Ещё больше Pandas

Основы Pandas

- Полезные **методы** и **функции**
  - describe, get_dummies и т.д.
- **Индексирование**
  - loc/iloc
- Операции над **столбцами**
  - arithmetic, apply и т.д.
- Функциональность как в **SQL**
  - merge, join, groupby

Основы Pandas

- Полезные **методы** и **функции**
  - describe, get_dummies и т.д.
- **Индексирование**
  - loc/iloc
- Операции над **столбцами**
  - arithmetic, apply и т.д.
- Функциональность как в **SQL**
  - merge, join, groupby

И даже больше!

Продвинутые функции для обработки данных

Создание сводной таблицы

- Похоже на сводные таблицы (Pivot Tables) в **Excel**
- Перестроить датасет, в котором каждая строка содержит **категорию** с **подкатегорией**, которые связанным с некоторым **значением**
  - Например: **Great Britain, online** имеет прибыль (**profit of) 50**

Country	Type	Profit
GB	online	50
US	online	25
GB	In store	30
US	In store	100

Country	Profit: In Store	Profit: Online
GB	30	50
US	100	25

Сводные таблицы в Pandas

Две функции для создания сводной таблицы из DataFrame

- **.pivot(**index, columns, [values]**)**
  - применима для **численных** и **категориальных** данных
  - не производит агрегаций, поэтому не сработает, если есть дубликаты строк с одинаковыми категориями
- .**pivot_table**(index, columns, [values], [aggfunc])
  - применима только для численных данных
  - может произвести **агрегацию** данных, если имеются несколько строк с одинаковыми категориями (похоже на **groupby**)

Пример: **df.pivot(...)**

- .**pivot** работает для численных и категориальных значений, если среди строк нет таких, у которых значение категории совпадает
  - ***index***: столбец, который будет использоваться как индекс
  - ***columns***: категориальные столбцы, по которым будет произведено разбиение
  - ***values*** [необязательно]: если надо оставить только подмножество столбцов

```
df = pd.DataFrame({'country': ['GB', 'US', 'FR', 'FR', 'GB', 'US'],
                   'sales_type': ['online', 'online', 'online', 'in store', 'in store', 'in store'],
                   'revenue': [100, 200, 300, 400, 300, 600],
                   'owner': ['Paul', 'Laura', 'Eve', 'James', 'Peter', 'Sarah']})


# используйте values=”revenue”, если надо оставить только секцию revenue (оборот)
df.pivot(index="country", columns="sales_type")
        >               revenue           profit
        >               in store online   in store online
        >   country
        >   FR              400    300       150     30
        >   GB              300    100       200     50
        >   US              600    200       250     25
```

Пример: **df.pivot_table(...)**

- .**pivot_table** применяется для численных данных; производит агрегацию по строкам, если есть повторяющиеся строки
  - ***index***: столбец, который будет использоваться как индекс
  - ***columns***: категориальные столбцы, по котором будет произведено разбиение
  - ***values*** [необязательно]: если надо оставить только некоторые столбцы
  - **aggfunc** [необязательно]: функция агрегации

```
df = pd.DataFrame({'country': ['GB', 'GB', 'GB', 'GB', 'US', 'US', 'US', 'US'],
                   'sales_type': ['online', 'online', 'in store', 'in store', 'online', 'online', 'in store',
                                  'in store'],
                   'revenue': [100, 200, 300, 400, 300, 600, 200, 100],
                   'profit': [50, 25, 30, 150, 200, 250, 20, 30]})

# используйте values=”revenue”, если надо оставить только секцию revenue (оборот)
df.pivot_table(index="country", columns="sales_type", aggfunc="sum")
        >                profit         revenue
        >             in store online in store online
        >   country
        >   GB              180     75      700    300
        >   US               50    450      300    900
```

Функция **melt** в Pandas

- Операция, обратная операции **pivot**
- Перестроить датасет, в котором столбцы содержат **категорию** и **значение** для каждого *варианта* ***подкатегории***:

Country	Profit: In Store	Profit: Online
GB	30	50
US	100	25

Country	Type	Profit
GB	online	50
US	online	25
GB	In store	30
US	In store	100

Пример: **df.melt(...)**

- .**melt** принимает следующие параметры:
  - ***id_vars***: столбцы, которые будут использованы в качестве индекса
  - ***value_vars***: столбец/столбцы, которые будут использованы как *варианты подкатегорий*

```
df = pd.DataFrame({'country': ['FR', 'GB', 'US'],
                   'in store': [400, 300, 600],
                   'online': [300, 100, 200]})

df.melt(id_vars="country", value_vars=["in store", "online"])
        >     Country   variable    value
        >         FR    in store    400
        >         GB    in store    300
        >         US    in store    600
        >         FR    online      300
        >         GB    online      100
        >         US    online      200
```

MultiIndex

- иначе: иерархический индекс
- много уровней по столбцам или по строковому индексу
- некоторые команды по умолчанию возвращают датафрейм с **MultiIndex**, например:
  - groupby, pivot, и т.д.

	Profit	
Country	**In Store**	**Online**
GB	30	50
US	100	25

объект DataFrame с мульти-индексом по столбцам (2 уровня)

Пример: **MultiIndex** по столбцам

```
# Прежде чем создать датафрейм, мы создаем мульти-индекс по столбцам
columns = pd.MultiIndex.from_tuples([("profit","online"), ("profit", "in store")])

df = pd.DataFrame([[30,500],[100,25]], columns=columns)
        >         profit
        >   online      in store
        >   30          500
        >   100         25


# Чтобы выбрать отдельные столбцы, можно применить цепное индексирование
df["profit"]["online"]
        >   0     30
        >   1     100
        >   Name: online, dtype: int64

# или использовать метод .loc, передав ему кортеж (tuple)
df.loc[:, ("profit", "online")]
        >   0     30
        >   1     100
        >   Name: online, dtype: int64
```

Пример: строковый **MultiIndex**

```
# Прежде чем создать датафрейм, мы создаем индекс типа MultiIndex
index = pd.MultiIndex.from_tuples([("GB", "London"), ("GB", "Cambridge"), ("US", "New York")])

df = pd.DataFrame({"profit": [100, 200, 300], "revenue": [20, 30, 40]}, index=index)
        >                profit  revenue
        >   GB London     100    20
        >      Cambridge  200    30
        >   US New York   300    40


# Чтобы выбрать подмножество строк, можно использовать метод .loc, передав ему единственное значение
df.loc["GB", :]
        >                profit  revenue
        >   London       100    20
        >   Cambridge    200    30

# или кортеж, если надо выбрать единственную строку
df.loc[("GB", "London"), :]
        >   profit       100
        >   revenue      20
        >   Name: (GB, London), dtype: int64
```

Методы **stack** и **unstack**

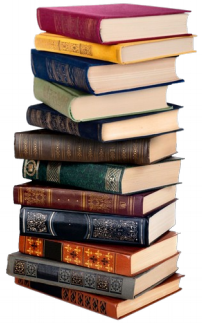
- методы **stack** и **unstack** преобразуют датафрейм с мулти-индексом по столбцам в датафрейм с мульти-индексом по строкам и наоборот
  - метод **stack** преобразует внутренний мулти-индекс **по столбцам** в мульти-индекс **по строкам**
  - метод **unstack** преобразует внутренний мульти-индекс **по строкам** в мульти-индекс **по столбцам**

Давайте посмотрим на примере!

Stack – мультииндекс, от столбцов к строкам

	Profit	
Country	**In Store**	**Online**
GB	30	50
US	100	25

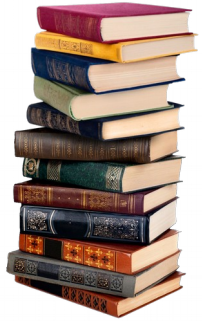
		Profit
GB	In Store	30
	Online	50
US	In Store	100
	Online	25



Unstack – мультииндекс, от строк к столбцам

		Profit
GB	In Store	30
	Online	50
US	In Store	100
	Online	25

	Profit	
Country	**In Store**	**Online**
GB	30	50
US	100	25



Пример: **Stack / Unstack**

```
# Прежде чем создать датафрейм, мы создаем мульти-индекс по столбцам
columns = pd.MultiIndex.from_tuples([("profit","online"), ("profit", "in store")])
df = pd.DataFrame([[30,500],[100,25]], columns=columns)
        >         profit
        >         online      in store
        >   0     30          500
        >   1     100         25

stacked = df.stack()
        >               profit
        >   0 in store  500
        >     online     30
        >   1 in store   25
        >     online    100

stacked.unstack()
        >         profit
        >     in store online
        >   0     500   30
        >   1     25    100
```

Метод **crosstab**

- метод **crosstab** используется для создания таблиц вида “X vs Y”, где значениями будут счетчики (**count**) встречаемости.
    - Например: мы хотим использовать **Country** в качестве индекса (X) и **Industry** в качестве столбца (Y).

Country	Industry
GB	**Banking**
US	Engineering
GB	**Banking**
US	Banking

Country	Banking	Engineering
GB	**2**	0
US	1	1

Пример: **crosstab – со столбцом значений**

- Как альтернатива простому подсчету строк, мы можем явно задать столбец, на основании которого будет вычисляться значение (value), и функцию агрегирования.

```
df = pd.DataFrame({"country": ["GB", "US", "GB", "GB", "US"],
                   "industry": ["Banking", "Engineering", "Healthcare", "Banking", "Banking"],
                   "profit": [100, 200, 50, 200, 100]})
        >     country   industry    profit
        >   0      GB    Banking     100
        >   1      US    Engineering 200
        >   2      GB    Healthcare   50
        >   3      GB    Banking     200
        >   4      US    Banking     100


pd.crosstab(index=df.country, columns=df.industry, values=df.profit, aggfunc="sum")
        >   industry  Banking  Engineering  Healthcare
        >   country
        >   GB          300.0       NaN         50.0
        >   US          100.0       200.0       NaN
```

Пример: **crosstab**

```
df = pd.DataFrame({"country": ["GB", "US", "GB", "GB", "US"],
                   "industry": ["Banking", "Engineering", "Healthcare", "Banking", "Banking"],
                   "profit": [100, 200, 50, 200, 100]})
        >     country   industry    profit
        >   0      GB    Banking     100
        >   1      US     Engineering  200
        >   2      GB    Healthcare   50
        >   3      GB    Banking     200
        >   4      US    Banking     100


pd.crosstab(index=df.country, columns=df.city, values=df.profit, aggfunc="sum")
        >   city  Cambridge  London  New York  Seattle
        >   country
        >   GB          200.0   150.0     NaN   NaN
        >   US          NaN   NaN   200.0 100.0
```

Как насчёт Series?

Метод **factorize**

- Метод **factorize** позволяет преобразовать *категориальные* значения в *целочисленные* метки (labels)
    - сохраняется связь исходных категориальных меток и соответствующих им целочисленных меток

```
countries = pd.Series(["GB", "US", "US", "GB"])
labels, uniques = pd.factorize(countries)

# первый объект содержит названия стран, которые были заменены численными метками
labels
        >   array([0, 1, 1, 0])

# второй это список попарно-различных значений, которые соответствуют численным меткам
uniques
        >   Index(['GB', 'US'], dtype='object')
```

Метод **cut**

- Метод **cut** преобразует **численные** данные в **категориальные**
  - диапазон значений разделяется на N фрагментов (bins – “корзин”)
  - каждое из значений заменяется на корзину, в которую оно попало
- Можно строить гистограммы

Пример: **pd.cut()**

```
values = [1, 1, 2, 3, 4, 5]

# создаем два интервала (“корзины”) и заменяем каждое значение интервалом, в который значение попало
binned = pd.cut(values, bins=2)
binned
        >   [(0.996, 3.0], (0.996, 3.0], (0.996, 3.0], (0.996, 3.0], (3.0, 5.0], (3.0, 5.0]]
        >   Categories (2, interval[float64]): [(0.996, 3.0] < (3.0, 5.0]]

# binned это категориальный ряд (Series), можно получить его значения и его попарно-различные категории
binned.get_values()
        >   array([Interval(0.996, 3.0, closed='right'),
        >         Interval(0.996, 3.0, closed='right'),
        >         Interval(0.996, 3.0, closed='right'),
        >         Interval(0.996, 3.0, closed='right'),
        >         Interval(3.0, 5.0, closed='right'),
        >         Interval(3.0, 5.0, closed='right')], dtype=object)

binned.categories
        >   IntervalIndex([(0.996, 3.0], (3.0, 5.0]]
        >               closed='right',
        >               dtype='interval[float64]')
```

Практика

further_pandas.ipynb