Конспект по теме «Изучение срезов данных»

Срезы данных и поиск авиабилетов

В ходе работы над задачей, часто нужны не все данные, а лишь их часть, или **срез данных**.

Срез данных можно получить, применив фильтр. Этим фильтром может стать **булев массив**, состоящий из *True* и *False*. Так, значениями *True* отметим, что хотим включить определённую строку в срез данных. Формировать этот фильтр вручную - занятие утомительное, а на больших датасетах - практически невозможное, поэтому в pandas применяются автоматические фильтры. Ниже приведён пример автоматического фильтра.

```
data['column'] == 'value'
```

Этот фильтр в дальнейшем служит как индекс датафрейма:

```
data[data['column'] == 'value']
```

Условием создания фильтра может выступать не только равенство, но и другие операции сравнения: `!=`, `>`, `>=`, `<`, `<=`. Значения в столбцах можно сравнивать и с заданными значениями, и между собой:

```
data['column1'] > data['column2']
```

В условиях допустимы арифметические операции:

```
data['column1'] / 2 > data['column2'] + 0.5
```

Чтобы проверить наличие конкретных значений в столбце, вызовем метод `isin()`:

```
data['column'].isin(('value1', 'value2', 'value3'))
```

Иногда нужно получить срез, соответствующий сразу нескольким условиям — для этого существуют логические операции. Отдельные условия указывают в скобках:

Name	Описание	Синтаксис
ИЛИ	Результат выполнения — *True*, если **хотя бы одно из условий** — *True*	`(df['a'] > 2) | (df['c'] != 'Y')`
И	Результат выполнения логической операции *True*, только если **оба условия** — *True*	`(df['a'] > 2) & (df['c'] != 'Y')`
НЕ	Результат выполнения — *True*, если условие — *False*	`~ ((df['a'] > 2) | (df['c'] != 'Y'))`

Срезы данных методом query()

Другим, более простым методом получения срезов является метод `query()`. Необходимое условие для среза записывается в строке, которую передают как аргумент методу `query()`. А его применяют к датафрейму. В результате получаем нужный срез.

```
data.query('column == value')
```

Условия, указанные в параметре `query()`:

- поддерживают разные операции сравнения: `!=`, `>`, `>=`, `<`, `<=`, - а также математические операции
- допускают вызов методов к столбцам: `column.mean()`
- проверяют, входят ли конкретные значения в список, конструкцией: `column in ("value1", "value2", "value3")`. Если нужно узнать, нет ли в списке определённых значений, пишут так: `a not in ("value1", "value2", "value3")`.
- работают с логическими операторами в привычном виде, где «или» — *or* , «и» — *and*, «не» — *not*. Указывать условия в скобках

необязательно. Без скобок операции выполняются в следующем порядке: сначала *not*, потом *and* и, наконец, *or*.

Также, в методе `query()` можно получать значения внешних переменных:

```
variable = 2
data.query('column > @variable')
```

Работа с датой и временем

При переводе данных из строки в тип *datetime*, как вы знаете, применяется метод `pd.to_datetime()`. Зачастую, pandas пытается самостоятельно угадать формат данных. Однако в таких случаях стоит указывать параметр `yearfirst=True`, который означает, что в переданной строке сначала идёт год. Но всё же рекомендуется прописывать формат вручную, чтобы избежать ошибок.

Для доступа к отдельным компонентам даты и времени используется атрибут `.dt` столбца с датой и временем:

```
data['datetime'].dt.year
data['datetime'].dt.weekday
```

Если нужно прибавить или убавить время, то используется `pd.Timedelta()`, которому можно передать желаемый сдвиг в днях, часах, минутах и т.д., передавая соответствующие параметры. Обратите внимание, что переход через 24 часа автоматически приводит к изменении даты, нам не нужно заниматься этой арифметикой.

```
data['shifted_datetime'] = data['datetime'] + pd.Timedelta(hours=10)
```

Также нам пригодится округлённое время, для этого в pandas есть метод `.dt.round()`. В качестве параметра передаётся шаг округления строкой вида '1H', что означает 1 час (H - hour - час). Не обязательно округлять с шагом в 1 час. Можем округлять и с шагом в несколько часов. Другие популярные единицы округления:

- 'D' day день

- 'H' - hour - час
- 'T' или 'min' - minute - минута
- 'S' - second - секунда

```
data['datetime_round'] = df['datetime'].dt.round('3H')
```

Если нужно округлить в меньшую или большую сторону, вместо метода `round()` используются методы `floor()` и `ceil()`, соответственно

Графики

За построение графиков в *Pandas* отвечает метод `plot()`, который строит графики по значениям столбцов из датафрейма. На оси абсцисс *(x)* расположились индексы, а на оси ординат *(y)* — значения столбцов.

```
data.plot()
```

У метода `plot()` есть параметр `style`, который отвечает за стиль отображения точек:

- `'o'` - вместо непрерывной линии, отображается каждая точка кружком
- `'x'` - вместо непрерывной линии, отображается каждая точка символом 'x'
- `'o-'` - отображается непрерывная линия и точка

Можно изменить оси: напрямую указать, какой столбец отвечает за какую ось

```
data.plot(x='column_x', y='column_y')
```

Границы осей можно скорректировать параметрами `xlim` и `ylim`, как и в случае с ящиком с усами:

```
data.plot(xlim=(x_min, x_max), ylim=(y_min, y_max))
```

Для отображения сетки, указывается параметр `grid`, равный *True*:

```
data.plot(grid=True)
```

Размером графика управляют через параметр `figsize`. Ширину и высоту области построения в дюймах передают параметру в скобках.

```
data.plot(figsize = (x_size, y_size))
```

Группировка с pivot_table()

Когда данных много, точки на графиках сливаются и из визуального представления мало что можно понять. Для решения этой проблемы применяется группировка данных. Пример ниже применяет группировку, благодаря которой график становится гораздо информативнее.

```
(data
     .query('column_id == "value"')
     .pivot_table(index='column1', values='column2')
     .plot(grid=True, figsize=(12, 5))
)
```

С помощью графиков с группировкой можно обнаружить выбросы, которые до этого не были видны. Так, иногда из-за выбросов серьезно завышается среднее арифметическое. Как сделать так, чтобы аномально высокие значения не завышали среднее арифметическое? Решений этой задачи может быть два:

- Убрать выбросы
- Вместо среднего арифметического считать медиану. Медиана устойчива к выбросам, но всё же не безупречна: пики всё ещё могут отображаться

Сохраняем результаты

В ходе работы вы можете обнаружить, что в данных, которые вам предоставили, содержится ошибка. В таком случае, нужно сформулировать проблему, чтобы упростить поиск потенциальной ошибки в алгоритме выгрузки данных. Правильное сообщение об ошибке,

или **баг-репорт**, должно чётко объяснять, в чём именно ошибка и как её найти. Коллеги, отвечающие за выгрузку, *ничего о результатах нашего исследования не знают*. Поэтому нужно чётко формулировать, *где мы видим проблему*.