Классификация данных и задачи EDA

Все курсы > Анализ и обработка данных > Занятие 3

В рамках вводного курса мы начали знакомиться с основами описательной статистики, построили первые графики и узнали, что такое EDA. На первом занятии этого курса мы описали этапы решения задачи машинного обучения, и, в частности, рассмотрели основные составляющие исследовательского анализа данных.

Раздел по исследовательскому анализу данных состоит из двух занятий:

1. на первом **теоретическом** занятии мы рассмотрим различные классификации данных, их взаимосвязь с задачами EDA, а также коротко поговорим про четыре библиотеки Питона для построения визуализаций (Matplotlib, Pandas, Seaborn и Plotly Express);
2. на втором занятии мы **на практике** используем полученные знания для *системного анализа двух датасетов* (параллельно изучив новые инструменты описательной статистики); кроме того, мы поговорим про технические особенности *построения графиков в Matplotlib*.

Навыки проведения EDA приобретаются только с практикой, однако я надеюсь, что этот раздел станет достаточно надежной отправной точкой в этом процессе.

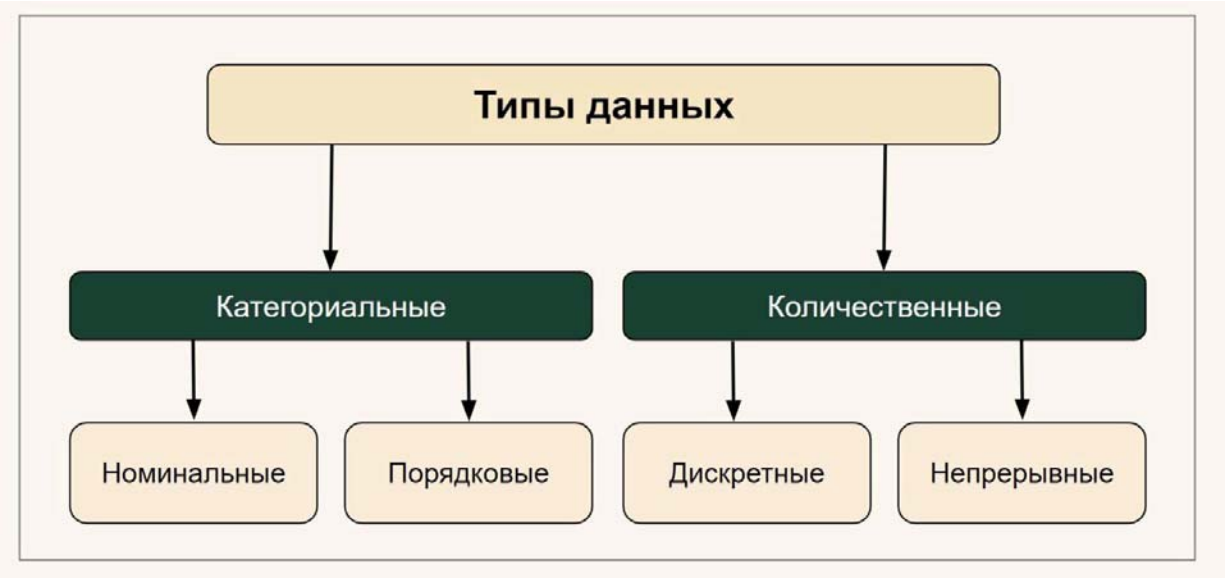
Кроме того, добавлю, что про **обработку данных** (data preprocessing), важный аспект построения модели, мы поговорим в третьем и четвертом разделах курса.

Откроем блокнот к этому занятию

Классификация данных

Качественные и количественные данные

Начнем с того, что более внимательно посмотрим на существующие типы данных. Их правильное понимание позволит более качественно проводить EDA.



Коротко опишем каждый из типов данных.

Качественные данные

Качественные или **категориальные данные** (categorical data) описывают принадлежность объекта к определенной группе.

Категориальные данные бывают двух видов: номинальные и порядковые.

Номинальные данные

Категории **номинальных данных** (nominal categorical data) не могут быть упорядочены, их сравнение не имеет смысла. Например, на прошлом занятии мы рассматривали марки автомобилей. Сравнение брендов в отрыве от их характеристик невозможно. То же самое касается типов товаров, жанров музыкальных произведений и т.д.

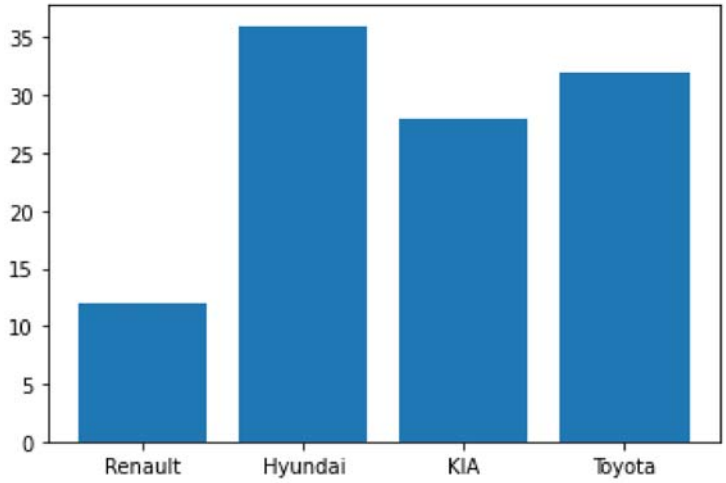
Приведем пример данных о количестве автомобилей различных марок у дилера.

```
# поместим данные о количестве автомобилей различных марок в дата
cars = pd.DataFrame({'model' : ['Renault', 'Hyundai', 'KIA', 'Toy
                     'stock' : [12, 36, 28, 32]})

cars
```

	model	stock
0	Renault	12
1	Hyundai	36
2	KIA	28
3	Toyota	32

Выведем эти данные с помощью столбчатой диаграммы.

```
# обратите внимание, что служебную информацию о графике можно убр
# так и с помощью точки с запятой ";"
plt.bar(cars.model, cars.stock);
```



Порядковые данные

При этом категориям **порядковых данных** (ordinal categorical data) свойственна внутренняя иерархия, их можно проранжировать. Например, к таким категориям относятся значения шкалы удовлетворенности потребителей (очень доволен, доволен, в целом доволен и т.д.) или возрастные категории (до 18 лет, от 18 до 24 и т.д.).

В качестве примера используем различные оценки, которые поставили потребители.

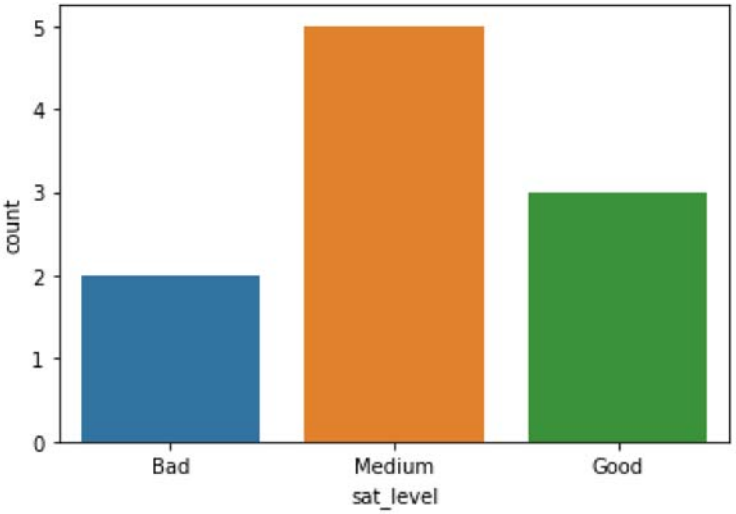
```
1  # соберем данные об уровне удовлетворенности десяти человек
2  satisfaction = pd.DataFrame({'sat_level': ['Good', 'Medium', 'Goo
3
4  satisfaction
```

	sat_level
0	Good
1	Medium
2	Good
3	Medium
4	Bad
5	Medium
6	Good
7	Medium
8	Medium
9	Bad

Визуализируем эти данные с помощью countplot (используем библиотеку Seaborn).

> Типы графиков и технические особенности их построения мы рассмотрим на следующем занятии.

```
# переведем данные в тип categorical
satisfaction.sat_level = pd.Categorical(satisfaction.sat_level,
                                        categories = ['Bad', 'Med
                                        ordered = True)

# построим столбчатую диаграмму типа countplot
# с количеством оценок в каждой из категорий
sns.countplot(x = 'sat_level', data = satisfaction);
```



Количественные данные

Количественные признаки, которые также называют **числовыми**, могут быть дискретными или непрерывными.

Дискретные данные

Дискретные данные (discrete numerical data) принимают строго определенные значения. Например, это может быть количество сотрудников на предприятии или количество бракованных деталей в произведенной партии.

Непрерывные данные

Непрерывные данные (continuous numerical data) всегда выражены неограниченным числом значений. Говоря более точно, они не имеют конечной точности измерений. Примером могут быть измерения каких-либо физических показателей (температура, плотность и т.д.).

Также напомню, что, как мы узнали на прошлом курсе, количественные данные характеризуются определенным распределением, дискретным или непрерывным.

Распределение Пуассона

В качестве примера дискретных данных возьмем новое для нас **распределение Пуассона** (Poisson discrete distribution), которое используется для моделирования частоты какого-либо события в определенный интервал времени.

$$X \sim Pois(\lambda)$$

Функция вероятности (pmf, probability mass function) распределения Пуассона описывается следующей формулой.

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda},,$$

где

- k — количество событий;
- λ — матожидание случайной величины (то есть среднее количество событий за интервал времени).

Предположим, что мы исследуем частоту поступающих в колл-центр звонков. Если известно, что в среднем каждую минуту в колл-центр поступает три звонка, и эта случайная величина следует распределению Пуассона, то мы можем воспользоваться функцией **np.random.poisson()** для моделирования этого процесса.

```
# передадим функции np.random.poisson()
# матожидание (lam) и желаемое количество экспериментов (size)
res = np.random.poisson(lam = 3, size = 1000)

# выведем первые 10 значений
res[:10]
```

```
array([4, 2, 2, 2, 2, 1, 3, 3, 2, 2])
```

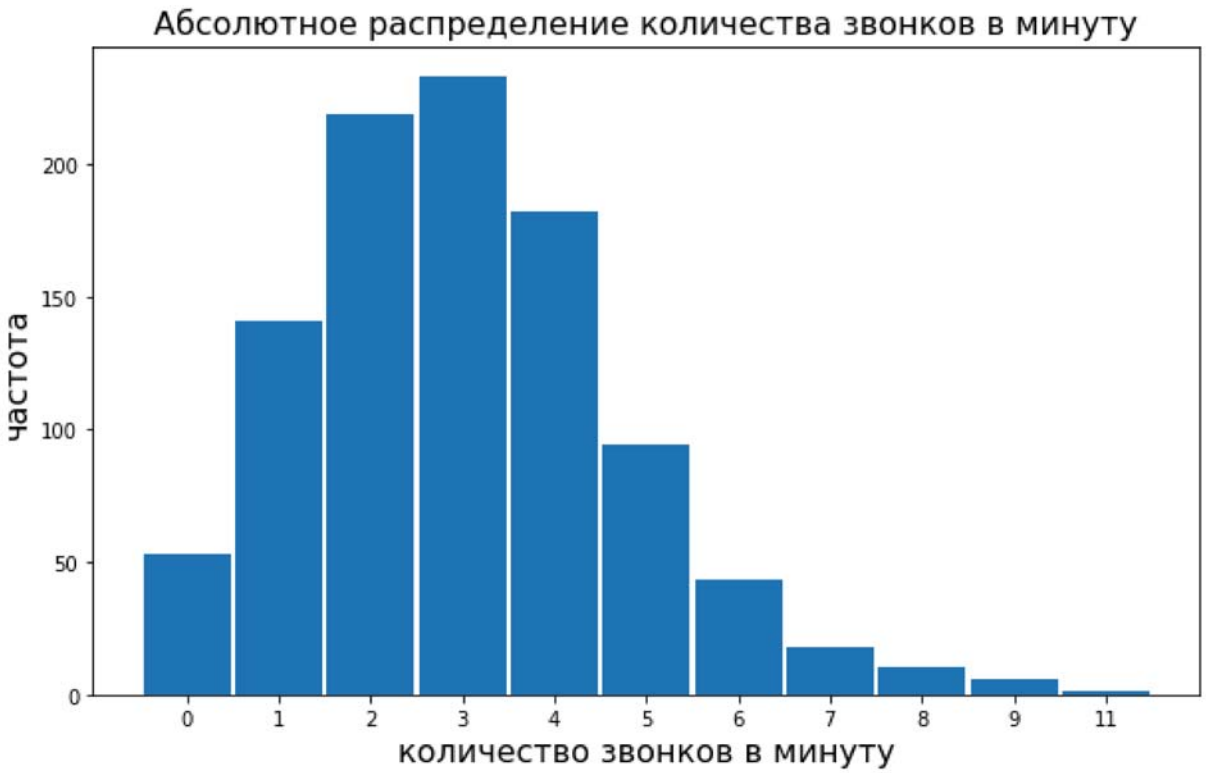
Посмотрим, сколько раз звонили в течение минуты и как часто встречалось такое количество звонков. Другими словами, посмотрим сколько раз могли звонить ноль раз, один раз, два раза и т.д.

```
# для этого воспользуемся знакомой нам функцией np.unique()
unique, counts = np.unique(res, return_counts = True)
unique, counts
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 11]),
 array([ 53, 141, 219, 233, 182,  94,  43,  18,  10,   6,   1]))
```

В частности, мы видим, что 53 раза в колл-центр не поступило ни одного звонка, 141 раз поступил один звонок и т.д.). Выведем эту информацию на графике.

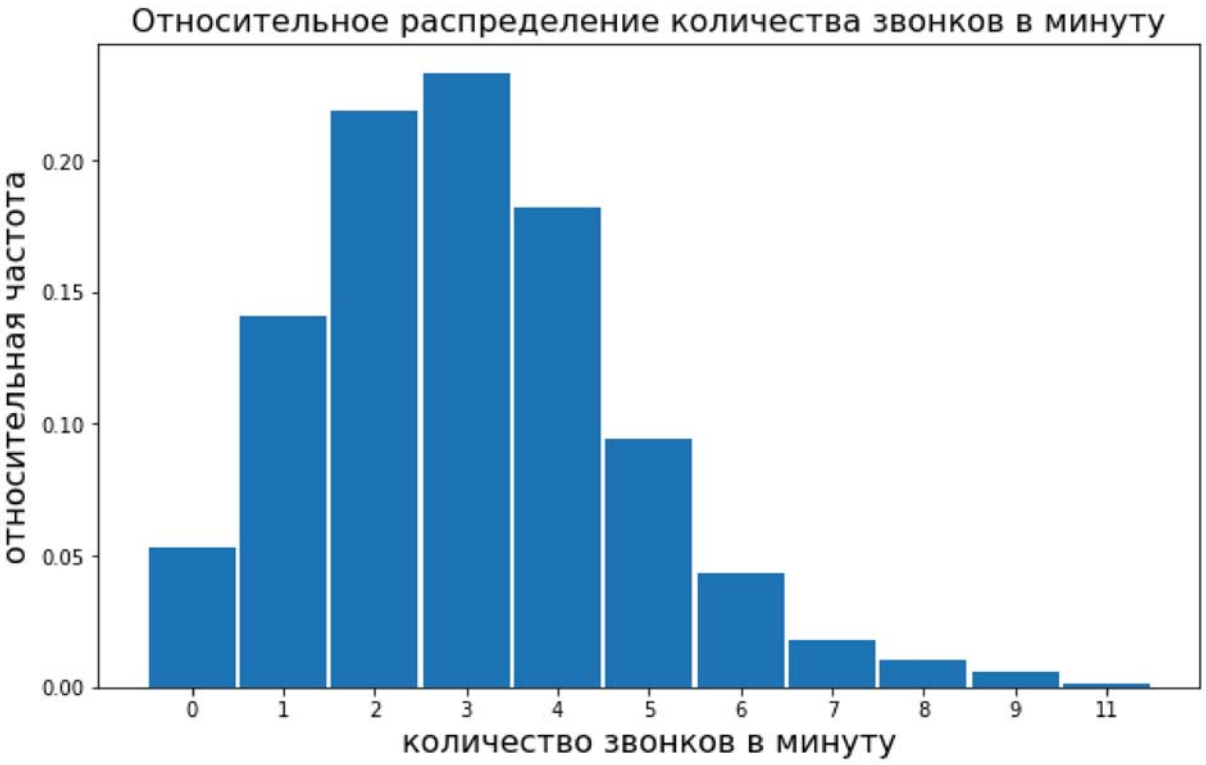
```
plt.figure(figsize = (10,6))
# перед построением графика переведем значения unique в тип str
plt.bar([str(x) for x in unique], counts, width = 0.95)
plt.title('Абсолютное распределение частоты звонков в минуту', fo
plt.xlabel('number of calls', fontsize = 16)
plt.ylabel('count', fontsize = 16);
```



Теперь посмотрим, как распределено количество звонков относительно их общего числа.

```
plt.figure(figsize = (10,6))
# для этого просто разделим количество звонков в каждом из столбц
plt.bar([str(x) for x in unique], counts / len(res), width = 0.95
plt.title('Относительное распределение количества звонков в минут
plt.xlabel('количество звонков в минуту', fontsize = 16)
```

```
6 plt.ylabel('относительная частота', fontsize = 16);
```



Возможно будет полезно вспомнить, какие есть способы преобразования списка чисел в список из строк.

Перейдем к расчету вероятности. Предположим, нас спрашивают, какова вероятность получить более шести звонков в минуту и какова вероятность получить от двух до шести звонков.

```
1 # разделим число наблюдений, в которых было более шести звонков,
2 np.round(len(res[res > 6])/len(res), 3)
```

```
1 0.035
```

По большому счету, мы посчитали количество наблюдений в столбцах, где было 7, 8, 9 и 11 звонков в минуту и разделили на их общее количество. Аналогично посчитаем относительное количество наблюдений в столбцах 2, 3, 4, 5 и 6.

```
1 np.round(len(res[res <= 6])/len(res) - len(res[res < 2])/len(res)
```

```
1 0.771
```

Как и в случае с другими распределениями, мы можем воспользоваться **библиотекой scipy** для того, чтобы построить график и рассчитать теоретическую вероятность случайной величины.

```
# построим график теоретической вероятности
from scipy.stats import poisson

# создадим последовательность целых чисел от 0 до 14
x = np.arange(15)
# передадим их в функцию poisson.pmf()
# mu в данном случае это матожидание (lambda из формулы)
f = poisson.pmf(x, mu = 3)

# построим график теоретического распределения, изменив для нагл
plt.figure(figsize = (10,6))
plt.bar([str(x) for x in x], f, width = 0.95, color = 'green')
plt.title('Теоретическое распределение количества звонков в мину
plt.xlabel('количество звонков в минуту', fontsize = 16)
plt.ylabel('относительная частота', fontsize = 16);
```

Теперь воспользуемся **функцией poisson.cdf()**, то есть *функцией распределения* (cumulative distribution function), для расчета теоретической вероятности получить более шести звонков в минуту и вероятности получить от двух до шести звонков.

Напомню, что функция распределения рассчитывает вероятность P того, что случайная величина X примет значение меньшее или равное k.

$$cdf = P(X \leq k)$$

Например, рассчитаем вероятность получения нуля звонков или одного звонка в час.

```
# на графике это сумма площадей первого и второго столбцов
poisson.cdf(1, 3).round(3)
```

```
0.199
```

Теперь перейдем к решению исходной задачи.

```
# найдем площадь столбцов до шести звонков в минуту включительно
# и вычтем результат из единицы
np.round(1 - poisson.cdf(6, 3), 3)
```

```
0.034
```

```
# для выполнения второго задания вычтем площадь столбцов ноль и о
# из площади столбцов до шестого включительно
np.round(poisson.cdf(6, 3) - poisson.cdf(1, 3), 3)
```

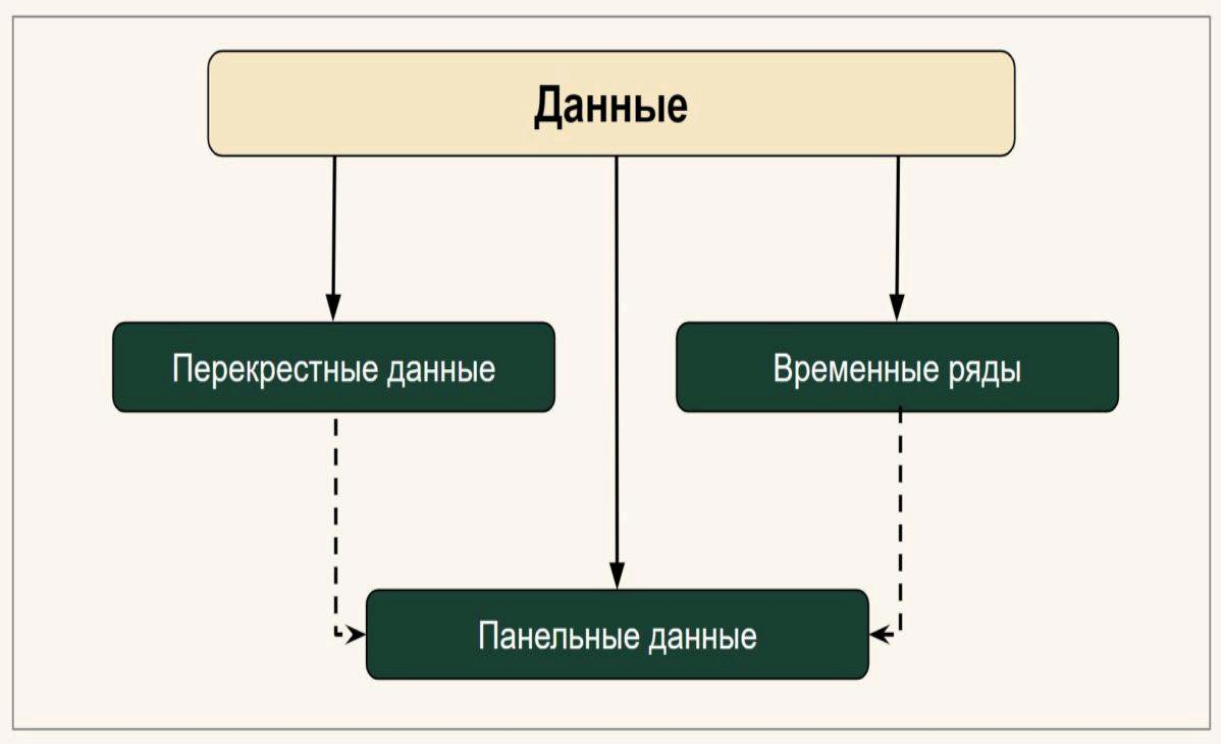
```
0.767
```

Как мы видим, полученная ранее эмпирическая вероятность достаточно близка к теоретическим расчетам. Напомню, это возможно благодаря закону больших чисел.

Примером непрерывных количественных данных может служить рост человека, который, как мы уже знаем, следует нормальному распределению.

Перекрестные данные, временные ряды и панельные данные

Еще одной классификацией данных является разделение на перекрестные данные, временные ряды и панельные данные.



Перекрестные данные

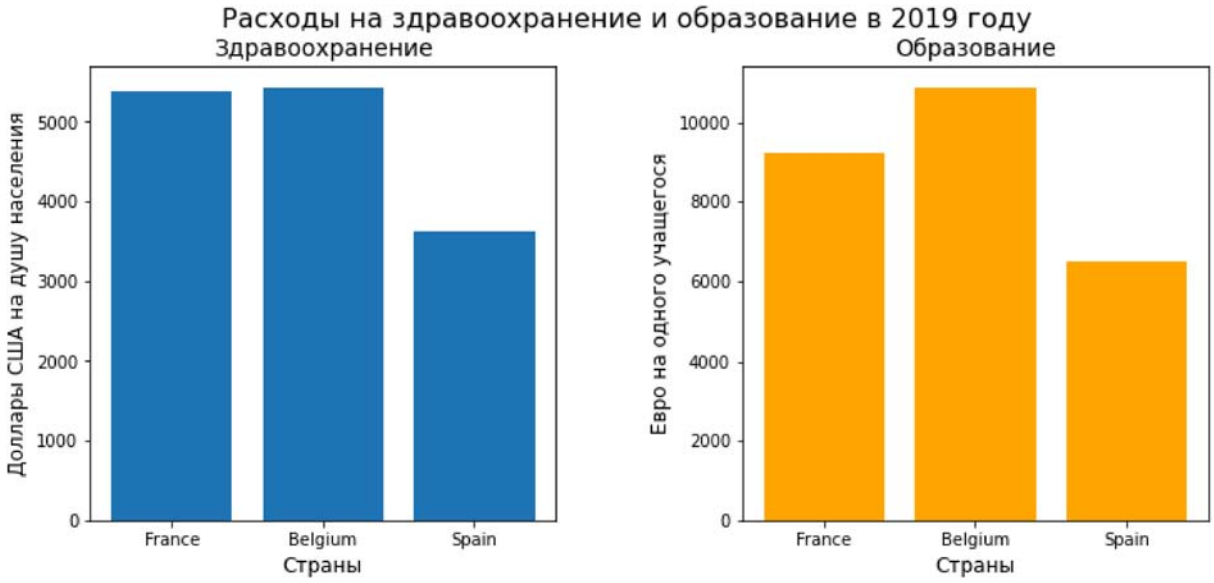
Примером **перекрестных данных** (cross-sectional data) могут быть расходы на здравоохранение и образование на душу населения и одного учащегося соответственно в трех странах мира в 2019 году.

```
# создадим датафрейм с данными по Франции, Бельгии и Испании
csect = pd.DataFrame({'countries' : ['France', 'Belgium', 'Spain'
                      'healthcare' : [4492, 5428, 3616],
                      'education': [9210, 10869, 6498]})

# посмотрим на результат
csect
```

	countries	healthcare	education
0	France	5376	9210
1	Belgium	5428	10869
2	Spain	3616	6498

Теперь выведем данные на графике.

```python
# зададим размер фигуры для обоих графиков
plt.figure(figsize = (12,5))

# используем функцию plt.subplot() для создания первого графика
# передаваемые параметры: nrows, ncols, index
plt.subplot(1, 2, 1)
# построим столбчатую диаграмму для здравоохранения
plt.bar(csect.countries, csect.healthcare)
plt.title('Здравоохранение', fontsize = 14)
plt.xlabel('Страны', fontsize = 12)
plt.ylabel('Доллары США на душу населения', fontsize = 12)

# создадим второй график (index = 2)
# параметры можно передать одним числом
plt.subplot(122)
# построим столбчатую диаграмму для образования
plt.bar(csect.countries, csect.education, color = 'orange')
plt.title('Образование', fontsize = 14)
plt.xlabel('Страны', fontsize = 12)
plt.ylabel('Евро на одного учащегося', fontsize = 12)

# отрегулируем пространство между графиками
plt.subplots_adjust(wspace = 0.4)

# зададим общий график
plt.suptitle('Расходы на здравоохранение и образование в 2019 го

# выведем результат
plt.show()
```



Временные ряды

Расходы на здравоохранение и образование *одной* страны, но на протяжении десяти лет будут считаться **временными рядами** (time-series).

```
# создадим временной ряд расходов на здравоохранение во Франции
tseries = pd.DataFrame({'year' :       [2010, 2011, 2012, 2013,
                        'healthcare' : [4598, 4939, 4651, 4902,

# превратим год в объект datetime
tseries.year = pd.to_datetime(tseries.year, format = '%Y')
# и сделаем этот столбец индексом
tseries.set_index('year', drop = True, inplace = True)

# посмотрим на результат
tseries
```

year	healthcare
2010-01-01	4598
2011-01-01	4939
2012-01-01	4651
2013-01-01	4902
2014-01-01	4999
2015-01-01	4208
2016-01-01	4268
2017-01-01	4425
2018-01-01	4690
2019-01-01	4492

Выведем эти данные с помощью линейного графика.

```
# зададим размер графика
plt.figure(figsize = (12,5))
```

```
# дополнительно укажем цвет, толщину линии и вид маркера
plt.plot(tseries, color = 'green', linewidth = 2, marker = 'o')

# добавим подписи к осям и заголовок
plt.xlabel('Годы', fontsize = 14)
plt.ylabel('Доллары США', fontsize = 14)
plt.title('Расходы на здравоохранение на душу населения во Франц

# выведем результат
plt.show()
```

Панельные данные

Теперь, если объединить анализ расходов на здравоохранение по нескольким странам (перекрестные данные) с измерением этих данных во времени (временные ряды), мы получим то, что принято называть **панельными данными** (panel data).

Создание датафрейма с панельными данными

Для создания датафрейма с панельными данными вначале передадим в **функцию pd.DataFrame()** словарь, в котором будет один столбец. В этом столбце поочередно укажем расходы на здравоохранение на душу населения для Франции, Бельгии и Испании в период с 2015 по 2019 годы.

```
# первые пять цифр относятся к Франции, вторые пять - к Бельгии,
pdata = pd.DataFrame({'healthcare' : [4208, 4268, 4425, 4690, 449
```

Теперь подготовим кортежи для создания иерархического индекса и передадим их в функцию **pd.MultiIndex.from_tuples()**. После этого обновим атрибут **index** и выведем результат.

```
# создадим кортежи для иерархического индекса
rows = [('France', '2015'),
        ('France', '2016'),
        ('France', '2017'),
        ('France', '2018'),
        ('France', '2019'),
        ('Belgium', '2015'),
        ('Belgium', '2016'),
        ('Belgium', '2017'),
        ('Belgium', '2018'),
        ('Belgium', '2019'),
        ('Spain', '2015'),
        ('Spain', '2016'),
        ('Spain', '2017'),
        ('Spain', '2018'),
        ('Spain', '2019')]

# передадим кортежи в функцию pd.MultiIndex.from_tuples(),
# указав названия уровней индекса
custom_multindex = pd.MultiIndex.from_tuples(rows, names = ['cou

# сделаем custom_multindex индексом датафрейма с панельными данн
pdata.index = custom_multindex

# посмотрим на результат
pdata
```

		healthcare
country	year	
France	2015	4208
	2016	4268
	2017	4425
	2018	4690
	2019	4492
Belgium	2015	4290
	2016	4323
	2017	4618
	2018	4913
	2019	4960
Spain	2015	2349
	2016	2377
	2017	2523
	2018	2736
	2019	2542

Как мы видим, панельные данные — это *трехмерный массив*, в котором первым измерением являются субъекты наблюдения (в нашем случае, страны), вторым — временные периоды, а третьим — признаки.

Таким образом, можно сказать, что перекрестные данные и временные ряды являются частным случаем панельных данных для одного временного периода или одного субъекта исследования соответственно.

> Интересно, что название библиотеки Pandas происходит как раз от словосочетания «панельные данные» (panel data).

Остается их визуализировать.

Визуализация панельных данных

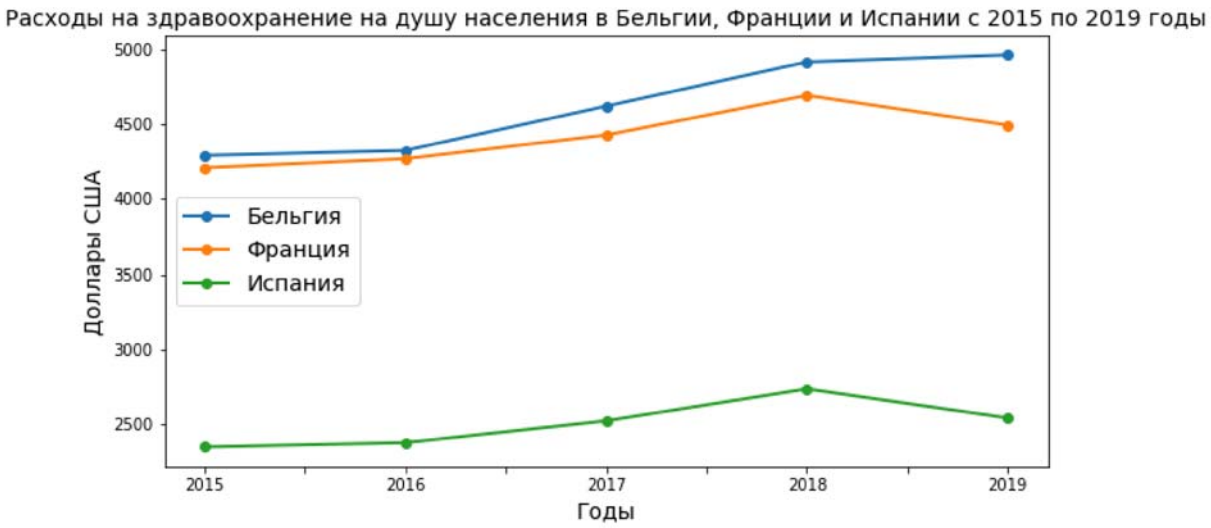
Вначале превратим трехмерный массив обратно в двумерный с помощью **метода .unstack()**.

```
# сделаем данные по странам (index level = 0) отдельными столбцам
pdata_unstacked = pdata.healthcare.unstack(level = 0)

# метод .unstack() выстроит столбцы в алфавитном порядке
pdata_unstacked
```

country	Belgium	France	Spain
year			
2015	4290	4208	2349
2016	4323	4268	2377
2017	4618	4425	2523
2018	4913	4690	2736
2019	4960	4492	2542

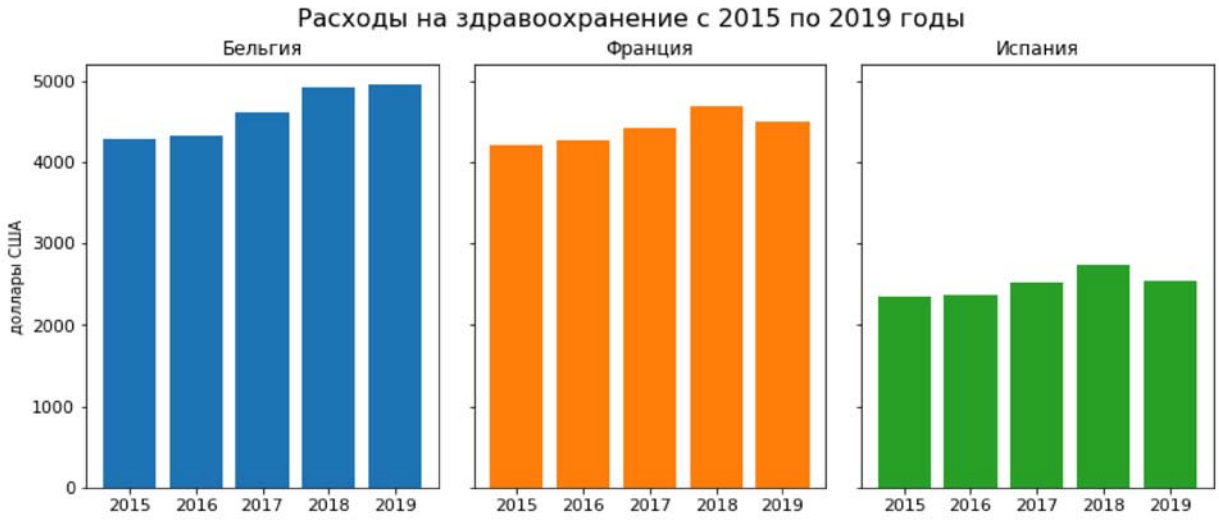
Теперь мы можем вывести эти данные на одном графике.

```
# зададим размер графика
plt.figure(figsize = (10,5))

# построим три кривые
pdata_unstacked.Belgium.plot(linewidth = 2, marker = 'o', label
pdata_unstacked.France.plot(linewidth = 2, marker = 'o', label =
pdata_unstacked.Spain.plot(linewidth = 2, marker = 'o', label =

# дополним подписями к осям, заголовком и легендой
plt.xlabel('Годы', fontsize = 14)
plt.ylabel('Доллары США', fontsize = 14)
plt.title('Расходы на здравоохранение на душу населения в Бельги
plt.legend(loc = 'center left', prop = {'size': 14})

plt.show()
```



Кроме того, с помощью параметра `subplots = True` **метода .plot()** мы можем разместить эти данные на трех подграфиках (в этот раз используем столбцатую диаграмму).

```
pdata_unstacked.plot(kind = 'bar', # создадим столбчатые диаграм
                     subplots = True, # которые разместим на под
                     layout = (1, 3), # на сетке из одной строки
                     rot = 0, # сделаем подписи по оси x горизон
                     figsize = (13, 5), # зададим внешний размер
                     sharey = True, # важно иметь одинаковую шка
                     fontsize = 11, # зададим размер шрифта един
                     width = 0.8, # ширину столбцов
                     xlabel = '', # кроме того, уберем подписи к
                     ylabel = 'доллары США', # поставим подпись
                     legend = None, # уберем легенду
                     title = ['Бельгия', 'Франция', 'Испания'])

# отрегулируем ширину между графиками
plt.subplots_adjust(wspace = 0.1)

# добавим общий заголовок
plt.suptitle('Расходы на здравоохранение с 2015 по 2019 годы', f
```



Замечу, что в последующих разделах мы будем рассматривать только перекрестные данные и временные ряды.

Одномерный и многомерный анализ

В соответствии с еще одной классификацией данные предполагают одномерный и многомерный анализ.

При **одномерном анализе** (univariate analysis) мы сосредоточены на изучении одного единственного показателя. **Многомерный анализ** (multivariate analysis) предполагает, что мы изучаем сразу несколько признаков.

Представленные выше перекрестные данные с расходами на здравоохранение *и* образование — это пример многомерного анализа.

Временной ряд и панельные данные, касающиеся только расходов на здравоохранение — это одномерный анализ.

Для наглядности приведем пример многомерных временных рядов (multivariate time series) и панельных данных (multivariate panel data).

Многомерный временной ряд

Создадим временной ряд расходов на здравоохранение во Франции на душу населения в долларах с 2010 по 2019 годы и приведем процент ВВП, потраченный на образование, за аналогичный период.

```
# подготовим данные
tseries_mult = pd.DataFrame({'year' :       [2010, 2011, 2012, 2
                             'healthcare' : [4598, 4939, 4651, 4
                             'education' :  [5.69, 5.52, 5.46, 5

# превратим год в объект datetime
tseries_mult.year = pd.to_datetime(tseries_mult.year, format = '
# и сделаем этот столбец индексом
tseries_mult.set_index('year', drop = True, inplace = True)

# посмотрим на результат
tseries_mult
```

year	healthcare	education
2010-01-01	4598	5.69
2011-01-01	4939	5.52
2012-01-01	4651	5.46
2013-01-01	4902	5.50
2014-01-01	4999	5.51
2015-01-01	4208	5.46
2016-01-01	4268	5.48
2017-01-01	4425	5.45
2018-01-01	4690	5.41
2019-01-01	4492	6.62

Многомерные панельные данные

Теперь добавим расходы на образование в панельные данные.

```
# вначале создадим датафрейм с данными расходов на здравоохранен
pdata_mult = pd.DataFrame({'healthcare, per capita' : [4208, 426
                           'education, % of GDP'  : [5.46, 5.48,

# создадим кортежи для иерархического индекса
rows = [('France', '2015'),
        ('France', '2016'),
        ('France', '2017'),
        ('France', '2018'),
        ('France', '2019'),
        ('Belgium', '2015'),
        ('Belgium', '2016'),
        ('Belgium', '2017'),
        ('Belgium', '2018'),
        ('Belgium', '2019'),
        ('Spain', '2015'),
        ('Spain', '2016'),
        ('Spain', '2017'),
        ('Spain', '2018'),
        ('Spain', '2019')]

```

```
# передадим кортежи в функцию pd.MultiIndex.from_tuples(),
# указав названия уровней индекса
custom_multindex = pd.MultiIndex.from_tuples(rows, names = ['cou

# сделаем custom_multindex индексом датафрейма с панельными данн
pdata_mult.index = custom_multindex

# посмотрим на результат
pdata_mult
```

		healthcare, per capita	education, % of GDP
country	year		
France	2015	4208	5.46
	2016	4268	5.48
	2017	4425	5.45
	2018	4690	5.41
	2019	4492	6.62
Belgium	2015	4290	6.45
	2016	4323	6.46
	2017	4618	6.43
	2018	4913	6.38
	2019	4960	6.40
Spain	2015	2349	4.29
	2016	2377	4.23
	2017	2523	4.21
	2018	2736	4.18
	2019	2542	4.26

> Таким образом, очевидно, что все три предложенные классификации могут описывать одни и те же данные.

Теперь давайте свяжем описанные выше классификации с задачами EDA.

Задачи EDA

В ходе исследовательского анализа данных нам необходимо решить три основные задачи:

- описать данные;
- найти различия;
- выявить закономерности.

Подробнее поговорим про каждую из них.

Описание данных

Описание данных предполагает одномерный анализ, потому что мы каждый раз работаем только с одним признаком. В **категориальных данных** мы, прежде всего, находим уникальные категории и оцениваем количество в каждой из них.

Анализ **количественной переменной** предполагает оценку среднего, стандартное отклонение, диапазон, персентили и другие показатели, характеризующие распределение числового признака.

Нахождение различий

Нахождение различий — это многомерный анализ, потому что в нем учавствуют два или более признаков. В частности, мы можем выявить **отличия одного качественного признака под влиянием другого**. Например, в датасете «Титаник», к которому мы вновь обратимся на следующем занятии, можно посмотреть, как связана выживаемость пассажира с принадлежностью к первому, второму и третьему классам.

Возможно нахождение **различий количественного признака в разрезе определенной категории**. Например, в том же датасете мы посмотрим на распределение возраста в зависимости от пола.

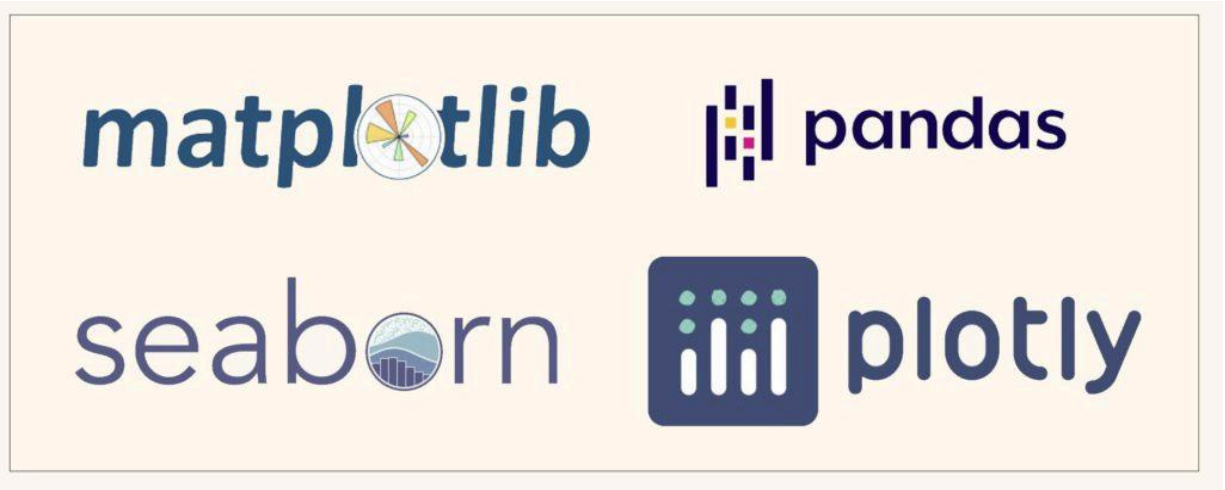
Выявление закономерностей

Закономерности или взаимосвязи в данных могут быть выявлены между **двумя количественными признаками**. Например, в датасете Tips («чаевые»), который мы также будем использовать для иллюстрации процесса EDA, мы можем попытаться обнаружить взаимосвязь между размером счета и оставленными чаевыми.

Теперь про технические средства, которые нам понадобятся.

Какие библиотеки мы будем использовать

В рамках сегодняшнего занятия мы детально познакомимся с четырьмя библиотеками.



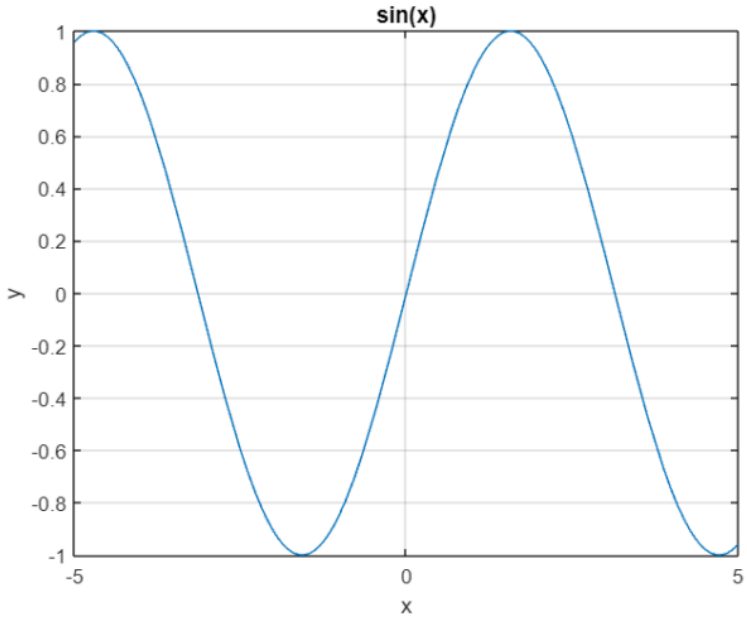
Библиотека Matplotlib

Стиль MATLAB

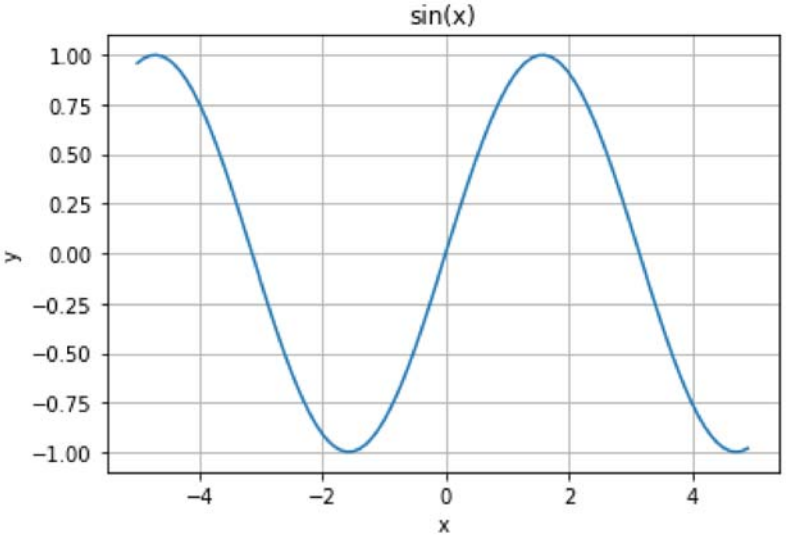
Библиотека Matplotlib, а точнее, ее модуль pyplot позволяет строить графики через команды, схожие с командами языка MATLAB. В этом случае для построения графика мы записываем серию команд, преобразовывающих каким-либо образом ту или иную часть графика.

Сравните код в MATLAB для построения графика синусоиды и соответствующий код на Питоне с использованием библиотеки Matplotlib.

```
fplot(@sin)
grid on
title('sin(x)')
xlabel('x');
ylabel('y');
```




```
# зададим последовательность от -5 до 5 с шагом 0,1
x = np.arange(-5, 5, 0.1)

# построим график синусоиды
plt.plot(x, np.sin(x))

# зададим заголовок, подписи к осям и сетку
plt.title('sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid();
```



Объектно-ориентированный подход

Кроме того, библиотека Matplotlib предполагает использование принципов объектно-ориентированного программирования. В Matplotlib есть два класса:

- **figure** — класс-контейнер для хранения подграфиков;
- **axes** — подграфики внутри контейнера.

Рассмотрим эти классы на практике.

```
# создадим объект класса figure
fig = plt.figure()

# и посмотрим на его тип
print(type(fig))
```

```
<class 'matplotlib.figure.Figure'>
<Figure size 432x288 with 0 Axes>
```

Обратите внимание, что пока у объекта fig ‘0 axes’, то есть контейнер мы создали, а подграфики — еще нет.

```
# применим метод .add_subplot() для создания подграфика (объекта
# напомню, что первые два параметра задают количество строк и сто
# третий параметр - это индекс (порядковый номер подграфика)
ax = fig.add_subplot(2, 1, 1)

```

```
# посмотрим на тип этого объекта
print(type(ax))
```

```
<class 'matplotlib.axes._subplots.AxesSubplot'>
```

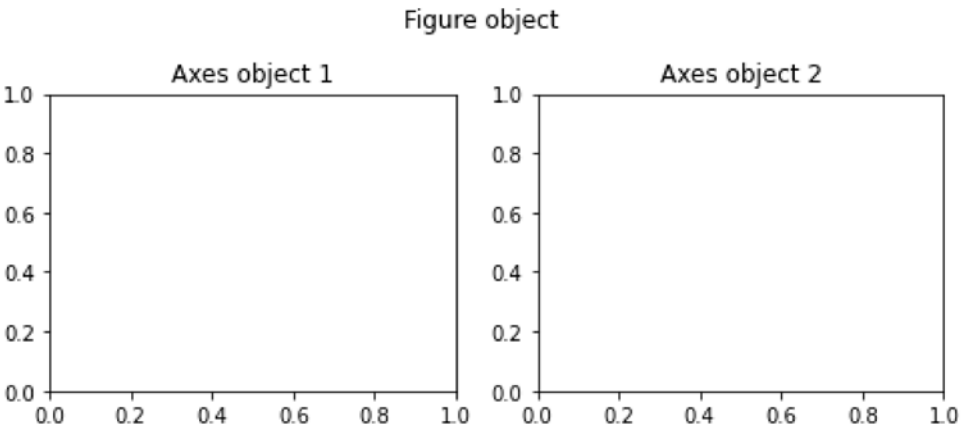
Выведем информацию о количестве подграфиков с помощью **атрибута number** объекта **fig**.

```
fig.number
```

```
1
```

Приведем пример наполнения подграфиков.

```
# вначале создаем объект figure, указываем размер объекта
fig = plt.figure(figsize = (8,6))
# и его заголовок с помощью метода .suptitle()
fig.suptitle('Figure object')
# можно и plt.suptitle('Figure object')

# внутри него создаем первый объекта класса axes
ax1 = fig.add_subplot(2, 2 ,1)
# к этому объекту можно применять различные методы
ax1.set_title('Axes object 1')

# и второй (напомню, параметры можно передать без запятых)
ax2 = fig.add_subplot(2, 2, 2)
ax2.set_title('Axes object 2')

# выведем результат
plt.show()
```



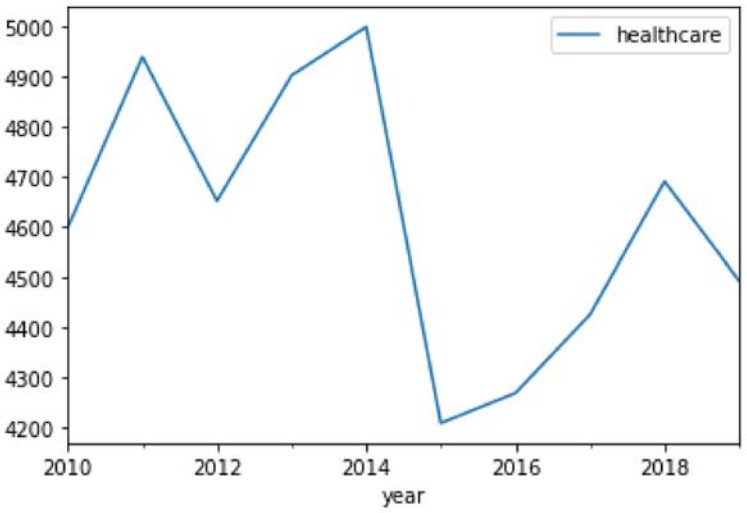
Способы создания подграфиков, а также настройку отдельных элементов графиков мы разберем на следующем занятии.

Библиотека Pandas

Когда мы работаем с датафреймами, зачастую бывает удобно строить графики непосредственно в библиотеке Pandas, в частности, с помощью **метода .plot()**. «Под капотом» Pandas использует объекты библиотеки Matplotilb для построения графиков.

```
# в этом несложно убедиться с помощью функции type()
type(tseries.plot())
```

```
matplotlib.axes._axes.Axes
```



Библиотека Seaborn

Библиотека Seaborn представляет собой надстройку над библиотекой Matplotlib и прекрасно интегрирована в датафреймы. Seaborn позволяет строить графики, используя меньшее количество кода, чем Matplotlib.

Примеры создания графиков с seaborn приведены выше.

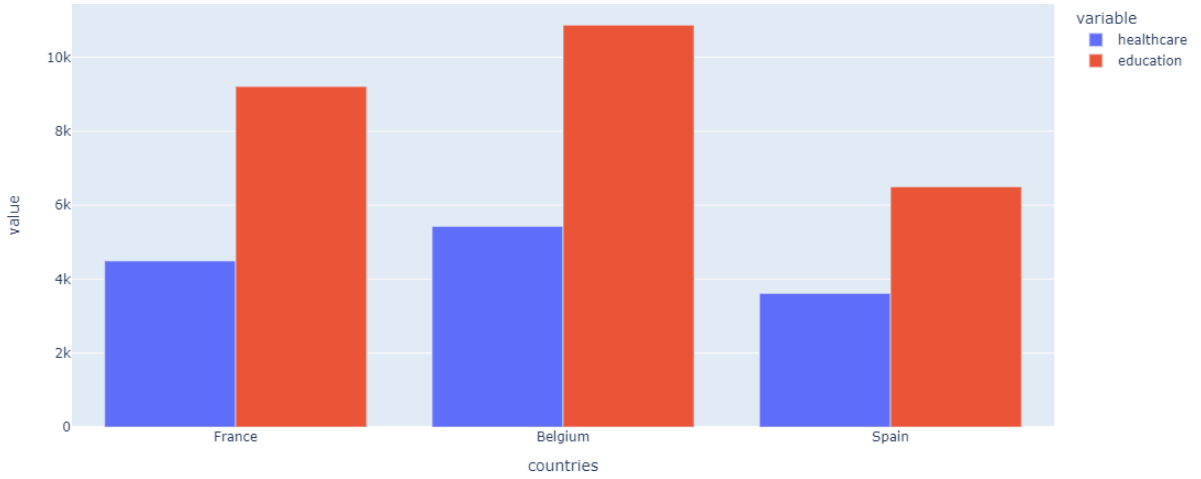
> В целом Pandas и Seaborn удобно использовать для быстрого построения визуализаций и исследования данных. Matplotlib чаще используется в тех случаях, когда необходима гораздо более тонкая настройка графиков.

Модуль Plotly Express

Модуль **Plotly Express** библиотеки Plotly отличается тем, что позволяет строить интерактивные графики. По сути, это небольшие веб-приложения. Plotly Express обычно импортируется как `px`.

Рассмотрим несложный пример построения графика.

```
# по оси x разместим страны, по оси y - признаки
# параметр barmode = 'group' указывает,
# что столбцы образования и здравоохранения нужно разместить рядо
# а не внутри одного столбца (stacked)
px.bar(csect,
       x = 'countries',
       y = ['healthcare', 'education'],
       barmode = 'group')
```



Интерактивный график можно посмотреть в блокноте к занятию.

Подведем итог

На сегодняшнем занятии мы рассмотрели различные классификации данных. В частности, мы вновь обратились к качественным и количественным переменным, посмотрели на различия между перекрестными данными, временными рядами и панельными данными. Мы также узнали про многомерный и одномерный анализ данных.

Кроме того, мы поговорили про задачи исследовательского анализа данных и перечислили библиотеки, позволяющие визуализировать данные.

В качестве дополнительного материала мы детально рассмотрели распределение Пуассона.

Теперь перейдем к практической части раздела и применим системный подход к анализу двух датасетов.