Pandas

Библиотека Pandas это одна из самых популярных библиотек Python для анализа и обработки данных. Она предоставляет удобные структуры данных и множество инструментов для работы с таблицами, статистикой и преобразованием информации. Благодаря pandas можно легко читать, фильтровать, преобразовывать и анализировать большие объемы данных.

В pandas есть две ключевые структуры:

Series - одномерный массив, чем-то похожий на обычный список или столбец в таблице. Каждый элемент **Series** имеет свой индекс.

DataFrame - двумерная таблица, напоминающая электронную таблицу Excel или SQL-таблицу. **DataFrame** состоит из строк и столбцов, каждый столбец может иметь свой тип данных.

Перед началом работы нужно импортировать pandas. Обычно это делают так:

```
import pandas as pd
```

1 Запись и чтение DataFrame из файлов

Файлы с расширением .**csv** часто применяются для хранения табличной информации. Для записи данных из **DataFrame** в такой файл можно использовать метод .to_csv. Вот некоторые из его часто используемых параметров (рис.1):

– sep — символ-разделитель для столбцов в файле. По умолчанию это запятая, но можно указать точку с запятой (;), табуляцию (\t) и другие символы.

– index — логический параметр, определяющий, нужно ли включать индексные значения в сохраняемый файл.

```
In [ ]: b.to_csv("test.csv", sep=";", index=False)
```

Рисунок 1 – Запись в файл

Прочитать массив из файла можно с помощью функции pd.read_csv (рис.2). Здесь также можно указать разделитель столбцов в параметре sep.

```
In [ ]: b = pd.read_csv("test.csv", sep=";")

        b
```

Рисунок 2 – Чтение из файла

Можно также читать данные из Excel, баз данных и других форматов.

2 Создание DataFram

Самый простой способ задать **DataFrame** - с помощью словаря (рис.3), в котором каждый ключ отвечает за столбец, а соответствующее значение - это список из элементов данного столбца. Эти списки должны иметь одинаковую длину.

```
In [ ]: a = {
            "col1": [1, 2, 4, 5, 6, 7, 8],
            "col2": ["a", "c", "e", "g", "z", "x", "y"]
        }

        b = pd.DataFrame(a)

        b
```

Рисунок 3 – Создание DataFrame из словаря

С помощью атрибута .**shape** можно посмотреть форму массива **DataFrame**. Атрибут .**columns** содержит массив из столбцов, а .**index**, как и ранее, содержит массив индексов (рис.4).

```
In [ ]: print("Форма b: {}".format(b.shape))

        print("Столбцы b: {}".format(b.columns))

        print("Индексы b: {}".format(b.index))

        Форма b: (7, 2)
        Столбцы b: Index(['col1', 'col2'], dtype='object')
        Индексы b: RangeIndex(start=0, stop=7, step=1)
```

Рисунок 4 – Атрибуты массивов

После загрузки данных удобно сразу посмотреть на их структуру:

```
print(df.head())       # Показать первые 5 строк
print(df.info())      # Краткая информация о столбцах и типах
данных
print(df.describe())   # Основные статистические показатели по
                      числовым столбцам
```

Общую информацию о массиве можно запросить с помощью метода .**info()**. Нам вернётся информация об индексах и столбцах данного массива, о том, какие типы данных хранятся в каждом из столбцов, а также информация о том, сколько памяти выделено под данный массив (рис.5).

```
In [ ]: b.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 2 columns):
col1    7 non-null int64
col2    7 non-null object
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
```

Рисунок 5 – Методы info

С помощью метода .**describe()** можно получить некоторые статистические характеристики по столбцам с числовыми значениями: среднее значение, среднее квадратическое отклонение, максимум, минимум, квантили и пр (рис.6).

```
In [ ]: b.describe()

Out[40]: count    5.000000
         mean     3.600000
         std      2.408319
         min      1.000000
         25%      2.000000
         50%      3.000000
         75%      5.000000
         max      7.000000
         dtype: float64
```

Рисунок 6 – Основные статистики

3 Выбор данных из массива DataFrame

Для получения данных из массива **DataFrame** используется следующий синтаксис, например, с помощью методов .**head()** и .**tail()** можно получить несколько первых или несколько последних строк таблицы.

Отдельный столбец можно получить, передав его название в квадратные скобки массива:

```
ages = df['Возраст']
```

Если мы хотим указать несколько столбцов, в квадратные скобки нужно подать список из столбцов. Тогда нам вернётся подтаблица исходной таблицы опять в формате **DataFrame**.

Получить данные из строк таблицы **DataFrame** можно получить с помощью атрибута .**loc**. Этот атрибут представляет собой что-то вроде двумерного массива. Конкретное значение (или несколько значений) этого

массива можно получить, указав нужный индекс строки и название колонки.(рис.7)

```
In [ ]: b.loc[2, "col1"]
Out[44]: 4
```

Рисунок 7 – Фильтрация данных

Для фильтрации строк по условию:

```
adults = df[df['Возраст'] >= 18]
```

Добавить новый столбец можно просто присваиванием:

```
df['Возраст_в_месяцах'] = df['Возраст'] * 12
```

Случайный выбор значений из DataFrame. Случайный выбор строк из массива DataFrame производится с помощью метода .sample. Несколько важных параметров:

– frac - какую долю от общего числа строк нужно вернуть (число от 0 до 1)

– n - сколько строк нужно вернуть (число от 0 до числа строк в массиве)

– replace - индикатор того, производится ли выбор *с возвращением*, т.е. с возможным повторением строк в выборке, или *без возвращения* (True или False)

Нельзя использовать параметры frac и n одновременно, нужно выбрать какой-то один (рис.8).

```
In [ ]: b.sample(frac=0.5, replace=True)
```

Out[53]:

	col1	col2
6	8	y
2	4	e
4	6	z
4	6	z

Рисунок 8 – Случайный выбор значений

Если требуется просто перемешать всю выборку, это также можно выполнить с помощью метода .**sample**, передав в него параметр **frac**=1.

4 Работа с данными в Pandas.

Слияние данных. Рассмотрим следующий пример. Допустим, что мы работаем с небольшой структурой факультетов и кафедр. Наша задача - систематизировать работу с подраздеениями.

У нас есть таблица **faculty**, содержащая данные об факультетах: их идентификаторы (**faculty_id**) и названия (**faculty_name**) (рис.9):

```
import pandas as pd
faculty = pd.DataFrame({
    'faculty_id': [1, 2, 3],
    'faculty_name': ['ФКСИС', 'ФИТУ', 'ФИБ'],
})
faculty
```

	faculty_id	dep_title
0	2	ПОИТ
1	2	Кафедра высшей математики
2	3	Кафедра защиты информации
3	4	Кафедра экономики

Рисунок 9 – Исходная таблица

Кроме того, у нас есть таблица departments, содержащая информацию о кафедрах этих факультетов. В этой таблице также есть колонка faculty_id, а также колонка dep_title, содержащая название кафедры (рис.10):

```
departments = pd.DataFrame({
    'faculty_id': [2, 2, 3, 4],
    'dep_title': ['ПОИТ', 'Кафедра высшей математики', 'Кафедра
защиты информации',
                  'Кафедра экономики'],
})

departments
```

	author_id	book_title
0	2	ПОИТ
1	2	Кафедра высшей математики
2	3	Кафедра защиты информации
3	4	Кафедра экономики

Рисунок 10 – Создание нового набора данных

Что делать, если мы, например, захотим сопоставить названия книг именам их авторов? Для этого используется функция **pd.merge**: в эту функцию помещаются те таблицы, которые мы хотим соединить, а также несколько других важных аргументов:

– on - параметр, отвечающий за то, какой столбец мы будем использовать для слияния,

– how - каким образом производить слияние.

Опишем подробнее, какие значения может принимать параметр how:

– "inner" - внутреннее слияние. В этом случае в слиянии участвуют только те строки, которые присутствуют в обоих таблицах,

– "left" - в слиянии участвуют все строки из левой таблицы,

– "right" - то же самое, но для правой таблицы,

– "outer" - внешнее слияние, соединяются все строки как из левой, так и из правой таблицы. Пример:

```
pd.merge(faculty, departments, on='faculty_id', how='inner')
```

Результат привезен на рисунке 11.

	faculty_id	faculty_name	dep_title
0	2	ФКСИС	ПОИТ
1	2	ФКСИС	Кафедра высшей математики
2	3	ФИБ	Кафедра защиты информации

Рисунок 11 - Результат внутреннего объединения

Если мы выбираем "left", "right" или "outer", может случиться так, что строку из одной таблицы будет невозможно соединить со второй. Например, мы видим, что в нашей таблице departments нет кафедр факультета ФИТУ (его id равен 1). В свою очередь, в таблице departments есть кафедра, для которой faculty_id равен 4, хотя, в таблице faculty нет записи с таким faculty_id. Рассмотрим внешнее слияние этих таблиц (рис.12):

```
merged_df = pd.merge(faculty, departments, on='faculty_id',
how='outer')

merged_df
```

	faculty_id	faculty_name	dep_title
0	1	ФИТУ	NaN
1	2	ФКСИС	ПОИТ
2	2	ФКСИС	Кафедра высшей математики
3	3	ФИБ	Кафедра защиты информации
4	4	NaN	Кафедра экономики

Рисунок 12 – Результат внешнего слияния

Как мы видим, в получившейся таблице присутствуют пропущенные значения (NaN).

Работа с пропущенными данными. Чтобы найти и обработать пропущенные значения, используются методы:

```
df.isnull()          # Проверка на пропуски
df.dropna()         # Удаление строк с пропущенными значениями
df.fillna(0)         # Замена пропусков на 0
```

Пропущенные значения в **Series** или **DataFrame** можно получить с помощью метода .**isnull** (рис.14). Наоборот, все имеющиеся непустые значения можно получить с помощью метода .**notnull** (рис.14):

```
merged_df[merged_df["dep_title"].isnull()]
```

	faculty_id	faculty_name	dep_title
0	1	ФИТУ	NaN

Рисунок 13– Фильтр по пустым значениям

```
merged_df[merged_df["faculty_name"].notnull()]
```

	faculty_id	faculty_name	dep_title
0	2	ФКСИС	ПОИТ
1	2	ФКСИС	Кафедра высшей математики
2	3	ФИБ	Кафедра защиты информации

Рисунок 14 – Выбор не пустых значений

Заполнить пропущенные значения (рис.15) каким-то своим значением можно с помощью метода .**fillna()**:

```
merged_df["faculty_name"] =
merged_df["faculty_name"].fillna("unknown")
merged_df
```

	faculty_id	faculty_name	dep_title
0	1	ФИТУ	NaN
1	2	ФКСИС	ПОИТ
2	2	ФКСИС	Кафедра высшей математики
3	3	ФИБ	Кафедра защиты информации
4	4	unknown	Кафедра экономики

Рисунок 15 – Заполнение пустых значений

Добавление столбцов в **DataFrame**. Допустим, на каждой из кафедр работает по 1 человеку. Мы хотели бы создать в таблице merged_df столбец quantity, который бы содержал количество сотрудников на каждой кафедре.

Создание нового столбца в таблице **DataFrame** происходит аналогично созданию нового значения в словаре dict. Достаточно просто объявить значение merged_df["quantity"]. Если подать в это значение какое-нибудь число или строку, то все значения в данном столбце приравняются к этому числу или строке. Также можно подать сюда список, тогда значения из этого списка поступят в соответствующие строки этого столбца. В этом случае длина списка обязана совпадать с числом строк таблицы.

Итак, выберем все строки с непустым значением поля dep_title, и для них запишем в столбец quantity число 1 (рис. 16). Это можно сделать с помощью атрибута .**loc**:

```
merged_df.loc[merged_df["dep_title"].notnull(), "quantity"] = 1
```

	faculty_id	faculty_name	dep_title	quantity
0	1	ФИТУ	NaN	NaN
1	2	ФКСИС	ПОИТ	1.0
2	2	ФКСИС	Кафедра высшей математики	1.0
3	3	ФИБ	Кафедра защиты информации	1.0
4	4	unknown	Кафедра экономики	1.0

Рисунок 16 – Добавление нового столбца

Теперь заполним все пропуски в этом столбце числом 0 (рис.17):

```
merged_df["quantity"].fillna(0, inplace=True)
```

	faculty_id	faculty_name	dep_title	quantity
0	1	ФИТУ	NaN	0.0
1	2	ФКСИС	ПОИТ	1.0
2	2	ФКСИС	Кафедра высшей математики	1.0
3	3	ФИБ	Кафедра защиты информации	1.0
4	4	unknown	Кафедра экономики	1.0

Рисунок 17 – Заполнение пропусков числом 0

Удаление данных. Для удаления данных из **DataFrame** используется метод .**drop**. В этот метод подаётся метка элемента, который необходимо удалить (индекс строки или название столбца), а также ось **axis**. При **axis**=0 удаляется строка, при значении **axis**=1 – столбец. Вначале добавим новый столбец (рис.18), затем удалим его (рис.19)

```
merged_df["wage"] = 2000
```

	faculty_id	faculty_name	dep_title	quantity	wage
0	1	ФИТУ	NaN	0	2000
1	2	ФКСИС	ПОИТ	1	2000
2	2	ФКСИС	Кафедра высшей математики	1	2000
3	3	ФИБ	Кафедра защиты информации	1	2000
4	4	unknown	Кафедра экономики	1	2000

Рисунок 18 – Добавление нового столбца wage

```
merged_df.drop("wage", axis=1, inplace=True)
```

	faculty_id	faculty_name	dep_title	quantity
0	1	ФИТУ	NaN	0
1	2	ФКСИС	ПОИТ	1
2	2	ФКСИС	Кафедра высшей математики	1
3	3	ФИБ	Кафедра защиты информации	1
4	4	unknown	Кафедра экономики	1

Рисунок 19 – Удаление столбца

Теперь удалим строку с индексом 1:

```
merged_df.drop(1, axis=0, inplace=True)
```

Сортировка данных. Параметр **ignore_index**=True подаётся сюда, чтобы индексы соединяемых таблиц не учитывались. В результирующей таблице будут использованы стандартные последовательные индексы, начинающиеся с 0. Отсортируем эту таблицу по столбцу **faculty_id**. Это делается с помощью метода .**sort_values**(рис.20):

```
merged_df.sort_values(by="faculty_id", inplace=True)
```

	faculty_id	faculty_name	dep_title	quantity
0	1	ФИТУ	NaN	0
2	2	ФКСИС	Кафедра высшей математики	1
3	3	ФИБ	Кафедра защиты информации	1
4	4	unknown	Кафедра экономики	1

Рисунок 20 – Сортировка данных

Соединение таблиц. Для соединения таблиц можно пользоваться функцией **pd.concat**. С этой функцией мы уже знакомились, когда изучали библиотеку **numpy**. Здесь эта функция работает аналогичным образом: соединяет таблицы либо вертикально (если указан параметр **axis**=0), либо горизонтально (если **axis**=1).

Соединение происходит с сохранением индексов (рис.21-22), если не указан параметр **ignore_index**=True.

```
df1 = pd.DataFrame({
    'faculty_id': [3, 5],
```

```
    'faculty_name': ['ФИБ', 'ИЭФ'],
    'dep_title': ['Инфокоммуникационных технологий',
'Менеджмента'],
    'quantity': [2, 3],
})

df2 = pd.concat([merged_df, df1], axis=0, ignore_index=True)
```

	faculty_id	faculty_name	dep_title	quantity
0	1	ФИТУ	NaN	0
1	2	ФКСИС	Кафедра высшей математики	1
2	3	ФИБ	Кафедра защиты информации	1
3	4	unknown	Кафедра экономики	1
4	3	ФИБ	Инфокоммуникационных технологий	2
5	5	ИЭФ	Менеджмента	3

Рисунок 21 – Соединение таблиц верикально

```
df3 = pd.DataFrame(
    {'wage': [1800, 1900, 2000, 2100, 1950]},
    index=[1, 2, 3, 5, 6],
)

df4 = pd.concat([df2, df3], axis=1)
```

	faculty_id	faculty_name	dep_title	quantity	wage
0	1.0	ФИТУ	NaN	0.0	NaN
1	2.0	ФКСИС	Кафедра высшей математики	1.0	1800.0
2	3.0	ФИБ	Кафедра защиты информации	1.0	1900.0
3	4.0	unknown	Кафедра экономики	1.0	2000.0
4	3.0	ФИБ	Инфокоммуникационных технологий	2.0	NaN
5	5.0	ИЭФ	Менеджмента	3.0	2100.0

Рисунок 22 – Соединение таблиц горизонтально

Операции над таблицами. Как и ранее с массивами **numpy** и **Series**, с таблицами **DataFrame** можно производить различные математические

операции. Например, значения различных столбцов можно поэлементно перемножать (рис.23), складывать и пр.

```
df4["total"] = df4["quantity"] * df4["wage"]
```

	faculty_id	faculty_name	dep_title	quantity	wage	total
0	1.0	ФИТУ	NaN	0.0	NaN	NaN
1	2.0	ФКСИС	Кафедра высшей математики	1.0	1800.0	1800.0
2	3.0	ФИБ	Кафедра защиты информации	1.0	1900.0	1900.0
3	4.0	unknown	Кафедра экономики	1.0	2000.0	2000.0
4	3.0	ФИБ	Инфокоммуникационных технологий	2.0	NaN	NaN
5	5.0	ИЭФ	Менеджмента	3.0	2100.0	6300.0

Рисунок 23 – Перемножение столбцов

С помощью следующих методов можно посчитать основные статистики по желаемым столбцам:

– df4["price"].max() – максимум;
– df4["price"].min() – минимум;
– df4["price"].mean() – среднее;
– df4["price"].median() – медиана;
– df4["price"].std() - среднее квадратическое значение;
– df4["price"].var() – дисперсия.

С помощью метода .**nlargest** можно вывести несколько наибольших значений. Указывается то, сколько значений нужно вернуть, а также то, по какому именно значению нужно сортировать (рис.24):

```
df4.nlargest(2, "wage")
```

	faculty_id	faculty_name	dep_title	quantity	wage	total
5	5.0	ИЭФ	Менеджмента	3.0	2100.0	6300.0
3	4.0	unknown	Кафедра экономики	1.0	2000.0	2000.0

Рисунок 24 – Выбор 2 самых больших значений

Имеется также аналогичный метод .**nsmallest**.

С помощью метода .**unique** можно получить уникальные значения заданного столбца:

```
df4["faculty_name"].unique()
```

Если нужно получить не уникальные значения, а лишь их количество, можно воспользоваться методом .**nunique**.

С помощью метода .**value_counts** можно получить информацию о том, сколько раз каждое уникальное значение появляется в данном столбце:

```
df4["faculty_name"].value_counts()
```

Группировка данных. Данные в таблице DataFrame можно группировать по повторяющимся значениям выбранного столбца. Группировка позволяет вычислять какие-то *агрегированные* значения, т.е. значения, полученные каким-то образом из групп других значений. Например, если мы захотим сгруппировать нашу таблицу по значениям author_name, то каждая группа будет содержать все строки с одинаковым значением author_name. По таким группам можно затем посчитать какую-нибудь агрегирующую функцию, например, сумму, среднее, минимум и др.

Вот несколько способов это сделать. В первом случае мы просто выбираем конкретный столбец из группировки и применяем к нему какую-то агрегирующую функцию (рис.25):

```
groupby = df4.groupby("faculty_name")
```

```
groupby["wage"].mean()
```

faculty_name	wage
unknown	2000.0
ИЭФ	2100.0
ФИБ	1900.0
ФИТУ	NaN
ФКСИС	1800.0

dtype: float64

Рисунок 25 – Агрегирующие функции

Второй способ - с помощью метода .**agg**. Данный метод является более гибким. Например, он позволяет вычислять одновременно несколько различных агрегирующих функций от разных столбцов (рис.26):

```
groupby.agg({"wage": "max", "total": "count"})
```

faculty_name	wage	total
unknown	2000.0	1
ИЭФ	2100.0	1
ФИБ	1900.0	1
ФИТУ	NaN	0
ФКСИС	1800.0	1

Рисунок 26 - Метод agg

Сводная таблица. В Pandas есть функция DataFrame.pivot_table(), которая позволяет быстро преобразовать DataFrame в сводную таблицу.

Обобщенная схема работы функции pivot_table (рис.27):

```
DataFrame.pivot_table(  data,
                        values=None,
                        index=None,
                        columns=None,
                        aggfunc='mean')
```

Основные входные аргументы метода:

– data – это числовой столбец, к которому мы применяется функция агрегации, например, продажи, скорость, цена и т. Д.

– index – столбцы, которые необходимо преобразовать в строки, можно передать несколько значений в виде списка;

– columns – переменные столбца, которые необходимо представить, как столбец;

– aggfunc – тип операции, выполняемой с данными, т. е. сумма, среднее значение, количество и т. д.

Пример: необходимо вычислить сколько книг каких авторов имеется в мягком и твердом переплетах:

```
pd.pivot_table(df4, index=["faculty_name"], columns =['wage'],
values=["quantity"], aggfunc=np.sum, fill_value=0)
```

	quantity			
wage	1800.0	1900.0	2000.0	2100.0
faculty_name				
unknown	0.0	0.0	1.0	0.0
ИЭФ	0.0	0.0	0.0	3.0
ФИБ	0.0	1.0	0.0	0.0
ФКСИС	1.0	0.0	0.0	0.0

Рисунок 27 – Сводная таблбица