Software Engineering from Innopolis

Vladimir Maximov

30 августа 2023 г.

1 Итеритуемые объекты в Python

1.1 Цель

Изучить коллекции в Python и работу с ними.

1.2 Задачи

- Ознакомиться со списками в Python.
- Рассмотреть популярные операции, используемые над списками slice, filter, map, reduce.
- Ознакомиться с кортежем в Python.
- Разобрать различные подходы создания коллекций.
- Ознакомиться с множествами в Python.
- Понять, чем руководствоваться при выборе коллекции для хранения объектов.

1.3 Список

Список - упорядоченная и изменяемая коллекция объектов:

```
1   my_list = [1, 2, 3, None, [0], "word"]
```

Слайс (срез) списка - создание нового списка, используя элементы уже существующего:

```
1   my_list[2:4:1]
```

Берем со 2 элемента включительно по 4 элемент невключительно с шагом 1.

Краткая форма создания списка называется List comprehention (генератор списка):

```
1   [x for x in range(10)]
```

1.4 Кортеж

Кортеж - упорядоченная и неизменяемая коллекция:

```
1   (1, 2, 3, 4, 5)
```

1.5 Множество

Множество - неупоярдоченная коллекция значений, в которой не допускаются повторения и не может содержать не хешируемых объектов:

```
1   {1, 2, 3, "some string"}
```

1.6 Словари

Словарь - изменяемая коллекция объектов, которые обладают ключевыми словами. В Python словарь можно описать указанием ключей и значений элементов или генерирующим выражением:

```
1   {"key1" : "value1",
2   "key2" : "value2",
3   "key3" : "value3"}
```

1.7 Использование памяти

Для хранения коллекций выделяется чуть больше памяти, чем фактически необходимо. Делается это для того, чтобы интерпретатор Python при добавлении элементов выделял память реже.

В Python в коллекции можно хранить разные объекты, каждый из которых может быть непредвиданного размера. Поэтому в Python в списке хранятся указатели на нужные объекты.

1.8 Itertools

itertools - модуль, который предоставляет различные функции для работы с итерируемыми объектами. Его можно использовать для упрощения записи операций над итерируемыми объектами. Примеры методов:

itertools.combinations - метод для поиска подмножеств итерируемого объекта, возвращает генератор:

```
1   import itertools
2   itertools.combinations("ABCD", 2)
3   list(itertools.combinations("ABCD", 2))
4   # Вывод: [(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)]
```

itertools.compress - метод, который выбирает из исходного итерируемого объекта элементы, согласно селектору (маске):

```
1   import itertools
2   itertools.compress([1,2,3,4], [1,0,1,0])
3   list(itertools.compress([1,2,3,4], [1,0,1,0]))
4   # Вывод: [1,3]
```

2 Функции над итерируемыми объектами

2.1 Цель

Ознакомиться с тем, какие операции могут быть сделаны на итерируемых объектах и что можно представить в качестве итерируемого объекта.

2.2 Задачи

- Ознакомиться как еще можно работать со списками
- На примерах понять, какие объекты можно представлять в качестве итерируемого объекта.

2.3 Itertools (продолжение)

Если стандартного набора библиотеки itertools не хватает, то можно использовать сторонний пакет - more_itertools, он расширяет возможности работы с итерируемыми объектами. Примеры функций:

Функция chunked:

```
from more_itertools import chunked
iterable = [0, 1, 2, 3, 4, 5, 6, 7, 8]
list(chunked(iterable, 3))
# Вывод: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Функция flatten:

```
from more_itertools import flatten
iterable = [(0, 1), (2, 3)]
list(flatten(iterable))
# Вывод: [0, 1, 2, 3]
```

Функция split_at:

```
from more_itertools import split_at
list(split_at('abcdcba', lambda x: x == 'b'))
# Вывод: [['a'], ['c', 'd', 'c'], ['a']]
```

Функция transpose:

```
from more_itertools import transpose
list(transpose([(1, 2, 3), (11, 22, 33)]))
# Вывод: [['a'], ['c', 'd', 'c'], ['a']]
```

Функция windowed:

```
from more_itertools import windowed
all_windows = windowed([1, 2, 3, 4, 5], 3)
list(all_windows)
# Вывод: [(1, 2, 3), (2, 3, 4), (3, 4, 5)]
```

2.4 Filter, Map, Reduce

Иногда для целесообразного использования памяти лучше написать часть программы в функциональном стиле. Особенно это применимо когда нужно пройтись по большому количеству элементов. Для этого в python существуют встроенные функции. Их использование будет рассмотрено на простых примерах.

2.4.1 Filter

Filter - функция, которая позволяет выбрать из любого итерируемого объекта элементы удовлетворяющие условию. В качестве аргумента принимает функцию или лямбда-выражение и список, из которого отфильтрует значения:

```
items = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
def my_filter_expr(item):
    return item > 0
positive_items = tuple(filter(my_filter_expr, items))
print(positive_items)
# Вывод: (1, 2, 3, 4, 5)
```

Важно понимать что получившийс объект не хранит в себе все элементы получившегося списка. Он хранит информацию о том как можно получить каждый из последующих элементов списка. Важно понимать как это работает чтобы в дальнейшем не попадать на ошибки. Самый близкий родственный объект к получившемуся фильтру - итератор. Итератор это специальный объект в python который выдает по одном элементу и по нему можно пройтись только один раз.

2.4.2 Map

Map - функция, которая позволяет применить функцию ко всему списку значений. В качестве аргумента принимает функцию или лямбда-выражение и список, к элементам которого будет применена функция:

```
list(map(lambda a: a[0]**a[1], [(0, 2), (1, 2), (2, 2)]))
# Вывод: [0, 1, 4]
```

2.4.3 Reduce

Reduce - функция, применяющая другую функцию к последовательным парам значений в списке, аналог функции Fold в Wolfram Mathematica:

```
from functools import reduce
many_items = [1,2,3,4]
product = reduce(lambda a, b: a * b, many_items)
# Вывод: 24
```

В начале в качестве переменной a функция берет значение 1, в качестве переменной b - 2, на втором шаге переменная a - результат функции на предыдущей итерации, b - 3 и т.д. У функции есть третий необязательный аргумент - начальное значение.

2.5 Метод скользящего окна

Метод скользящего окна представляет собой процесс, при котором окно фиксированного размера последовательно перемещается по набору данных. Значения внутри окна анализируются и обрабатываются для получения новых результатов, например, вычисления среднего или медианного значения.

2.6 Операции из линейной алгебры

Линейная алгебра это раздел математики, основными конструкциями в которой являются списочный типы данных. Матрицы, векторы, тензоры - это все понятия из линейной алгебры. Самые базовые математические операции в линейной алгебре - это операции на этих структурах определенных в линейном пространстве - транспонирование, перемножение матриц и векторов, нахождение определителей, решение системы линейныхуравнений. Пример перемнжения матриц в помощью more_itertools:

```
1   from more_itertools import matmul
2   matrix1 = [
3   [1,2,3],
4   [4,5,6],
5   ]
6   matrix2 = [
7   [7, 8],
8   [9, 10],
9   [11, 12],
10  ]
11  list(matmul(matrix1, matrix2))
12  # Вывод: [[58, 64], [139, 154]]
```

2.7 Линейная алгебра с numpy

numpy - библиотека, которая заточена под выполнение операций на матрицах. Основные фукнции можно посмотреть по ссылке. Пример перемножения матриц с помощью numpy:

```
1   import numpy as np
2   matrix1 = np.array([
3   [1,2,3],
4   [4,5,6],
5   ])
6   matrix2 = np.array([
7   [7, 8],
8   [9, 10],
9   [11, 12],
10  ])
11  matrix1 @ matrix2
12  # Вывод: array([[58, 64], [139, 154]])
```

3 Базы данных и SQL, работа с бд из Python

3.1 Цель

Рассмотреть базы данных как один из возможных источников данных для работы.

3.2 Задачи

- Разобраться как работать с табличными данными
- Понять, как можно обращаться с табличными данными из python
- Выяснить, каким инстументарием можно воспользоваться для работы с табличными данными.

3.3 Табличное представление данных, именованные данные, работа с ними

Когда мы работае с данными, часто так бывает что они представлены в виде таблиц. То есть представлят собой последовательные наборы строк с данными, в которой каждый отдельный элемент может быть ассоциирован с наименованием и/или типом данных.

3.4 Реляционные базы данных

Табличное представление свойственно для реляционных БД. База данных (БД) - это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера. В таких таблицах все поля не только уникально проименованы, но и имеют строгий тип данных, а также могут иметь ограничения по уникальности и связи с другими таблицами. Пройденный курс по SQL с заданиями в СУБД MySQL по ссылке.

3.5 Работа из python: sqlite3, psycopg2, pandas

Для работы с данными из реляционных БД в языке программирования python существуют библиотеки для низкоуровневого взаимодействия с ними. Для связи с БД sqlite, которая не запускается на отдельном сервере а хранится файлом в локальной файловой системе, существует модуль sqlite3, который не нужно устанавливать отдельно, он встроен в стандартную библиотеку python. В его стандартный функционал входит соединение с базой, исполнение sql выражений и работа с полученным от резаультата выражением как с объектом знакомым языку программирования (итерируемый объект).

```
1     import sqlite3
2     from contextlib import closing
3
4     sqlite_filename = "datasets/movies.sqlite"
5     con = sqlite3.connect(sqlite_filename)
6     def dict_factory(cursor, row):
7         d = {}
8         for idx, col in enumerate(cursor.description):
9             d[col[0]] = row[idx]
10        return d
11
12    con.row_factory = dict_factory
13
14    with closing(con.cursor()) as cur:
15        for row in cur.execute("SELECT * FROM movies JOIN\
16        \directors ON movies.director_id = directors.id"):
17            print(row)
```

В данном случае мы с использованием sqlite3 получаем все фильмы из таблицы *movies*, режиссеры которых присутствуют в таблице *directors*.

Самым классическим инструментом для работы с датасетами является pandas. Помимо того чтобы работать с данныими из электронных таблиц, он может забрать данные из sql-query, если в качестве параметра ему передать соединение с базой. Данные, полученные с помощью sql select можно сразу же загрузить в датафрейм. у pandas есть такой инструмент.

```
1    import pandas as pd
2    import pandas.io.sql as psql
3
4    connection = psycopg2.connect("dbname=test user=student1\
5    \password=student1 host=212.109.218.158 port=5432")
6    dataframe = psql.read_sql('SELECT * FROM files JOIN\
7    \person ON files.owner = person.id;', connection)
```

3.6 Краткое объяснение нереляционных баз данных на примере MongoDB

Не все БД занимаются хранением связанных прямоугольных таблиц. Это происходит потому что тип хранилища состоящий из строго типизорованных связанных таблиц подзодит не для всех типов задач. Один из примеров такой базы - MongoDB.

Mongo - популярная хостируемая на сервере документная (не реляционная, NoSQL) БД. Структура данных в этой и подобных базах отличается от всего пройденного до этого. Напоминает словари в python или же json файл. Причинами выбора именно нереляционной базы для вашего проекта могут быть следующие:

- В проекте (или его части) не нужны взаимоотношения между объектами, вы собираетесь хранить много данных в коллекции, каждый объект в которой самостоятелен сам по себе. Часто это фронтенд-ориентированные проекты или части проектов. (отсутствие связей между объектами и быстрый поиск по коллекции позволяют быстро выдавать данные для отображения на сайте или мобильном устройстве)
- Заранее не известна модель хранимых данных
- Отличная масштабируемость

4 numpy / pandas

4.1 Цель

Ознакомиться с функционалом основного инструмента анализа данных.

4.2 numpy

Самый фундаментальный пакет для работы с данными. Часто он выступает основой для других пакетов работы с данными. В частности это происходит потому что разработчики библиотеки на очень глубинном уровне переработали взаимедействие python с коллекциями (особенно многомерными). В результате чего работа с numpy.ndarray происходит значительны быстрее по скорости чем со стандартным list.

Основной пакет, который будет использоваться - numpy.linalg. Все алгоритмы, связанные с линейной алгеброй есть в данном пакете.

4.3 pandas

Обычно при работе с данными мы наблюдаем что каждый кусочек данных помимо числовых значений может относиться к разным категориям, быть логически отделимым от другого кусочка таких же данных по набору признаков. Тут для анализа данных появляется такой инструмент как pandas. Он реализует работу с кусками данных. Тип объекта с данными который загружен с помощью pandas будет называтья DataFrame, а одна отдельная запись из датафрейма будет называться series.

Внутри себя pandas использует алгебру реализованную в NumPy. pandas имеет под собой зависимость в numpy.

Series (серия)- это одномерный, именованный (помеченный) массив, который может хранить данные любого типа, включая целые числа, числа с плавающей точкой, строки и объекты Python. Базовая структура серии состоит из двух компонентов: данных и индекса. В серии данные представляют собой одномерный массив NumPy, содержащий фактические значения данных.

```
data_frame_covid.T.iloc[:,0]
# Вывод:
# country         Afghanistan
# continent              Asia
# date             2020-04-12
# day                      12
# month                     4
# year                   2020
# cases                    34
# deaths                    3
# country_code            AFG
# population       37172386.0
# Name: 0, dtype: object

type(data_frame_covid.T.iloc[:,0])
# Вывод: pandas.core.series.Series
```

Индекс - это структура, похожая на массив, которая помечает каждый элемент массива данных. По умолчанию индекс представляет собой последовательность целых чисел, начинающуюся с нуля, но также можно предоставить пользовательские метки для индекса. Это облегчает обращение и манипулирование элементами данных на основе их меток. Чтобы создать серию, можно использовать конструктор pd.Series() и передать список, словарь или массив NumPy в качестве данных. Также можно передать необязательный параметр индекса, чтобы указать пользовательские метки для элементов данных.

DataFrame (ДатаФрейм, Таблица данных) - это двумерная, изменяемая в размере и гетерогенная структура табличных данных с маркированными осями (строками и столбцами). Его можно рассматривать как коллекцию объектов Series, где каждый столбец представляет собой объект Series. DataFrames предназначены для обработки широкого спектра типов данных, что делает их универсальным и мощным инструментом для обработки и анализа данных. Базовая структура DataFrame состоит из трех компонентов: данных, индекса и столбцов. В DataFrame данные хранятся в виде коллекции из одного или нескольких объектов Series, каждый столбец которых представляет собой отдельную Series. Данные в каждой Series могут быть различных типов, таких как целые числа, числа с плавающей точкой или строки.

Столбцы являются структурой, подобной массиву, которая помещает метку на каждый столбец в DataFrame.

```
1   data_frame_covid.columns
2   # Вывод: Index(['country', 'continent', 'date', 'day',
3   #       'month', 'year', 'cases',
3   #       'deaths', 'country_code', 'population'],
4   #       dtype='object')
```

Как Series так и DataFrame предоставляют методы для анализа данных, такие как:

- Выборка данных: можно указать особые строки и колонки с которыми вы будете работать, используя индексы, слайсы, условия
- Очистка данных: можно заполнять пропущенные данные, удалять записи в которых не хватает данных.
- Преобразование данных: можно применять функции по всей длинне фрейма или по его отдельным частям
- Аггрегация данных: можно вычислять такие оценочные величины как сумма, среднее, количество.А также группировать данные по своим критериям при помощи groupby.
- Сортировка: можно упорядочить данные основываясь на значениях в колонках, а также по своим критериям
- Функциональность временных рядов: вы можете работать с данными временных рядов, пересчитывать данные с разной частотой и выполнять вычисления с использованием скользящего окна в DataFrame или Series.

В качестве основных типов данных для работы, pandas использует типы данных из NumPy по пораву наследования, ознакомиться с которыми можно по ссылке.

Во многом успех работы с данными зависит в скорости доступа к данным и возможности быстро отфильтровать ненужное. Для ускорения работы нам могут помочь индексы. При создании датафрейма у него имеется стандартный индекс для быстрого доступа через него до строк данных. Выбирать такие строки можно с помощью функций *.loc*() и *.iloc*() (i==integer). Для датафрейма можно установить новый индекс (при этом либо заменив старый либо создав новый датафрейм из существующего с новым индексом).

Комбинирование фреймов:

```
1   data_frame_covid[1:3]
2       .append(data_frame_covid[-3:-1], ignore_index=True)
```

	country	continent	date	day	month	year	cases	deaths	country_code	population
0	Albania	Europe	2020-04-12	12	4	2020	17	0	ALB	2866376.0
1	Algeria	Africa	2020-04-12	12	4	2020	64	19	DZA	42228429.0
2	Yemen	Asia	2020-04-12	12	4	2020	0	0	YEM	28498687.0
3	Zambia	Africa	2020-04-12	12	4	2020	0	0	ZMB	17351822.0

Рис. 1: результат комбинирования фреймов

Для того чтобы выбрать несколько условий для фильтрации датафрейма можно воспользоваться методом query(), он позволяет в одну строку записать все условия фильтрации которые вам могут быть нужны при выборе данных. при этом в результате вы получите новый фрейм с данными.

```
1   data_frame_covid.query("cases > 5")
```

	country	continent	date	day	month	year	cases	deaths	country_code	population
0	Afghanistan	Asia	2020-04-12	12	4	2020	34	3	AFG	37172386.0
1	Albania	Europe	2020-04-12	12	4	2020	17	0	ALB	2866376.0
2	Algeria	Africa	2020-04-12	12	4	2020	64	19	DZA	42228429.0
3	Andorra	Europe	2020-04-12	12	4	2020	21	2	AND	77006.0
7	Argentina	America	2020-04-12	12	4	2020	162	7	ARG	44494502.0
...	...	...	...	...	...	...	...	...	...	...
194	United_Arab_Emirates	Asia	2020-04-12	12	4	2020	376	4	ARE	9630959.0
195	United_Kingdom	Europe	2020-04-12	12	4	2020	8719	839	GBR	66488991.0
198	United_States_of_America	America	2020-04-12	12	4	2020	28391	1831	USA	327167434.0
199	Uruguay	America	2020-04-12	12	4	2020	7	0	URY	3449299.0
200	Uzbekistan	Asia	2020-04-12	12	4	2020	172	1	UZB	32955400.0

Рис. 2: фильтрация фрейма

5 Разведывательный анализ данных

5.1 Цель

Научиться понимать, описывать набор данных с помошью инструментов на языке программирования python.

5.2 Смысл разведывательного анализа данных

Когда мы работаем с небольшими данными, мы хотя бы в общих чертах понимаем основные характеристики датафрейма: сколько в нем данных, насколько одни разнородны, что их объединяет, могут ли быть колонки связаны друг с другом.

```
1    data = {
2        'Name': ['Вася', 'Коля', 'Маша', 'Сережа'],
3        'Age': [20, 21, 19, 18],
4        'Height': [183, 176, 163, 180],
5    }
6
7    simple_data_frame = pd.DataFrame(data)
8    simple_data_frame
```

	Name	Age	Height
0	Вася	20	183
1	Коля	21	176
2	Маша	19	163
3	Сережа	18	180

Рис. 3: небольшой датафрейм

Для начала работы с большими фреймами есть разведывательный анализ данных (Exploratory Data Analysis, EDA). Это набор техник работы с данными, позволяющий, если кратко говорить, ответить на вопрос "что передо мной такое, с чем я работаю?". Вопрос звучит просто, однако он требует умения работы со статистикой, понимания соответствующих терминов. Также вопрос усложняется тем что на него существует одновременно большое множество правильных ответов и чем больше вы исследуете датасет тем больше верных утверждений вы на него обнаружите.

Для описания датафрейма в целом есть метод $info()$:

```
1    data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96225 entries, 0 to 96224
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   name                     96225 non-null  object
 1   has_test                 96225 non-null  bool
 2   response_letter_required 96225 non-null  bool
 3   salary_from              56084 non-null  float64
 4   salary_to                39687 non-null  float64
 5   salary_currency          61763 non-null  object
 6   salary_gross             61493 non-null  object
 7   published_at             96225 non-null  object
 8   created_at               96225 non-null  object
 9   parsed_at                96225 non-null  object
 10  url                      96225 non-null  object
 11  employer_name            96225 non-null  object
 12  description              96223 non-null  object
 13  alternate_url            96225 non-null  object
 14  area_id                  96225 non-null  int64
 15  area_name                96225 non-null  object
dtypes: bool(2), float64(2), int64(1), object(11)
memory usage: 10.5+ MB
```

Рис. 4: результат метода info()

Самое первое что стоит сделать до математических оценок чего бы то ни было - это проверить, наколько пуст наш набор данных. Есть вероятность что во множестве критически важных колонок нашего датасета вписаны пустые значения и тогда ценность для нас такого набора данных резко снижается.

```
1    data_frame.isna().sum()
```

```
name                          0
has_test                      0
response_letter_required      0
salary_from                40141
salary_to                  56538
salary_currency            34462
salary_gross               34732
published_at                  0
created_at                    0
parsed_at                     0
url                           0
employer_name                 0
description                   2
alternate_url                 0
area_id                       0
area_name                     0
dtype: int64
```

Рис. 5: наполненность датасета

5.3 Оценка центрального положения

Базовый шаг любого развед-анализа (одно из первых действий которое вы выполняете) это определение среднего положения. То есть ответ на вопрос "как выглядит самое типичное значение в моих данных?". То есть оценка того, где расположено большинство данных (центральная тенденция). При этом не всегда взятие "среднего арифметического"это самый правильный подход к такой оценке. А вдруг мой набор данных содержит странные выбросы самого наибольшего и наименьшего значния? А откуда я вообще знаю что такое среднее чувствительно к выбросам? Для того чтобы получить "самое среднее"среднее есть различные методы.

```
1    data_frame.describe()[critical_columns_numeric]
```

	salary_from	salary_to
count	3.182200e+04	3.182200e+04
mean	6.394237e+04	1.129903e+05
std	5.447614e+04	1.684308e+06
min	2.000000e+00	2.500000e+01
25%	3.350000e+04	4.500000e+04
50%	5.000000e+04	7.000000e+04
75%	8.000000e+04	1.300000e+05
max	3.400000e+06	3.000000e+08

Рис. 6: наполненность датасета

Оценки:

- mean - среднее - среднее арифметическое - самая примитивная оценка центрального положения.
- Средневзвешенное - вычисляется как сумма значений умноженная на веса, деленная на сумму весов. Такая оценка среднего положения предполагает наличие колонки с весами, то есть некоего коэффициента, показывающего что значение из одной строчки ценятся чуть больше чем значение из другой.
- Среднеусеченное - это такое среднее значение которое вычисляется после отсечения предельных значений. Такое понимание среднего позволяет обрезать "хвосты"распределения (что не всегда уместно, особенно при резком перекосе асимметрии распределения).
- Медиана (50й процентиль, 0,5-квантиль, Q2) - значение, которое находится в середине упорядоченного набора данных.
- Мода - наиболее часто встречающаяся категория или значение в наборе данных.

5.4 Оценка вариабельности

Вариабельность (дисперсность, variance) показывает насколько наборы данных расположены плотно друг к другу. Из оценок на основе дисперсности можно выяснить насколько строки данных логически близки друг к другу, то есть сделать вывод либо "данные слишком разнородны чтобы их как-то ообщать"либо "подавляющее большинство данных кучкуется вместе в пределах таких-то значений".

Самый первый количественный показатель тут это стандартное отклонение. Оно говорит насколько данные в наборе в целом разросаны от центрального значения.

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

```
1    data_frame.std(numeric_only=True)
```

```
id                          4.194741e+06
has_test                    1.608025e-01
response_letter_required    1.793307e-01
salary_from                 8.322305e+04
salary_to                   1.511755e+06
area_id                     2.038774e+02
dtype: float64
```

Рис. 7: оценка вариабельности

Среднее абсолютное отклонение (mean absolute deviation) - среднее абсолютных значений отклонений от среднего, имеет смысл использоать в случае большого числа "выбросов поскольку в отличие от std(), mad() к ним менее чувствителен.

$$\bar{d} = \frac{\sum_{i=1}^{n} \mid x_i - \bar{x} \mid}{n}$$

Статистические показатели на основе сортированных (ранжированных) данных называются порядковыми статистиками. Элементарная мера — это размах: разница между самым крупным и самым малым значениями. Во избежание чувствительности к выбросам вы можете обратиться к размаху данных после отбрасывания значений с каждого конца. Эти

типы оценок формально основываются на разницах между процентилями. В наборе данных P-й процентиль является таким значением, что, по крайней мере, P процентов значений принимает это значение или меньшее, и по крайней мере (100 - P) процентов значений принимает это значение или большее.

5.5 Оценки распределения

Разведывание распределения данных показывает как данные расположены в совокупности. Здесь активно используются визуальные инструменты.

В статистической теории центральное положение и вариабельность упоминаются как моменты распределения первого и второго порядка. Моменты третьего и четвертого порядка — это асимметрия и эксцесс.

Асимметрия (skewness) - это мера асимметрии распределения. Это значение может быть положительным или отрицательным.

- Отрицательная асимметрия указывает на то, что хвост находится в левой части распределения, которая простирается в сторону более отрицательных значений.
- Положительная асимметрия указывает на то, что хвост находится на правой стороне распределения, которая простирается в сторону более положительных значений.
- Нулевое значение указывает на то, что в распределении вообще нет асимметрии, что означает, что распределение совершенно симметрично.

Эксцесс (kurtosis) - это мера того, является ли распределение тяжелым или легким хвостом по сравнению с нормальным распределением.

- Эксцесс нормального распределения равен 3.
- Если данное распределение имеет эксцесс меньше 3, говорят, что оно является игровым , что означает, что оно имеет тенденцию производить меньше и менее экстремальных выбросов, чем нормальное распределение.
- Если данное распределение имеет эксцесс больше 3, говорят, что оно лептокуртическое , что означает, что оно имеет тенденцию производить больше выбросов, чем нормальное распределение.

5.6 Matplotlib

Один из инструментов визуализации позволяет строить различные двухмерные и трехмерные графики. Очень хорошо подходит для статистики. Большинство статистической визуализации которое вы увидите при изучении темы где-то в интернете обычно сделано либо через matplotlib либо через seaborn, иногда с использованием bokeh.

5.7 boxplot

Коробчатая диаграмма (иногда в русскоязычной литературе называется "коробка с усами") позволяет наглядно продемонстрировать плотноть распределения данных. Она основана на процентилях, то есть показывает выше и ниже какого уровня распределен набор значений. На ней показываются медианное значение - линия посередине коробки, 75 и 25 процентили - верхняя инижняя граница коробки (IQR), "усы"коробки обозначают "максимумы"и "минимумы"набора данных, а точками обозначаются "выбросы те значения, встретить которые маловерятно.

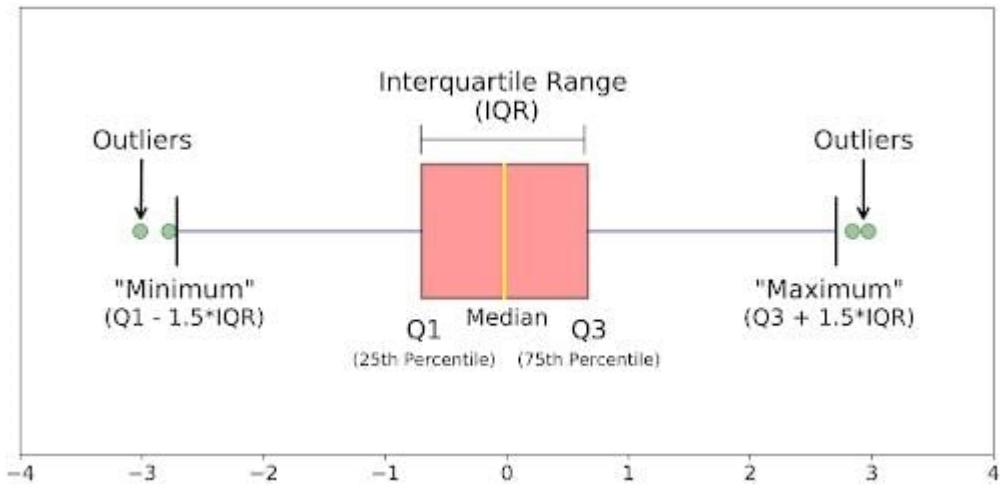

Рис. 8: коробчатая диаграмма

5.8 histogram

Гистограмма относится к частоте появления переменных в интервале.

```
1    data_frame[column_of_interest]
2        .value_counts()
3        .nlargest(40)
4        .plot(kind='bar', figsize=(15,5))
5        # nlargest(x) возвращает x наибольших значений.
6    plt.title("Частота предложений по зарплате")
7    plt.ylabel('Число предложений')
8    plt.xlabel('Salary from')
```

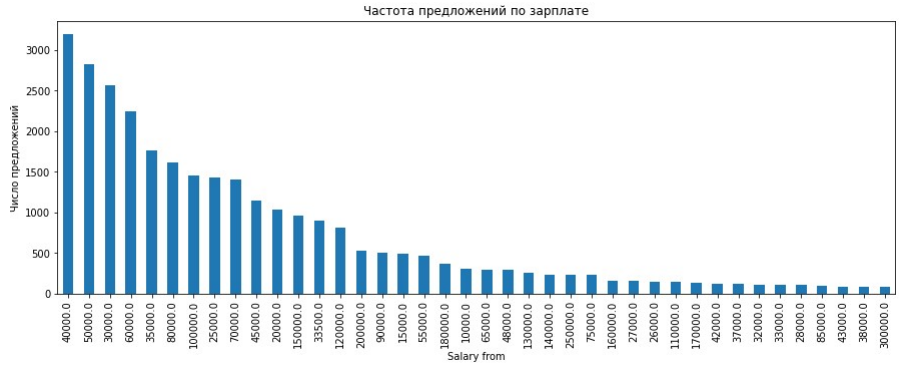

Рис. 9: гистограмма

5.9 Корреляция

Развед анализ также предполагает выяснение зависимости между исследуемыми переменными. Матрица корелляции - один из основных способов понять взаимосвязь между переменными.

```
1    plt.figure(figsize=(10,5))
2    c = data_frame[[column_of_interest, "salary_to", "area_id"]]
3    .corr()
4    sns.heatmap(c, cmap="BrBG", annot=True)
```

	salary_from	salary_to	area_id
salary_from	1.000000	0.043354	-0.007019
salary_to	0.043354	1.000000	-0.000718
area_id	-0.007019	-0.000718	1.000000

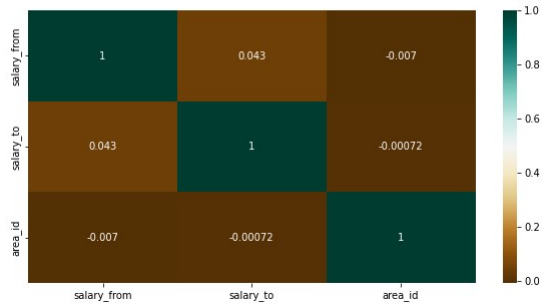

Рис. 10: матрица корреляции

6 Стандартизация данных

6.1 Смысл стандартизации данных

Иногда при работе с данными требуется привести все признаки (фичи/колонки) к некоему "общему знаминателю". Такая ситуация может быть в тех случаях когда потребителем ваших данных является не человек который может прочитать объяснение по датасету и сделать выводы, а какая-то другая программа, например та, в которой реализуется алгоритм машинного обучения. Стандартизация включает в себя приведение значений, существующих в разных шкалах, к общей шкале, что упрощает их сравнение. Это важно при построении моделей машинного обучения, поскольку вам нужно, чтобы распределение значений столбца не было чрезмерно или недостаточно представлено в ваших моделях.

6.2 Z-score

Z-оценка это оценка того насколько далеко находится значение от центрального положения.

$$Z = \frac{x - \mu}{\sigma}$$

- μ - среднее значение всей выборки
- σ - std()

Стандартизировать набор данных означает масштабировать все значения в наборе данных таким образом, чтобы среднее значение равнялось 0, а стандартное отклонение равнялось 1.

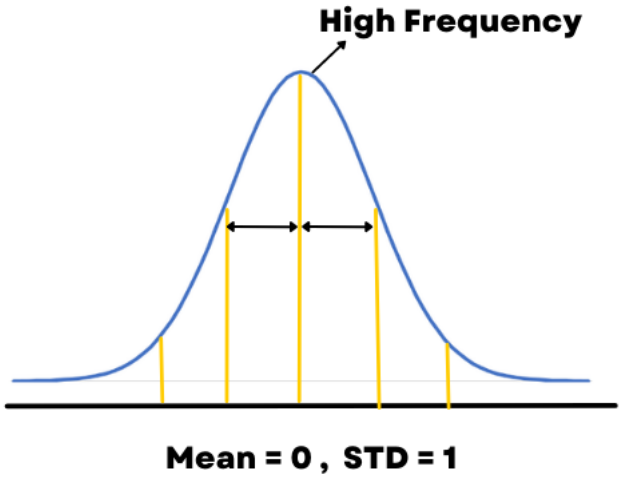

Рис. 11: Z-score

6.3 sklearn StandardScaler / MinMaxScaler

Библиотека scikit-learn (sklearn) это еще один инструмент в вашем наборе. Это библиотека для машинного обучения с открытим исходным кодом, она позволяет решать задачи обучения с учителем и без, умеет

в оценку моделей. Помимо этого она предоставляет набор инструментов для предобработки данных, именно последнее нас на сегодняшний день пока что и интересует.

В нем есть реализация нескольких методов скейлинга для данных StandardScaler, MaxAbsScaler, MinMaxScaler.

StandardScaler удаляет среднее значение и масштабирует данные до единичной дисперсии. Однако выбросы влияют на вычисление эмпирического среднего значения и стандартного отклонения. StandardScaler не может гарантировать сбалансированные масштабы признаков при наличии выбросов (при аутлаерах будут перекосы)

MinMaxScaler изменяет масштаб набора данных таким образом, чтобы все значения функций находились в диапазоне [0, 1]. То есть все очено просто - самое большое значение в выборке преобразуется в 1, самое маленькое - в 0, а остальные в зависимости от величины - между ними.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

MaxAbsScaler довольно похож на MinMaxScale. максимальное абсолютное масштабирование (maximum absolute scaling) изменяет масштаб каждого признака, чтобы в колонках было значение от -1 до 1. (-1 только если есть отрицательные значения, иначе [0,1]

$$x_{scaled} = \frac{x}{max(|x|)}$$