Язык Python. Часть 2

Лекция 7

Пример создания

```python
import pandas as pd

df = pd.DataFrame(
        {
            "Name": [
                "Braund, Mr. Owen Harris",
                "Allen, Mr. William Henry",
                "Bonnell, Miss. Elizabeth",
            ],
            "Age": [22, 35, 58],
            "Sex": ["male", "male", "female"],
        }
)
print(df)
```

DataFrame

- Поймем, что происходит
  - Вызываем конструктор DataFrame
  - Передаем начальные данные в виде словаря
  - Ключ - строка, имя колонки
  - Значение - список, значения соответствующей колонки
  - Если напечатаем полученное значение, увидим красивую таблицу

DataFrame

- Поделаем еще что нибудь
  - У нас в момент инициализации все списки одной длины
  - Попробуем один удлиннить
  - Или укоротить
  - И создать DataFrame
  - Получим исключение
  - Потому что нарушили "табличность"

Продолжаем пример

```
import pandas as pd

df = pd.DataFrame(
        {
            "Name": [
                "Braund, Mr. Owen Harris",
                "Allen, Mr. William Henry",
                "Bonnell, Miss. Elizabeth",
            ],
            "Age": [22, 35, 58],
            "Sex": ["male", "male", "female"],
        }
)

print(df["Name"])
```

Series

- Извлекаем колонку из таблицы
- Это объект другого типа, Series
- Многие общие методы с DataFrame
- Но разные детали поведения
- Под колонкой с данными видим еще пару аттрибутов
- Имя и тип

Тип данных колонки

- dtype - тип данных
- Это не питоновский тип
- Этой свой, пандасовский тип
- В наших конкретных данных
  - В одной колонке - только числа
  - В двух других - только строки

Тип данных колонки

- Мы эти типы нигде явно не указывали
- Но при чтении данных pandas заметил, что в одной из колонок - только числа
- Более того, он заметил, что числа не совсем гигантские
- Поэтому его тип - int64
- Он ограничен в размере - в отличие от питоновского целого
- Но позволяет быстрее обрабатывать большие таблицы

Тип данных колонки

- Какие бывают типы данных Pandas
  - int64 - целое
  - float64 - вещественное (как в Python)
  - bool - логическое
  - datetime64 - дата и время
  - timedelta - разница между datatime64, временной интервал
  - category - ограниченный список текстовых значений (уровень образования)
  - object - прочее, включая строки

Series

- Вернемся к примеру
- Возьмем в логике инициализации одно из значений возраста в кавычки
- Тогда все значения перестанут быть числами
- И pandas определит тип столбца как object
- И это повлияет на операции над столбцом

Продолжаем дальше пример

```
import pandas as pd

df = pd.DataFrame(
        {
            "Name": [
                "Braund, Mr. Owen Harris",
                "Allen, Mr. William Henry",
                "Bonnell, Miss. Elizabeth",
            ],
            "Age": [22, 35, 58],
            "Sex": ["male", "male", "female"],
        }
)

print(df["Age"].max())
```

Поймем, что происходит

- Первые два вызова - определение характеристик числового столбца
- Но максимум определился бы и для строкового столбца
- Получили бы "последний по алфавиту"
- А mean - смысла не имеет
- В частности - если бы "испортили" числовой столбец строкой

Что делает describe

Что делает describe

- Если строковый тип, то показывает
  - count - количество значений
  - unique - количество уникальных
  - top - самое частотное
  - freq - встречаемость самого частотного

Что делает describe

Вызванный над DataFrame

- Определяем числовые столбцы
- По каждому из них
- Показывает то, что показал бы при вызове над ними

Что делает describe

- Это все поведение по умолчанию
- Все детали невозможно рассказать
- Подробности в документации
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.descri
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.describe.ht

Добавление элементов

- Проще всего добавлять в конец
- Есть метод append
- В него можно передать словарь
- Как в инициализации
- Только значениями будут не списки, а одиночные значения
- А можно передать "безымянную" строчку
- В виде объекта класса Series

Добавление элементов

- Рекомендуемым способом пополнения таблицы
- Является функция concat
- В общем смысле она склеивает однородные таблицы
- Которые передаются параметром-списком
- Но можно использовать DataFrame-конструктор
- Чтобы создать временный DataFrame
- Для добавления

Продолжаем дальше пример

```
import pandas as pd

df = pd.DataFrame(
        {
            "Name": [
                "Braund, Mr. Owen Harris",
                "Allen, Mr. William Henry",
                "Bonnell, Miss. Elizabeth",
            ],
            "Age": [22, 35, 58],
            "Sex": ["male", "male", "female"],
        }
)

df2 = pd.concat([df, pd.DataFrame({'Name':['Jane'], 'Age':[
```

Чтение из csv-файла

- Как и везде - много параметров
- Рассматриваем поведение по умолчанию
- И самые ключевые параметры
- Можно почитать из csv-файла: pd.read_csv("data/titanic.csv")
- В csv-файлах первая строка бывает заголовочной
- В ней хранятся имена полей
- В терминах DataFrame они станут именами колонок

Чтение из csv-файла

- Типы данных колонок определяется анализом читаемых данных
- Чтобы понять, что наопределялось, есть атрибут types
- Есть разные модификации csv
- Например, excel
- Для него есть своя функция: pd.read_excel("titanic.xlsx", sheet_name="passengers")

Запись

- Мы можем насобирать данных
- И потом сохранить в csv-файл
- Тоже разные форматы
- И методы с общим префиксом to_
- titanic.to_csv("titanic.csv")

Вырезки из таблицы

- Метод head возвращает первые строчки таблицы или столбца
- Можно выбрать несколько колонок
- И работать с ними как с подтаблицей
- Нужно в квадратных скобках указать список имен столбцов
- Вот так: titanic[["Age", "Sex"]]

Выборка по условию

- Можно получить таблицу со строками, для которых выполнено условие
- above_35 = titanic[titanic["Age"] > 35]
- Таблица из строчек, в который поле Age больше, чем 35
- Пассажиры 2 и 3 классов: titanic[titanic["Pclass"].isin([2, 3])]
- Можно определить размеры таблицы: above_35.shape

Выборка по индексу

- Можно применить классическую вырезку
- К таблице или столбцу
- И получить подмножество строк/элементов
- Определяемое индексами границ

Двумерные выборки

- Пока что мы делали одномерные выборки
- А иногда хочется выбрать какие-то столбцы и какие-то строки
- Для этого есть вспомогательная структура
- Ее можно получить через атрибут loc или iloc

loc/iloc

- После loc/iloc идут квадратные скобки
- Внутри них - запятая
- Слева вырезка по строкам
- Справа - вырезка по столбцам

loc

- loc подразумевает логическую выборку по столбцам
- И символическую по строкам, то есть имена колонок
- Логическая выборка - это обычно логическое выражение с участием столбцов
- То есть формулируем критерий на основе значений в столбцах

Пример на loc

```
1 adult_names = df.loc[df["Age"] < 35, "Name"]
2 adult_names.head()
```

Проанализируем пример

- По первому измерению берем столбец 'Age'
- И сравниваем с 35
- Формируем логический Series
- То есть отбираем записи, в которых возраст превышает 35
- И берем из них только поле 'Name'
- Это по второму измерению

iloc

- Выборка по номерам строк
- И по номерам столбцов
- Например так: titanic.iloc[9:25, 2:5]
- Строки в диапазоне 9-25 и столбцы в диапазоне 2-5
- Как всегда - левая граница входит, правая не входит