Машинное обучение

Ошибки в данных

Материалы > Анализ и обработка данных

На прошлом занятии, посвященном практике EDA, мы работали с «чистыми данными», то есть такими данными, в которых нет ни ошибок, ни пропущенных значений. К сожалению, так бывает далеко не всегда.

Сегодня мы научимся очищать данные от дубликатов, неверных и плохо отформатированных значений, а также исправлять ошибки в дате и времени. На следующем занятии мы поговорим про работу с пропусками.

Откроем блокнот к этому занятию

Ошибки в данных могут встречаться по многим причинам. Они могут быть связаны с человеческим фактором, например, простой невнимательностью или быть вызваны сбоями в работе записывающего какие-либо показатели оборудования.

В качестве примера мы будем использовать несложный датасет, в котором содержатся данные за 2019 год об отдельных финансовых показателях сети магазинов одежды, представленной в нескольких городах мира. В частности, нам доступна следующая информация:

- **month** — за какой месяц сделана запись;
- **profit** — *прибыль* по сети;
- **MoM** — изменение *выручки* (revenue) сети по отношению к предыдущему месяцу;
- **high** — магазин с наибольшей *маржинальностью* (margin) продаж.

Создадим датафрейм из словаря.

```
financials = pd.DataFrame({'month' :   ['01/01/2019', '01/02/2019', '01/03/2019', '01/0
                           'profit' : ['1.20$', '1.30$', '1.25$', '1.25$', '1.27$', '1.
                           'MoM' :     [0.03, -0.02, 0.01, 0.02, -0.01, -0.015, 0.017,
                           'high' :    ['Dubai', 'Paris', 'singapour', 'singapour', 'mo
                           })

financials
```

	month	profit	MoM	high
0	01/01/2019	1.20$	0.030	Dubai
1	01/02/2019	1.30$	-0.020	Paris
2	01/03/2019	1.25$	0.010	singapour

3	01/03/2019	1.25$	0.020	singapour
4	01/04/2019	1.27$	-0.010	moscow
5	01/05/2019	1.11$	-0.015	Paris
6	01/06/2019	1.23$	0.017	Madrid
7	01/07/2019	1.20$	0.040	moscow
8	01/08/2019	1.31$	0.020	london
9	01/09/2019	1.24$	0.010	london
10	01/10/2019	1.18$	0.000	Moscow
11	01/11/2019	1.17$	-0.010	Rome
12	01/12/2019	1.23$	2.000	madrid
13	01/12/2019	1.23$	2.000	madrid

Воспользуемся **методом .info()** для получения общей информации о датасете.

```
financials.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   month   14 non-null     object 
 1   profit  14 non-null     object 
 2   MoM     14 non-null     float64
 3   high    14 non-null     object 
dtypes: float64(1), object(3)
memory usage: 576.0+ bytes
```

Перейдем к поиску ошибок в данных.

Дубликаты

Поиск дубликатов

Заметим, что хотя данные представлены за 12 месяцев, в датафрейме тем не менее содержится 14 значений. Это заставляет задуматься о **дубликатах** (duplicates) или повторяющихся значениях. Воспользуемся **методом .duplicated()**. На выходе мы получим логический массив, в котором повторяющееся значение обозначено как True.

```
# keep = 'first' (параметр по умолчанию)
# помечает как дубликат (True) ВТОРОЕ повторяющееся значение
financials.duplicated(keep = 'first')
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
```

```
12    11    False
13    12    False
14    13     True
15    dtype: bool
```

```
# keep = 'last' соответственно считает дубликатом ПЕРВОЕ повторяющееся значение
financials.duplicated(keep = 'last')
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12     True
13    False
dtype: bool
```

Результат метода **.duplicated()** можно использовать как фильтр.

```
# с параметром keep = 'last' будет выведено наблюдение с индексом 12
financials[financials.duplicated(keep = 'last')]
```

	month	profit	MoM	high
12	01/12/2019	1.23$	2.0	madrid

Также заметим, что если смотреть по месяцам, у нас две дублирующихся записи, а не одна. В частности, повторяется запись не только за декабрь, но и за март. Проверим это с помощью параметра **subset**.

```
# с помощью параметра subset мы ищем дубликаты по конкретным столбцам
financials.duplicated(subset = ['month'])
```

```
0     False
1     False
2     False
3      True
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13     True
14    False
dtype: bool
```

```
# посчитаем количество дубликатов по столбцу month
financials.duplicated(subset = ['month']).sum()
```

```
2
```

Создадим новый фильтр и выведем дубликаты по месяцам.

```
# укажем параметр keep = 'last', больше доверяя, таким образом,
# последнему записанному за конкретный месяц значению
```

```
3 financials[financials.duplicated(subset = ['month'], keep = 'last')]
```

	month	profit	MoM	high
2	01/03/2019	1.25$	0.01	singapour
12	01/12/2019	1.23$	2.00	madrid

Аналогичным образом мы можем посмотреть на неповторяющиеся значения.

```
1 ( ~ financials.duplicated(subset = ['month'])).sum()
```

```
1 12
```

Этот логический массив можно также использовать как фильтр.

```
1 financials[ ~ financials.duplicated(subset = ['month'], keep = 'last')]
```

	month	profit	MoM	high
0	01/01/2019	1.20$	0.030	Dubai
1	01/02/2019	1.30$	-0.020	Paris
3	01/03/2019	1.25$	0.020	singapour
4	01/04/2019	1.27$	-0.010	moscow
5	01/05/2019	1.13$	-0.015	Paris
6	01/06/2019	1.23$	0.017	Madrid
7	01/07/2019	1.20$	0.035	moscow
8	01/08/2019	1.31$	0.020	london
9	01/09/2019	1.24$	0.010	london
10	01/10/2019	1.18$	0.000	Moscow
11	01/11/2019	1.17$	-0.010	Rome
13	01/12/2019	1.23$	2.000	madrid

Обратите внимание, индекс остался прежним (из него просто выпали наблюдения 2 и 12). Мы исправим эту неточность при удалении дубликатов.

Удаление дубликатов

Метод .drop_duplicates() удаляет дубликаты из датафрейма и, по сути, принимает те же параметры, что и метод **.duplicated()**.

```
1 # параметр ignore_index создает новый индекс
2 financials.drop_duplicates(keep = 'last',
3                            subset = ['month'],
4                            ignore_index = True,
5                            inplace = True)
6 financials
```

	month	profit	MoM	high
0	01/01/2019	1.20$	0.030	Dubai
1	01/02/2019	1.30$	-0.020	Paris

2	01/03/2019	1.25$	0.020	singapour
3	01/04/2019	1.27$	-0.010	moscow
4	01/05/2019	1.13$	-0.015	Paris
5	01/06/2019	1.23$	0.017	Madrid
6	01/07/2019	1.20$	0.035	moscow
7	01/08/2019	1.31$	0.020	london
8	01/09/2019	1.24$	0.010	london
9	01/10/2019	1.18$	0.000	Moscow
10	01/11/2019	1.17$	-0.010	Rome
11	01/12/2019	1.23$	2.000	madrid

Неверные значения

Распространенным типом ошибок в данных являются неверные значения.

> Базовый подход к поиску неверных значений — проверить, что *данные не противоречат своей природе*. Например, цена товара не может быть отрицательной.

В нашем случае мы видим, что в столбце MoM все строки отражают доли процента, а последняя строка — проценты. Из-за этого сильно искажается, например, средний показатель изменения выручки за год.

```
# рассчитаем среднемесячный рост
financials.MoM.mean()
```

```
0.17308333333333334
```

С учетом имеющихся данных вряд ли среднее изменение выручки (в месячном, а не годовом выражении) составило 17,3%. Заменим проценты на доли процента.

```
# заменим 2% на 0.02
financials.iloc[11, 2] = 0.02
```

Вновь рассчитаем средний показатель.

```
financials.MoM.mean()
```

```
0.008083333333333335
```

Новое среднее значение 0,8% выглядит гораздо реалистичнее.

Форматирование значений

Тип str вместо float

Попробуем сложить данные о прибыли.

```
financials.profit.sum()
```

```
'1.201.301.251.271.131.231.201.311.241.181.171.23'
```

Так как столбец profit содержит тип str, произошло объединение (concatenation) строк. Преобразуем данные о прибыли в тип float.

```
# вначале удалим знак доллара с помощью метода .strip()
financials['profit'] = financials['profit'].str.strip('$')

# затем воспользуемся знакомым нам методом .astype()
financials['profit'] = financials['profit'].astype('float')
```

Проверим полученный результат с помощью нового для нас **ключевого слова assert** (по-англ. «утверждать»).

Если условие идущее после assert возвращает True, программа продолжает исполняться. В противном случае Питон выдает AssertionError.

Приведем пример.

```
# напишем простейшую функцию деления одного числа на другое
def division(a, b):
  # если делитель равен нулю, Питон выдаст ошибку (текст ошибки указывать не обязательн
  assert b != 0, 'На ноль делить нельзя'
  return round(a / b, 2)
```

```
# попробуем разделить 5 на 0
division(5, 0)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-43-e021dc5a2ea2> in <module>()
      1 # попробуем разделить 5 на 0
----> 2 division(5, 0)

<ipython-input-42-862994227644> in division(a, b)
      3 def division(a, b):
      4   # если делитель равен нулю, Питон выдаст ошибку (текст ошибки указывать не обязательно)
----> 5   assert b != 0 , 'На ноль делить нельзя'
      6   return round(a / b, 2)

AssertionError: На ноль делить нельзя
```

SEARCH STACK OVERFLOW

Выражение `b != 0` превратилось в False и Питон выдал ошибку. Теперь вернемся к нашему коду.

```
# проверим, превратился ли тип данных во float
assert financials.profit.dtype == float
```

Сообщения об ошибке не появилось, значит выражение верное (True). Теперь снова рассчитаем прибыль за год.

```
financials.profit.sum()
```

```
14.709999999999999
```

Названия городов с заглавной буквы

Остается сделать так, чтобы названия всех городов в столбце high начинались с заглавной буквы. Для этого подойдет **метод .title()**.

```
financials['high'] = financials['high'].str.title()
financials
```

	month	profit	MoM	high
0	01/01/2019	1.20	0.030	Dubai
1	01/02/2019	1.30	-0.020	Paris
2	01/03/2019	1.25	0.020	Singapour
3	01/04/2019	1.27	-0.010	Moscow
4	01/05/2019	1.13	-0.015	Paris
5	01/06/2019	1.23	0.017	Madrid
6	01/07/2019	1.20	0.035	Moscow
7	01/08/2019	1.31	0.020	London
8	01/09/2019	1.24	0.010	London
9	01/10/2019	1.18	0.000	Moscow
10	01/11/2019	1.17	-0.010	Rome
11	01/12/2019	1.23	0.020	Madrid

Дата и время

Как мы уже знаем, с датой и временем гораздо удобнее работать, когда они представляют собой объект datetime. В этом случае нам доступны все возможности Питона по анализу и прогнозированию временных рядов.

Начнем с того, что воспользуемся **функцией pd.to_datetime()**, которой передадим столбец month и формат, которого следует придерживаться при создании объекта datetime.

```
# запишем дату в формате datetime в столбец date1
financials['date1'] = pd.to_datetime(financials['month'], format = '%d/%m/%Y')
financials
```

	month	profit	MoM	high	date1
0	01/01/2019	1.20	0.030	Dubai	2019-01-01
1	01/02/2019	1.30	-0.020	Paris	2019-02-01
2	01/03/2019	1.25	0.020	Singapour	2019-03-01
3	01/04/2019	1.27	-0.010	Moscow	2019-04-01
4	01/05/2019	1.13	-0.015	Paris	2019-05-01
5	01/06/2019	1.23	0.017	Madrid	2019-06-01
6	01/07/2019	1.20	0.035	Moscow	2019-07-01
7	01/08/2019	1.31	0.020	London	2019-08-01
8	01/09/2019	1.24	0.010	London	2019-09-01
9	01/10/2019	1.18	0.000	Moscow	2019-10-01
10	01/11/2019	1.17	-0.010	Rome	2019-11-01

10	01/11/2019	1.17	-0.010	Rome	2019-11-01
11	01/12/2019	1.23	0.020	Madrid	2019-12-01

Мы получили верный результат. Как и должно быть в Pandas, на первом месте в столбце date1 стоит год, затем месяц и наконец день. Теперь давайте попросим Питон самостоятельно определить формат даты.

```
# для этого используем pd.to_datetime() без дополнительных параметров
financials['date2'] = pd.to_datetime(financials['month'])
financials
```

	month	profit	MoM	high	date1	date2
0	01/01/2019	1.20	0.030	Dubai	2019-01-01	2019-01-01
1	01/02/2019	1.30	-0.020	Paris	2019-02-01	2019-01-02
2	01/03/2019	1.25	0.020	Singapour	2019-03-01	2019-01-03
3	01/04/2019	1.27	-0.010	Moscow	2019-04-01	2019-01-04
4	01/05/2019	1.13	-0.015	Paris	2019-05-01	2019-01-05
5	01/06/2019	1.23	0.017	Madrid	2019-06-01	2019-01-06
6	01/07/2019	1.20	0.035	Moscow	2019-07-01	2019-01-07
7	01/08/2019	1.31	0.020	London	2019-08-01	2019-01-08
8	01/09/2019	1.24	0.010	London	2019-09-01	2019-01-09
9	01/10/2019	1.18	0.000	Moscow	2019-10-01	2019-01-10
10	01/11/2019	1.17	-0.010	Rome	2019-11-01	2019-01-11
11	01/12/2019	1.23	0.020	Madrid	2019-12-01	2019-01-12

У нас снова получилось создать объект datetime, однако возникла одна сложность. Функция **pd.to_datetime()** предположила, что в столбце month данные содержатся в американском формате (месяц/день/год), тогда как у нас они записаны в европейском (день/месяц/год). Из-за этого в столбце date2 мы получили первые 12 дней января, а не 12 месяцев 2019 года.

```
# исправить неточность с месяцем можно с помощью параметра dayfirst = True
financials['date3'] = pd.to_datetime(financials['month'],
                                     dayfirst = True)
financials
```

	month	profit	MoM	high	date1	date2	date3
0	01/01/2019	1.20	0.030	Dubai	2019-01-01	2019-01-01	2019-01-01
1	01/02/2019	1.30	-0.020	Paris	2019-02-01	2019-01-02	2019-02-01
2	01/03/2019	1.25	0.020	Singapour	2019-03-01	2019-01-03	2019-03-01
3	01/04/2019	1.27	-0.010	Moscow	2019-04-01	2019-01-04	2019-04-01
4	01/05/2019	1.13	-0.015	Paris	2019-05-01	2019-01-05	2019-05-01
5	01/06/2019	1.23	0.017	Madrid	2019-06-01	2019-01-06	2019-06-01
6	01/07/2019	1.20	0.035	Moscow	2019-07-01	2019-01-07	2019-07-01
7	01/08/2019	1.31	0.020	London	2019-08-01	2019-01-08	2019-08-01
8	01/09/2019	1.24	0.010	London	2019-09-01	2019-01-09	2019-09-01
9	01/10/2019	1.18	0.000	Moscow	2019-10-01	2019-01-10	2019-10-01
10	01/11/2019	1.17	-0.010	Rome	2019-11-01	2019-01-11	2019-11-01

11	01/12/2019	1.23	0.020	Madrid	2019-12-01	2019-01-12	2019-12-01

Теперь мы снова получили верный формат.

```
# убедимся, что столбцы с датами имеют тип данных datetime
financials.dtypes
```

```
month             object
profit           float64
MoM              float64
high              object
date1     datetime64[ns]
date2     datetime64[ns]
date3     datetime64[ns]
dtype: object
```

Удалим избыточные столбцы и сделаем дату индексом.

```
financials.set_index('date3', drop = True, inplace = True) # drop = True удаляет столбе
financials.drop(labels = ['month', 'date1', 'date2'], axis = 1, inplace = True)
financials.index.rename('month', inplace = True)
financials
```

month	profit	MoM	high
2019-01-01	1.20	0.030	Dubai
2019-02-01	1.30	-0.020	Paris
2019-03-01	1.25	0.020	Singapour
2019-04-01	1.27	-0.010	Moscow
2019-05-01	1.13	-0.015	Paris
2019-06-01	1.23	0.017	Madrid
2019-07-01	1.20	0.035	Moscow
2019-08-01	1.31	0.020	London
2019-09-01	1.24	0.010	London
2019-10-01	1.18	0.000	Moscow
2019-11-01	1.17	-0.010	Rome
2019-12-01	1.23	0.020	Madrid

Посмотрим на еще один интересный инструмент. Предположим, что мы ошиблись с годом (вместо 2019 у нас на самом деле данные за 2020 год) или просто хотим создать индекс с датой с нуля. Для таких случаев подойдет **функция pd.data_range()**.

```
# создадим последовательность из 12-ти месяцев,
# передав начальный период (start), общее количество периодов (periods)
# и день начала каждого периода (MS, т.е. month start)
range = pd.date_range(start = '1/1/2020', periods = 12, freq = 'MS')

# сделаем эту последовательность индексом датафрейма
financials.index = range
financials
```

	profit	MoM	high
2020-01-01	1.20	0.030	Dubai

2020-02-01	1.30	-0.020	Paris
2020-03-01	1.25	0.020	Singapour
2020-04-01	1.27	-0.010	Moscow
2020-05-01	1.13	-0.015	Paris
2020-06-01	1.23	0.017	Madrid
2020-07-01	1.20	0.035	Moscow
2020-08-01	1.31	0.020	London
2020-09-01	1.24	0.010	London
2020-10-01	1.18	0.000	Moscow
2020-11-01	1.17	-0.010	Rome
2020-12-01	1.23	0.020	Madrid

Как мы уже знаем, когда индекс имеет тип данных datetime, мы можем делать срезы по датам.

```
# напоминаю, что для datetime конечная дата входит в срез
financials['2020-01': '2020-06']
```

	profit	MoM	high
2020-01-01	1.20	0.030	Dubai
2020-02-01	1.30	-0.020	Paris
2020-03-01	1.25	0.020	Singapour
2020-04-01	1.27	-0.010	Moscow
2020-05-01	1.13	-0.015	Paris
2020-06-01	1.23	0.017	Madrid

Завершим раздел про дату и время построением двух подграфиков. Для этого вначале преобразуем индекс из объекта datetime обратно в строковый формат с помощью **метода .strftime()**.

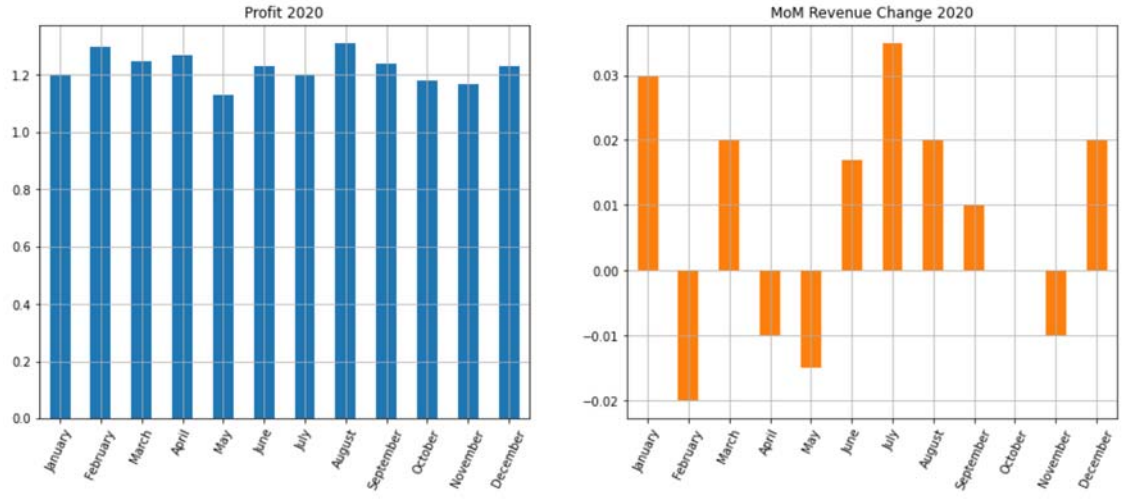
```
# будем выводить только месяцы (%B), так как все показатели у нас за 2020 год
financials.index = financials.index.strftime('%B')
financials
```

	profit	MoM	high
January	1.20	0.030	Dubai
February	1.30	-0.020	Paris
March	1.25	0.020	Singapour
April	1.27	-0.010	Moscow
May	1.13	-0.015	Paris
June	1.23	0.017	Madrid
July	1.20	0.035	Moscow
August	1.31	0.020	London
September	1.24	0.010	London
October	1.18	0.000	Moscow
November	1.17	-0.010	Rome

December	1.23	0.020	Madrid

Теперь используем **метод .plot()** библиотеки Pandas с параметром *subplots* = *True*.

```
# построим графики для размера прибыли и изменения выручки за месяц
financials[['profit', 'MoM']].plot(subplots = True,       # обозначим, что хотим несколь
                                   layout = (1,2),        # зададим сетку
                                   kind = 'bar',          # укажем тип диаграммы
                                   rot = 65,              # повернем деления шкалы оси
                                   grid = True,           # добавим сетку
                                   figsize = (16, 6),     # укажем размер figure
                                   legend = False,        # уберем легенду
                                   title = ['Profit 2020', 'MoM Revenue Change 2020']);
```



Подведем итог

Сегодня мы рассмотрели типичные ошибки в данных и способы их исправления. В частности, мы изучили, как выявить и удалить дубликаты, обнаружить неверные значения и скорректировать неподходящий формат. Кроме того, мы еще раз обратились к объекту datetime и посмотрели на возможности изменения даты и времени.