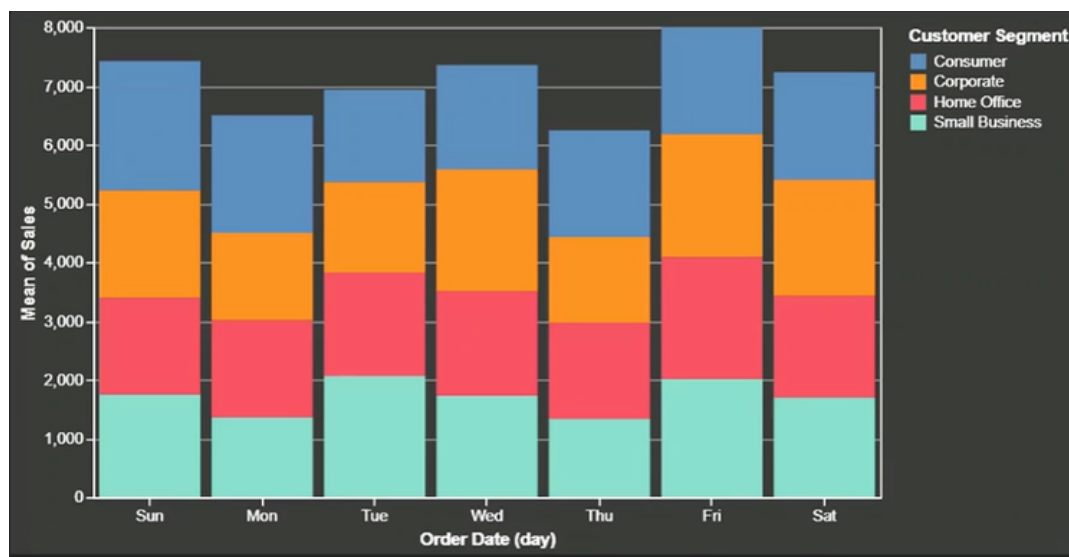
Все остальные виды графиков делаются по той же схеме.

Рассмотрим более тонкую настройку графиков. На столбчатой диаграмме выше видим, что ось `x` не очень хорошо настроена: есть две копии дня недели. Пропишем

для Altair тип этого поля — ординальное, также зададим ширину графика через свойство `properties`:

```
chart = alt.Chart(
    superstore_data
).mark_bar(
).encode(
    x='day(Order Date):O',
    y='mean(Sales)',
    color='Customer Segment'
).properties(
    width=500
)
chart
```

Теперь каждый день недели в единственном экземпляре:



Можно убрать горизонтальные линии сетки, поработав отдельно с осью `YO`. Идеология Altair следующая: если нас устраивают графики с настройками по умолчанию, мы можем использовать указание данных в виде строк после соответствующих параметров. Если требуется более тонкая настройка, необходимо использовать классы, встроенные в Altair.

В примере необходимо воспользоваться специальным классом `Altair` — `alt.Y`.

При этом важно указать, что будет выбираться кастомная ось:

```
chart = alt.Chart(
    superstore_data #выбор датасета
).mark_bar( #выбор глифов
).encode( #выбор кодирования
    x='day(Order Date):O',
    y=alt.Y('mean(Sales)', axis=alt.Axis(grid=False)),
    color='Customer Segment',
).properties(
    width=500
)
chart
```

Аналогично настраиваются подписи на осях.

`Altair` — подходящая под терминологию и идеологию визуализации данных библиотека. Чтобы получить график, достаточно прописать несколько строчек кода. Также можно воспользоваться тонкой настройкой внешнего вида графика и настроить его интерактивность.

3. Использование библиотек Python для визуализации. Библиотека `Seaborn`

Библиотека [Seaborn](#) обладает большим разнообразием графиков и может конкурировать с `Altair`. Библиотека `Seaborn` используется для исследовательского анализа данных. Она умеет делать большое количество стандартных статистических диаграмм, в том числе с доверительными интервалами.

Рассмотрим несколько кейсов на примере базы данных `superstore_data`.

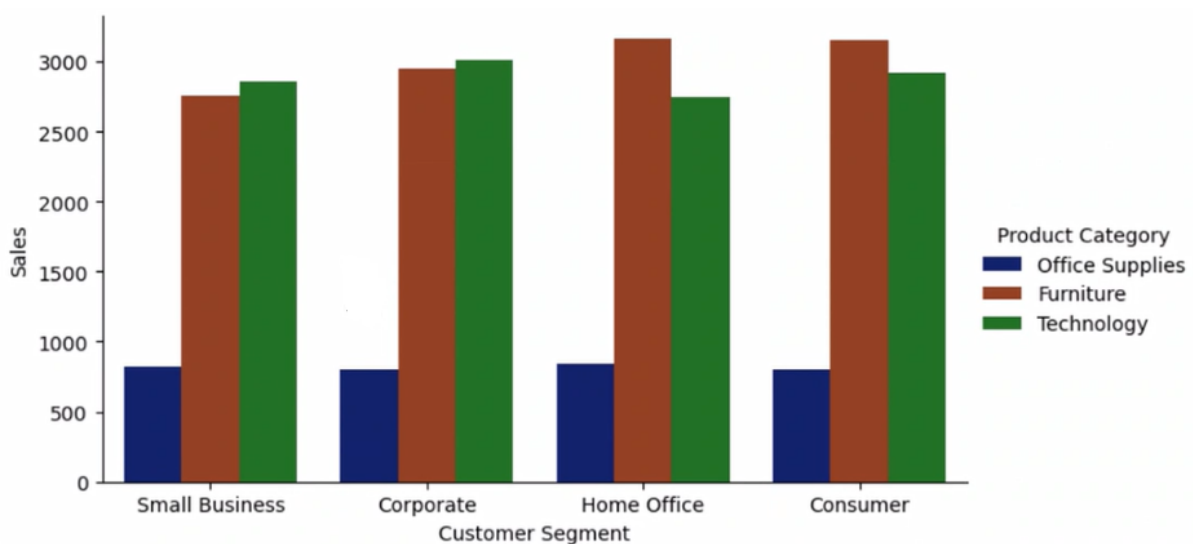
Импортируем данные для работы библиотеки и считаем файл, содержащий эти данные. Сразу же проверим типы столбцов:

```
from matplotlib import pyplot
import seaborn as sns
import pandas as pd
superstore_data = pd.read_excel('superstore.xlsx',
    parse_dates=['Order Date', 'Ship Date'])
superstore_data.dtypes
```


Построим столбчатую диаграмму. Для этого воспользуемся функцией `catplot`:

```
g = sns.catplot(  
    data=superstore_data  
    kind="bar",  
    x="Customer Segment",  
    y="Sales",  
    hue="Product Category",  
    palette="dark",  
    height=4,  
    aspect=16/9,  
    errorbar=None # отключаем доверительный интервал  
)
```

Результат:



В целом, синтаксис библиотеки похож на `Altair`.

Можно изменить названия осей:

```
g.set_axis_labels("Customer Segment", "Total Sales")  
g.legend.set_title("Category")
```

Чтобы сохранить график, сохраним файл стандартным для библиотеки `Matplotlib` методом. Такое возможно, поскольку библиотека `Seaborn` является надстройкой над более низкоуровневой библиотекой `Matplotlib`:

```
pyplot.savefig('seaborn-plot.svg')
```

Построим линейный график распределения продаж по дням. Проведем агрегацию данных по дате и категории товаров — для этого используем функцию библиотеки `Pandas` — `groupby()`. Агрегатной функцией будет функция суммирования `sum()`. Далее мы перестроим индекс, чтобы датафрейм содержал нужные нам столбцы `Order Date` и `Product Category`:

```
data = superstore_data[["Order Date", "Sales", "Product Category"]].groupby(["Order Date", "Product Category"]).aggregate(sum).reset_index()
data.head()
```

Результат:

	Order Date	Product Category	Sales
0	2009-01-01	Office Supplies	1052.84
1	2009-01-02	Furniture	5322.25
2	2009-01-02	Office Supplies	5779.62
3	2009-01-03	Furniture	960.76
4	2009-01-03	Office Supplies	504.06

Перестроим таблицу таким образом, чтобы столбец `Product Category` был разбит на несколько — по значениям, содержащимся в этом столбце. Значения в новых столбцах будут взяты из столбца `Sales`:

```
data_flattend = pd.pivot(
    data,
    index=['Order Date'],
    columns=['Product Category'],
    values='Sales'
```

```
) .reset_index().fillna(0)
```

Ниже результат. Нулями заполнены значения, где нет продаж. Мы сделали это с помощью функции `fillna`:

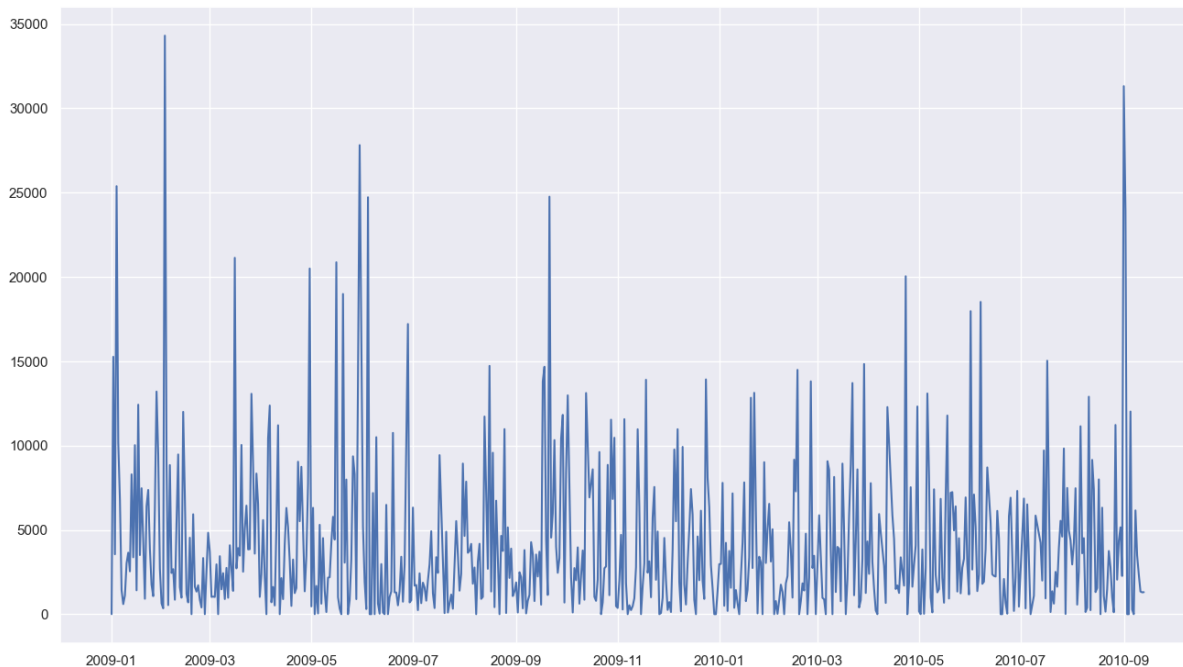
Product	Category	Order Date	Furniture	Office Supplies	Technology
0		2009-01-01	0.00	1052.84	0.0000
1		2009-01-02	5322.25	5779.62	0.0000
2		2009-01-03	960.76	504.06	30533.0355
3		2009-01-04	1039.56	214.69	0.0000
4		2009-01-05	28011.40	1171.57	10551.4340
...	
1413		2012-12-26	0.00	351.41	0.0000
1414		2012-12-27	759.94	16476.09	3883.4715
1415		2012-12-28	0.00	2721.18	0.0000
1416		2012-12-29	3711.04	13226.93	1936.4500
1417		2012-12-30	972.08	1523.04	672.9300

Далее воспользуемся функцией библиотеки `Seaborn` для построения линейного графика `lineplot`:

```
try:
    sns.set(rc = {'figure.figsize':(16,9)})
    sns.lineplot(
        x=data['Order Date'],
        y=data_flattened['Technology'],
    )
except TypeError:
    pass
```

Функция `set()` выставляет параметры графика, но, на самом деле, используются параметры `Matplotlib`.

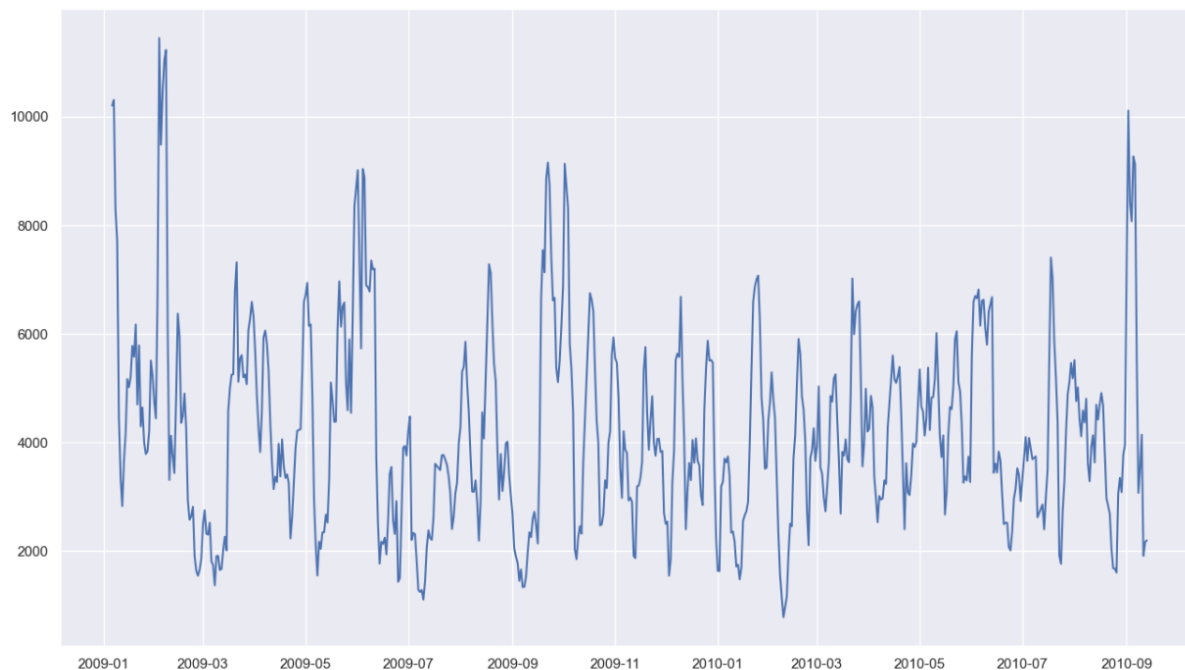
В результате получается довольно хаотичный и пилообразный график:



Чтобы его сгладить, воспользуемся функцией скользящего среднего:

```
try:
    sns.set(rc = {'figure.figsize': (16, 9)})
    sns.lineplot(
        x=data['Order Date'],
        y=data_flattened['Technology'].rolling(14).mean(),
    )
except TypeError:
    pass
```

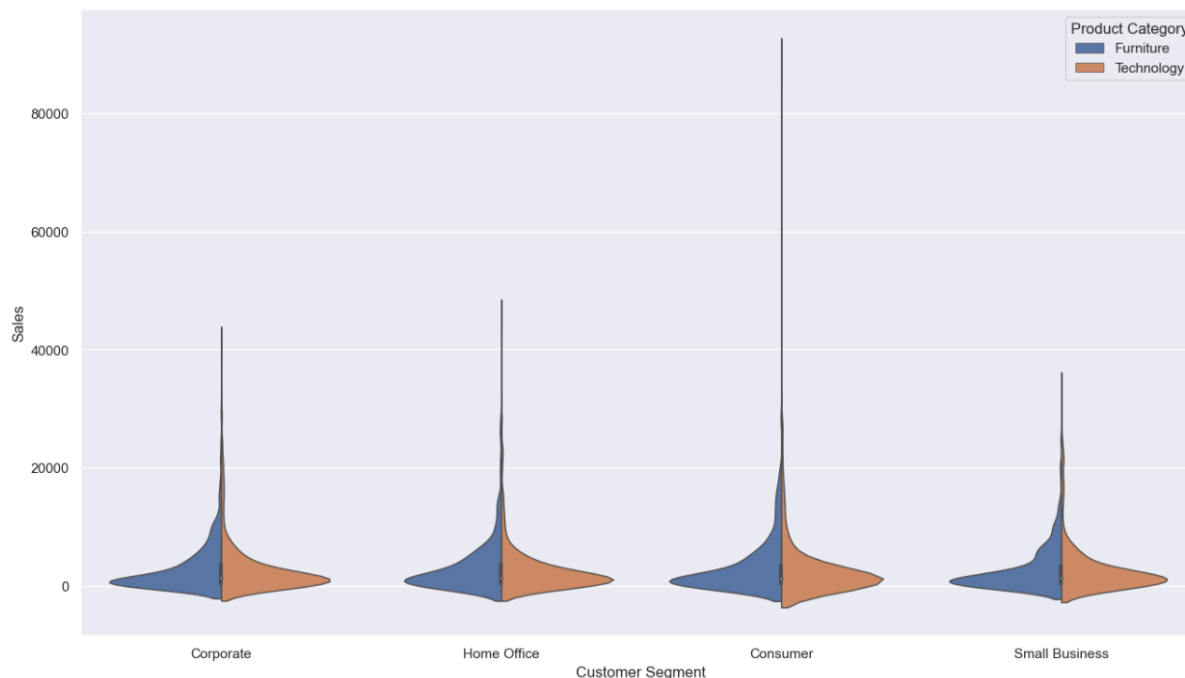
Результат:



Построим график сравнения распределений — violinplot. Мы разобьем данные по покупательскому сегменту. Изобразим распределение продаж. Дополнительно внутри каждого сегмента разобьем данные по категории товаров. Из категорий товаров возьмем только две — Technology и Furniture:

```
sns.violinplot(  
    data=superstore_data[(superstore_data['Product Category'] ==  
    'Technology') | (superstore_data['Product Category'] ==  
    'Furniture')],  
    x="Customer Segment",  
    y="Sales",  
    hue="Product Category",  
    split=True,  
    linewidth=1  
)
```

Часто на таком графике изображение распределений похоже на скрипку:



Библиотека `Seaborn` обычно используется в статистических контекстах — так она будет наиболее полезна, так как в нее зашито много стандартных для статистики графиков. Для интерактивных информационных панелей и презентаций лучше подойдут другие библиотеки, например, `Altair`.

Дополнительные материалы для самостоятельного изучения

1. [Визуализация данных — Yandex DataLens | Yandex Cloud — Сервисы](#)
2. [Example Gallery — Vega-Altair 5.0.0dev documentation](#)
3. [Example gallery — seaborn 0.12.2 documentation](#)