

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
**Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

С. В. Рындина

БАЗОВЫЕ ВОЗМОЖНОСТИ ЯЗЫКА PYTHON ДЛЯ АНАЛИЗА ДАННЫХ

Учебно-методическое пособие

ПЕНЗА 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

С. В. Рындина

Базовые возможности языка Python для анализа данных

Учебно-методическое пособие

Пенза
Издательство ПГУ
2022

УДК 65.29я7
Р95

Р е ц е н з е н т
кандидат технических наук,
доцент
А. А. Масленников

Рындина, Светлана Валентиновна.

Р95 Базовые возможности языка Python для анализа данных :
учеб.-метод. пособие / С. В. Рындина. – Пенза : Изд-во ПГУ,
2022. – 72 с.

Рассмотрены возможности языка программирования Python для анализа данных. Приведены задания для лабораторных работ по разведочному, описательному и исследовательскому анализу данных. Материал соответствует программе дисциплины «Анализ данных» и может быть использован при изучении дисциплин «Бизнес-аналитика на основе больших данных», «Интеллектуальные информационные системы», а также при написании выпускной работы бакалавра.

Издание подготовлено на кафедре «Цифровая экономика» и предназначено для обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика».

УДК 65.29я7

© Пензенский государственный
университет, 2022

СОДЕРЖАНИЕ

Введение	4
1. Среда разработки для языка Python: Anaconda, Google colab	5
2. Данные. Библиотеки pandas и NumPy	11
2.1. Типы данных и структуры. Построение срезов	11
2.2. Импорт данных. Подготовка данных.....	21
3. Описательный анализ данных. Библиотеки pandas и NumPy	32
4. Разведочный анализ данных. Библиотеки Matplotlib и Seaborn	43
4.1. Библиотека Matplotlib	43
4.2. Библиотека Seaborn	48
5. Проверка статистических гипотез. Библиотека SciPy, модуль stats	57
5.1. Тестирование гипотез для одной выборки	63
5.2. Тестирование гипотез для двух выборок.....	66
6. Лабораторные работы	68
6.1. Лабораторная работа «Импорт и преобразование данных».....	68
6.2. Лабораторная работа «Описательный анализ данных»	68
6.3. Лабораторная работа «Визуализация данных. Разведочный анализ данных»	69
6.4. Лабораторная работа «Исследовательский анализ данных».....	70
Список литературы	71

Введение

В соответствии с учебным планом студенты второго курса направления подготовки 38.03.05 «Бизнес-информатика» изучают дисциплину «Анализ данных».

Описательный анализ данных, разведочный анализ данных и проверка статистических гипотез – базовые методы работы с данными.

Целью учебно-методического пособия является рассмотрение основных типов данных и аналитических приемов работы с ними: преобразование, очистка, изучение распределений данных, тестирование гипотез для ответов на исследовательские вопросы о контексте представления данных, полезные визуализации для получения инсайтов, формулирования исследовательских вопросов, формирования отчетов и презентации результатов анализа.

В пособии рассмотрен язык программирования Python, облачный ресурс Google Colab (Colaboratory) и Anaconda – дистрибутив Python со встроенным в него пакетным менеджером conda и виртуальной средой Jupyter Notebook для работы с кодом по анализу данных на этом языке программирования.

Также приведены задания лабораторных работ по анализу данных, в которых используются представленные методы анализа и инструментальные средства для выполнения анализа данных.

1. Среда разработки для языка Python: Anaconda, Google colab

Для выполнения анализа данных с помощью языка Python установим дистрибутив Anaconda с официального сайта <https://www.anaconda.com/distribution/> (рис. 1.1).

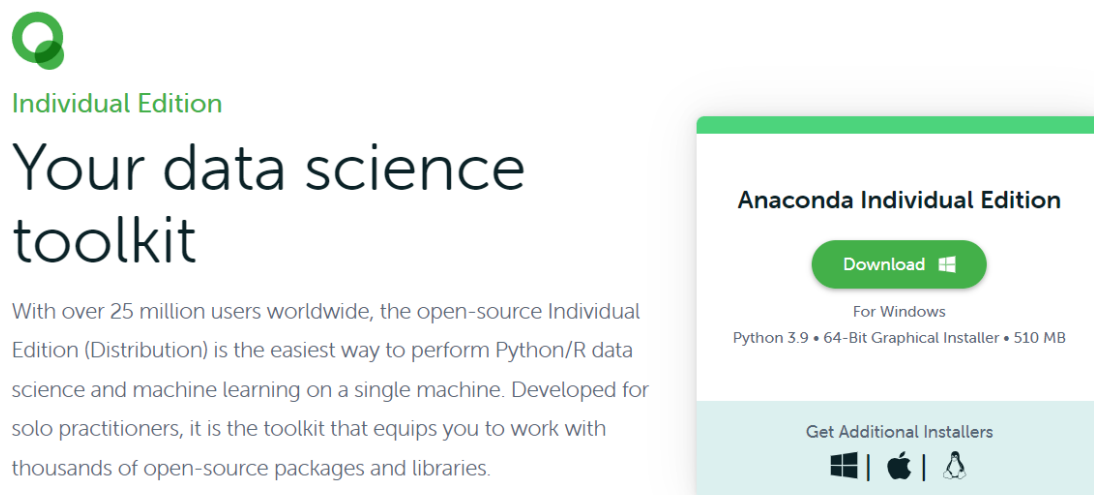


Рис. 1.1. Персональная редакция Anaconda

На момент написания пособия актуальная версия Anaconda Individual Edition для Python 3.9. Однако в пособии используется немного более ранняя версия для Python 3.7. Кроме open source версии Anaconda, есть и другие: коммерческая, командная и платформенная. Они предоставляют более широкие возможности по работе с данными и используют разные тарифные планы.

В дистрибутив Anaconda входит виртуальная среда по работе с данными Jupyter Notebook. Этот инструмент – интерактивная среда разработки, позволяющая сразу увидеть результат исполнения кода на языке Python. Запускать код на исполнение можно небольшими блоками, можно дополнять код текстовыми блоками Markdown, что делает исследовательскую работу с данными еще более удобной.

Возможность использовать для аналитики данных язык R также присутствует в Anaconda. Работать с кодом на языке R можно либо через терминал (Anaconda Prompt), либо через RStudio – интерактивную среду разработки на языке R, устанавливаемую через Anaconda Navigator.

Очень часто в платформенных, комплексных решениях для аналитиков данных или исследователей данных (data scientists) поддерживаются оба языка Python и R. Так, облачное решение RStudio Cloud [1] также позволяет писать код на двух языках (разумеется, не в одном

скрипте) и кроме проектов RStudio на языке R поддерживает создание кода на языке Python в Jupyter Notebook (рис. 1.2). Однако такая возможность имеется пока только в платных тарифных планах.

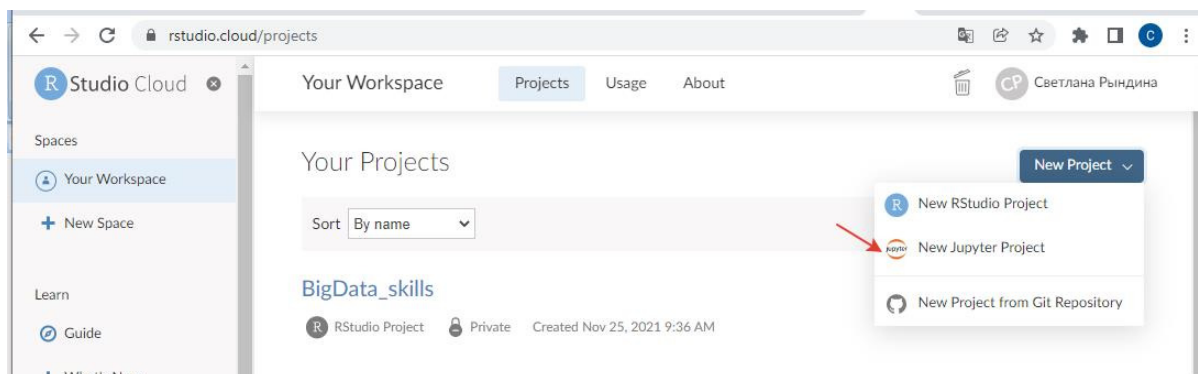


Рис. 1.2. RStudio Cloud

Вернемся к Anaconda. Это дистрибутив Python и R, который включает:

- пакеты для анализа данных (предустановленные и легко подключаемые) на языках Python и R;
- управление библиотеками, зависимостями и средами с помощью собственного менеджера Conda;
- библиотеки для настройки моделей машинного обучения и глубокого обучения: scikit-learn, TensorFlow и Theano;
- библиотеки для анализа данных, управления масштабируемостью и производительностью Dask, NumPy, pandas и Numba;
- возможности визуализации данных с помощью библиотек Matplotlib, Bokeh, Datashader и seaborn;
- библиотека для работы с математическими и статистическими моделями SciPy;
- Spyder (IDE/редактор) и Jupyter.

Экосистема Anaconda представлена на рис. 1.3.

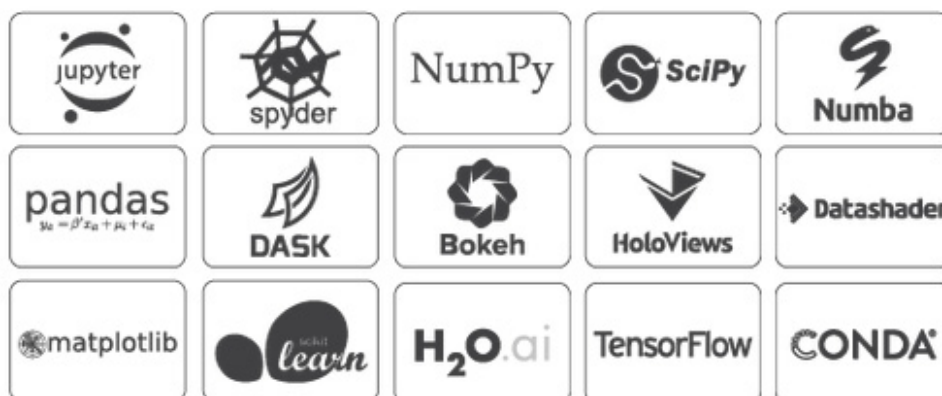


Рис. 1.3. Возможности Anaconda

При установке Anaconda необходимо обратить внимание на следующие параметры: если не устанавливаете программу для всех пользователей (в Windows для этого нужны привилегии Администратора) выберите вариант установки «Только я» (Just Me). В качестве папки для установки Anaconda выберите путь, который не содержит имен папок с пробелами в названии (таких как, например, Program Files) и не содержит не английских символов юникода (например, русских букв). В противном случае при подключении пакетов могут возникнуть проблемы интеграции.

Выберите, нужно ли добавлять Anaconda в переменную окружения PATH. Рекомендуется не делать этого, потому что это может повлиять на работу других программ (рис. 1.4).

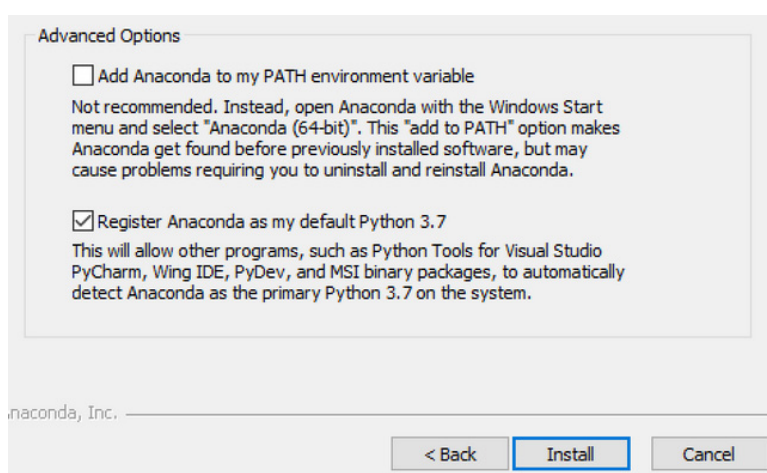


Рис. 1.4. Диалоговое окно инсталляции Anaconda

В Anaconda есть возможность работать с Jupyter Notebook. Notebook состоит из блоков кода или текста и имеет расширение .ipynb. Создать новый файл можно с домашней страницы (<http://localhost:8888/tree#>), которая открывается в браузере при запуске Jupyter Notebook (компьютер пользователя – локальный хост, выход в интернет не требуется). При выборе доступного языка Python 3 (рис. 1.5) открывается новый notebook (рис. 1.6).

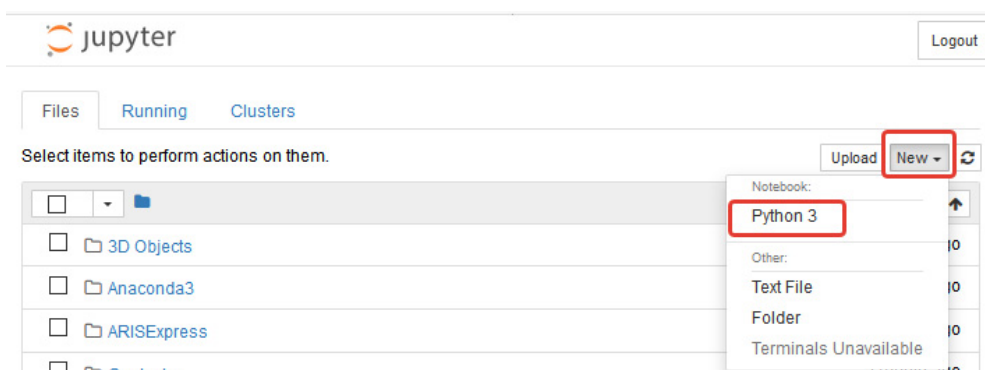


Рис. 1.5. Создание нового notebook

Новый блок в notebook добавляется с помощью кнопки **+** на панели инструментов. По умолчанию добавляется блок для ввода кода (рис. 1.6), но параметры блока можно изменить, выбрав в раскрывающемся списке Markdown (инструмент работы с текстом, который включает в себя массу возможностей работы с формулами и не только).

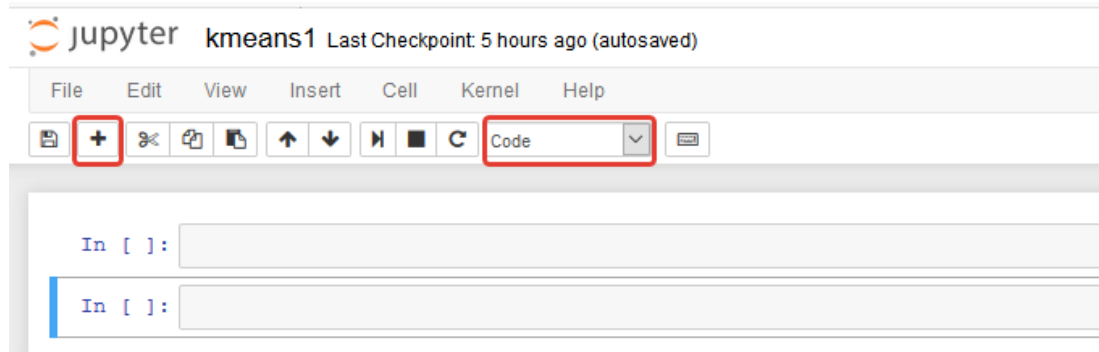



Рис. 1.6. Добавление новых блоков в notebook

Код в блоке с маркером `In []` можно редактировать и посылать на выполнение любое количество раз с помощью кнопки . Сразу под блоком кода, отправленного на выполнение, если этот код предусматривал вывод, появится блок `Out []` – с результатом исполнения кода. В квадратных скобках отображается порядковый номер отправленного на исполнение блока в текущем notebook (т.е. счетчик), результат исполнения кода в этом блоке будет иметь тот же порядковый номер только с маркером `Out` (рис. 1.7).

```
In [1]: import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
dataset = pd.read_csv('insurance.csv')
dataset.head(10)
```

Out[1]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960
7	37	female	27.740	3	no	northwest	7281.50560
8	37	male	29.830	2	no	northeast	6406.41070
9	60	female	25.840	0	no	northwest	28923.13692

Рис. 1.7. Результат выполнения кода в notebook

Браузерное решение (доступ через интернет) от компании Google: Colaboratory позволяет справиться с проблемой малой мощности компьютера для анализа данных [2].

Colaboratory или сокращенно Colab – это бесплатная среда для ноутбуков Jupyter, предоставляемая Google, где можно использовать бесплатные графические процессоры и TPU.

Для получения доступа к возможностям Colab нужно создать учетную запись в Google – клиента @gmail.com. Далее перейти по ссылке <https://colab.research.google.com>.

Откроется доступ к Welcome To Colaboratory и появится возможность создать новый notebook с помощью Google Colab. Стартовое окно Google Colab (рис. 1.8) содержит следующие опции:

- **Examples (примеры):** Содержит несколько Jupyter Notebook с примерами;
- **Recent (последние):** Jupyter Notebook, с которым недавно работали;
- **Google Drive:** сохраненные на вашем диске Google Jupyter Notebook;
- **GitHub:** доступ в репозиторий github для загрузки Jupyter Notebook в Colab (доступен при подключении к GitHub);
- **Upload (загрузить):** загрузка Jupyter Notebook из вашего локального каталога.

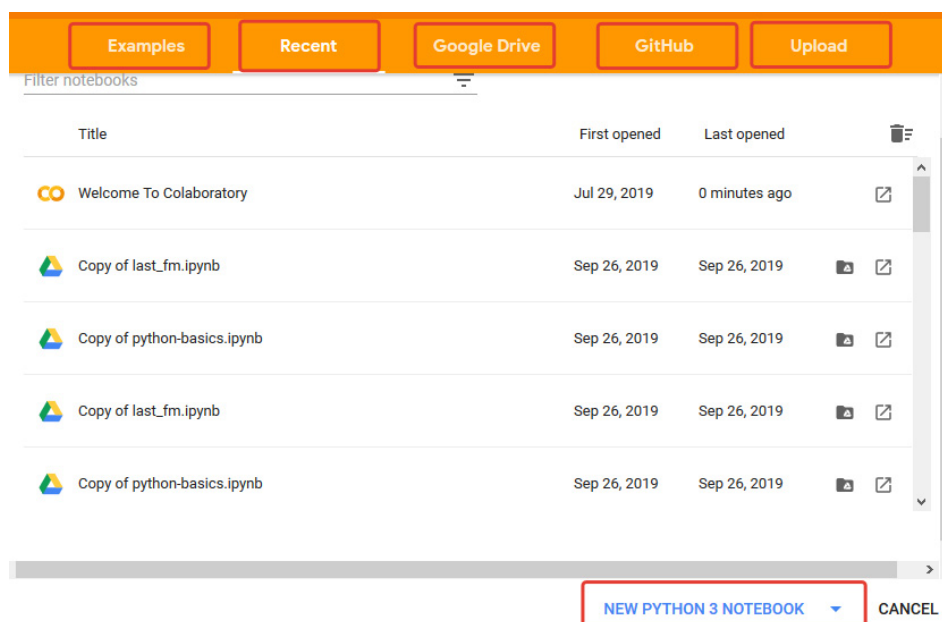



Рис. 1.8. Стартовое окно Google Colab

Новый файл Untitled.ipynb будет сохранен на подключенный к аккаунту @gmail.com Google Диск. Как и в Jupyter из дистрибутива Anaconda, файл notebook состоит из блоков текста и кода. В панели

инструментов добавление блока – это кнопка + рядом с нужной опцией, но можно навести мышь на центр блока снизу, и те же кнопки будут доступны для выбора (рис. 1.9). Код можно сразу передавать на выполнение кнопкой  слева от блока кода.

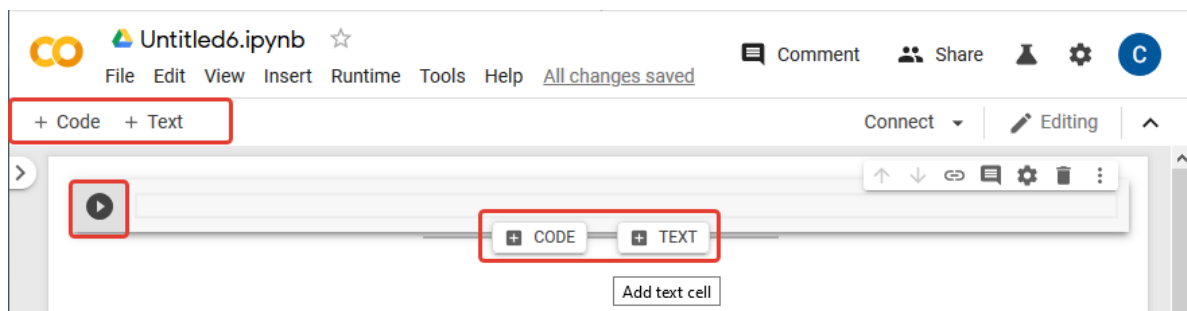


Рис. 1.9. Добавление блоков в notebook

Аналогично пакетам языка R [1] в языке Python для выполнения анализа данных используются модули (библиотеки). Модуль – файл с расширением .py, содержащий определения и другие инструкции языка Python.

Наиболее часто используемые модули для анализа данных уже входят в Anaconda и Google Colab, нет необходимости их загружать.

В начальном блоке notebook все необходимые библиотеки подключаются с помощью инструкции `import`. Но можно это сделать и позже, главное обращаться к функциям библиотеки после ее подключения.

```
# Импортируем библиотеку inspect
import inspect
```

После подключения, используя имя модуля, можно получить доступ к его функциям: `<имя модуля>.<имя функции>()`.

Можно подключать библиотеку, сразу задавая сокращенное имя для обращения к функциям библиотеки.

```
import pandas as pd
# Загружаем набор данных
iris_df = pd.read_csv('iris.csv')
```

Можно импортировать имена функций из модуля непосредственно в текущее пространство имен:

```
import matplotlib.pyplot as plt
```

Или импортировать конкретные функции из модуля.

```
from sklearn import datasets
# Загружаем встроенный набор данных
iris_df = datasets.load_iris()
```

Для построения графиков в Python будем использовать библиотеку `matplotlib` [3] и `seaborn` [4], для работы с многомерными массивами – `NumPy` [5], для работы с данными (чтения файлов, работы с пропусками и т.п.) – библиотеку `pandas` [6], для тестирования статистических гипотез – библиотеку `SciPy` модуль `stats` [7].

2. Данные.

Библиотеки pandas и NumPy

Любой проект по анализу данных начинается с данных. У данных есть тип и структура.

2.1. Типы данных и структуры.

Построение срезов

Все типы в Python являются объектами, при создании которых вызывается специальная функция – конструктор. Создание объекта любого типа связано с выделением памяти для размещения данных об этом объекте.

Переменная (variable) – это идентификатор, который указывает на определенную область памяти, где хранятся произвольные данные – созданный объект (значение переменной). Переменная лишь указывает на данные – хранит ссылку, а не сами данные. Если на созданный ранее объект ссылок в коде уже нет (например, переменная, ссылавшаяся на этот объект, теперь ссылается на другой), то объект удаляется с помощью сборщика мусора (Garbage Collection, GC), а занимаемая этим объектом память очищается. Python – язык со встроенным менеджером управления памятью, который выполняет данные операции автоматически.

Переменным необходимо давать информативные имена, по которым можно было бы понять, с какими данными она связана. Для задания имен переменных используется змеиный регистр (snake_case), используется либо только верхний регистр букв, либо только нижний (например, TAX_USA, shop_df).

Переменная должна быть проинициализирована (ссылаться на данные) перед использованием в выражении.

В Python есть несколько стандартных типов данных:

- числа (Numeric Type):
 - int – целое число;
 - float – число с плавающей точкой;
 - complex – комплексное число;
- логический тип (Boolean Type):
 - bool.

В Python существует специальное значение None типа NoneType, обозначающее нейтральное или «нулевое» поведение.

Коллекции (контейнеры) – это группа типов данных, которые содержат в себе другие данные и поддерживают [7]:

- проверку на вхождение элементов in и not in (True/False);
- определение размера len();
- возможность выполнения итераций (перемещения по элементам последовательности).

К коллекциям относят:

- строки (Text Sequence Type):
 - str;
- списки (Sequence Type):
 - list – список;
 - tuple – кортеж;
 - range – диапазон;
- множества (Set Types):
 - set – множество;
 - frozenset – неизменяемое множество;
- словари (Mapping Types):
 - dict.

Эти типы данных можно, в свою очередь, классифицировать по нескольким признакам:

- изменяемые (списки, словари и множества);
- неизменяемые (числа, строки и кортежи);
- упорядоченные (списки, кортежи, строки и словари);
- неупорядоченные (множества).

Последовательность – это упорядоченная коллекция (str, list, tuple и range), поддерживающая индексированный доступ к элементам. Получить доступ к отдельному элементу или группе элементов последовательности возможно с помощью оператора []. Индексация начинается с 0.

Для последовательностей можно выделить общие операции, например

`len(s)` #Длина (количество элементов) последовательности s

Также для каждого типа данных существуют характерные операции (дополнительные методы).

Множества поддерживают математические операции, характерные для множеств (пересечение, объединение и др.).

Тип данных dict (словарь) – это неупорядоченная коллекция пар элементов «ключ-значение».

Все объекты независимо от типа поддерживают ряд общих функций.

```
id(object) # Возвращает для object уникальный идентификатор (адрес в памяти)
help([object]) # Отображает справку для object.
type(object) # Возвращает тип object.
```

Операции над объектами выполняются в определенном порядке [8]:

1. `**`.
2. `-x`, `+x`.
3. `,` `/`, `//`, `%`.
4. `+`, `-`.
5. `<`, `<=`, `>`, `>=`, `!=`, `==`.
6. `is`, `is not`.
7. `in`, `not in`.
8. `not`, `and`, `or`.

Изменение порядка выполнения операций можно производить за счет использования скобок.

Имена функций задаются маленькими буквами, с подчеркиваниями между словами (например, `get_names`).

Имена классов задаются словами с заглавными (прописными) буквами без пробелов, это так называемый CamelCase или «верблюжья нотация» (например, `PlayMarket`).

Рассмотрим типы данных, которые есть в библиотеке **pandas**, и как они соотносятся со стандартными типами Python и типами данных в библиотеке NumPy (табл. 2.1).

Таблица 2.1

Типы данных в библиотеке **pandas** [9]

Pandas dtype	Стандартный тип данных Python	NumPy dtype	Описание
object	str	string_, unicode_	Текст
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Целое число
float64	float	float_, float16, float32, float64	Число с плавающей точкой
bool	bool	bool_	Логическое значение
datetime64	NA	NA	Дата и время
timedelta[ns]	NA	NA	Разница во времени
category	NA	NA	Список текстовых значений ограниченной длины

В библиотеке NumPy есть объект `ndarray`, который определяет структуру данных: многомерный однородный массив с заранее заданным количеством элементов.

Простейший способ определить новый объект `ndarray` – использовать функцию `array()`.

```
import numpy as np #Подключение библиотеки (сокращенное имя)
a = np.array([1, 2, 3]) #Создание массива
```

При импорте библиотеки `numpy` ей задается псевдоним `np`, он является общепринятым среди пользователей этой библиотеки.

Библиотека `pandas` предоставляет две структуры: `Series` и `DataFrame`.

С `Series` можно работать как с обычным массивом (обращаться по номеру индекса) и как с ассоциированным массивом, когда можно использовать ключ для доступа к элементам данных. `DataFrame` – это двумерная маркированная структура, фактически таблица данных.

Конструктор класса `Series`:

```
class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False,
                    fastpath=False)
```

Параметры класса `Series` описаны в табл. 2.2.

Таблица 2.2

Параметры класса `Series()`

Параметр	Описание
<code>data</code>	Массив, словарь или скалярное значение, на базе которого будет построен <code>Series</code>
<code>index</code>	Список меток, который будет использоваться для доступа к элементам <code>Series</code> . Длина списка должна быть равна длине <code>data</code>
<code>dtype</code>	Объект <code>numpy.dtype</code> , определяющий тип данных
<code>copy</code>	Создает копию массива данных, если параметр равен <code>True</code> , в ином случае ничего не делает

Пример создания структуры `Series` с помощью Jupyter (Anaconda) на рис. 2.1. (были определены только первые два параметра). При импорте библиотеки `pandas` ей задается псевдоним `pd`, он является общепринятым среди пользователей этой библиотеки.

```
In [1]: import pandas as pd
        employed = pd.Series([1, 2, 3, 4, 5], ['Anna', 'Maria', 'Ivan', 'Andrey', 'Alexander'])
        employed

Out[1]: Anna      1
        Maria     2
        Ivan      3
        Andrey    4
        Alexander  5
        dtype: int64
```

Рис. 2.1. Создание объекта `Series`

Конструктор класса DataFrame:

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
```

Параметры класса DataFrame описаны в табл. 2.3.

Таблица 2.3

Параметры класса DataFrame()

Параметр	Описание
data	Массив ndarray, словарь или другой DataFrame
index	Список меток для записей (имена строк таблицы)
columns	Список меток для полей (имена столбцов таблицы)
dtype	Объект numpy.dtype, определяющий тип данных
copy	Создает копию массива данных, если параметр равен True, в ином случае ничего не делает

Пример создания структуры DataFrame с помощью Jupyter (Anaconda) на рис. 2.2.

```
In [4]: import pandas as pd
empl = {"post":pd.Series(['manager', 'manager', 'assistant'], index=['Anna', 'Maria', 'Ivan']),
        "phone": pd.Series([1111111111, 2222222222, 3333333333], index=['Anna', 'Maria', 'Ivan'])}
employed = pd.DataFrame(empl)
employed
```

```
Out[4]:
```

	post	phone
Anna	manager	1111111111
Maria	manager	2222222222
Ivan	assistant	3333333333

Рис. 2.2. Создание объекта DataFrame

Наиболее часто используемые операции для доступа к элементам DataFrame представлены в табл. 2.4.

Таблица 2.4

Доступ к элементам DataFrame [10]

Операция	Синтаксис	Возвращаемый результат	Пример
Выбор столбца	df[col]	Series	Рис. 2.3
Выбор строки по метке	df.loc[label]	Series	Рис. 2.4
Выбор строки по индексу	df.iloc[loc]	Series	Рис. 2.5
Срез по строкам	df[0:4]	DataFrame	Рис. 2.6
Выбор строк, отвечающих условию	df[bool_vec]	DataFrame	Рис. 2.7

```
In [5]: #Операция - выбор столбца
employed['post']

Out[5]: Anna      manager
Maria      manager
Ivan      assistant
Name: post, dtype: object
```

Рис. 2.3. Выбор столбца

```
In [6]: #Операция - выбор строки по метке
employed.loc['Ivan']

Out[6]: post      assistant
phone      3333333333
Name: Ivan, dtype: object
```

Рис. 2.4. Выбор строки по метке

```
In [7]: #Операция - выбор строки по индексу
employed.iloc[0]

Out[7]: post      manager
phone      1111111111
Name: Anna, dtype: object
```

Рис. 2.5. Выбор строки по индексу

```
In [8]: #Операция - срез по строкам
employed[1:2]

Out[8]:
```

	post	phone
Maria	manager	2222222222

Рис. 2.6. Срез по строкам

```
In [9]: #Операция - выбор строк, отвечающих условию
employed[employed['post']=='manager']

Out[9]:
```

	post	phone
Anna	manager	1111111111
Maria	manager	2222222222

Рис. 2.7. Выбор строк, отвечающих условию

В библиотеке `pandas` есть возможность создания категориальных переменных, как номинальных, так и порядковых, с помощью объекта `Categorical`.

Категориальные показатели принимают ограниченный набор дискретных значений. Например, номинальные показатели: семейное положение, цвет, марка, бренд, архитектурный стиль, к ним относятся и показатели, принимающие всего два возможных значения: да/нет, есть/отсутствует, купил / не купил, отменил подписку / оплатил следующий период, ушел/остался, болен/здоров и т.п. А порядковые категориальные показатели можно упорядочить по степени проявления регистрируемого проявления свойства: рейтинг услуг (очень плохо / плохо / хорошо / очень хорошо), рейтинг гостиниц, ресторанов, рейтинг студентов, рейтинг компаний, уровень квалификации, инвестиционный рейтинг [10].

Создадим два объекта `Categorical`, один для номинального показателя пол (`gender`), другой для порядкового показателя размер (`size`). Для номинального показателя `gender` параметр `categories` не задан. Для порядкового показателя `size` параметр `categories` имеет важное значение и показывает порядок следования уровней показателя (традиционно от меньшего/худшего к большему/лучшему). Например, порядковый показатель, задающий статус клиента в программе лояльности. Статусы связаны с различными привилегиями, в том числе с размером предоставляемой скидки. Переменная `client_grade`, используемая для фиксации статуса клиента и принимающая значения `silver/gold/platinum`, имеет упорядочение от начального статуса к самому продвинутому. Или переменная `developer_grade`, определяющая несколько уровней для разработчика (по компетенциям, сложности проектов и оплате труда): `junior/middle/senior`.

```
import pandas as pd
gender_val = ["female", "male", "male", "female", "male", "female", "female", "male"]
gender_cat = pd.Categorical(gender_val)
gender_cat
size_val = ["XS", "L", "M", "XL", "XS", "S"]
size_cat = pd.Categorical(size_val, categories = ["XS", "S", "M", "L", "XL"])
size_cat
```

Выполнение кода для определения показателя пол (`gender`) представлено на рис. 2.8. Как видим, порядок значений установлен по алфавиту. Если определить показатель размер (`size`) без задания параметра `categories` у объекта `Categorical` (рис. 2.9), то порядок уровней также будет установлен по алфавиту, что не соответствует правильной размерной линейке. А с заданием параметра `categories` для показателя размер (`size`) размерная линейка становится верной (рис. 2.10). Параметр `categories` задает верный порядок следования значений категорий в порядке их возрастания.

```
In [14]: import pandas as pd
gender_val = ["female", "male", "male", "female", "male", "female", "female", "male"]
gender_cat = pd.Categorical(gender_val)
gender_cat

Out[14]: [female, male, male, female, male, female, female, male]
Categories (2, object): [female, male]
```

Рис. 2.8. Создание категориальной переменной `gender_cat`

```
In [15]: size_val = ["XS", "L", "M", "XL", "XS", "S"]
size_cat = pd.Categorical(size_val)
size_cat

Out[15]: [XS, L, M, XL, XS, S]
Categories (5, object): [L, M, S, XL, XS]
```

Рис. 2.9. Создание категориальной переменной `size_cat`

```
In [16]: size_val = ["XS", "L", "M", "XL", "XS", "S"]
size_cat = pd.Categorical(size_val, categories = ["XS", "S", "M", "L", "XL"])
size_cat

Out[16]: [XS, L, M, XL, XS, S]
Categories (5, object): [XS, S, M, L, XL]
```

Рис. 2.10. Создание категориальной переменной `size_cat`

Каждой категории объекта `Categorical` присваивается целочисленное значение (код). Эти значения можно посмотреть с помощью свойства `.codes` (рис. 2.11, 2.12).

```
In [17]: gender_val = ["female", "male", "male", "female", "male", "female", "female", "male"]
gender_cat = pd.Categorical(gender_val)
gender_cat.codes

Out[17]: array([0, 1, 1, 0, 1, 0, 0, 1], dtype=int8)
```

Рис. 2.11. Позиции категориального показателя `gender_cat`

```
In [19]: size_val = ["XS", "L", "M", "XL", "XS", "S"]
size_cat = pd.Categorical(size_val, categories = ["XS", "S", "M", "L", "XL"])
size_cat.codes

Out[19]: array([0, 3, 2, 4, 0, 1], dtype=int8)
```

Рис. 2.12. Позиции категориального показателя `size_cat`

Теперь становится понятным, для чего необходимо было упорядочивание уровней категориального признака: размер XS получил наименьший код – 0, а размер XL получил наибольший код – 4. И эти числовые значения выступают метками уровней категорий, причем задают верный порядок их следования.

Особенности работы с категориальными показателями в составе таблицы данных (`DataFrame`) рассмотрим на примере.

Создадим объект `DataFrame` (рис. 2.13).

```
import pandas as pd
shop = {"gender":pd.Series(["female", "male", "male", "female", "male", "female"]),
        "size":pd.Series(["XS", "L", "M", "XL", "XS", "S"]),
        "totalsum":pd.Series([3000, 2456, 8965, 4567, 3487, 955])}
shop_df = pd.DataFrame(shop)
```

```
In [22]: import pandas as pd
shop = {"gender":pd.Series(["female", "male", "male", "female", "male", "female"]),
        "size":pd.Series(["XS", "L", "M", "XL", "XS", "S"]),
        "totalsum":pd.Series([3000, 2456, 8965, 4567, 3487, 955])}
shop_df = pd.DataFrame(shop)
shop_df
```

Out[22]:

	gender	size	totalsum
0	female	XS	3000
1	male	L	2456
2	male	M	8965
3	female	XL	4567
4	male	XS	3487
5	female	S	955

Рис. 2.13. Создание объекта DataFrame

Преобразуем столбец показателя пол (gender) в числовые метки. Создадим новый столбец gender_cat в наборе данных shop_df и запишем в него числовые метки (рис. 2.14). Числовые метки из столбца gender извлекаются с помощью атрибута .cat.codes.

```
shop_df["gender"] = pd.Categorical(shop_df["gender"])
shop_df["gender_cat"] = shop_df["gender"].cat.codes
shop_df
```

```
In [24]: shop_df["gender"] = pd.Categorical(shop_df["gender"])
shop_df["gender_cat"] = shop_df["gender"].cat.codes
shop_df
```

Out[24]:

	gender	size	totalsum	gender_cat
0	female	XS	3000	0
1	male	L	2456	1
2	male	M	8965	1
3	female	XL	4567	0
4	male	XS	3487	1
5	female	S	955	0

Рис. 2.14. Преобразование категориального показателя в числовые метки

Преобразуем столбец показателя размер (size) в числовые метки. Создадим новый столбец size_cat в наборе данных shop_df и запишем в него числовые метки (рис. 2.15).

```
shop_df["size"] = pd.Categorical(shop_df["size"], categories = ["XS", "S", "M", "L", "XL"])
shop_df["size_cat"] = shop_df["size"].cat.codes
shop_df
```

```
In [25]: shop_df["size"] = pd.Categorical(shop_df["size"], categories = ["XS", "S", "M", "L", "XL"])
shop_df["size_cat"] = shop_df["size"].cat.codes
shop_df
```

```
Out[25]:
```

	gender	size	totalsum	gender_cat	size_cat
0	female	XS	3000	0	0
1	male	L	2456	1	3
2	male	M	8965	1	2
3	female	XL	4567	0	4
4	male	XS	3487	1	0
5	female	S	955	0	1

Рис. 2.15. Преобразование категориального показателя
в числовые метки

Добавление атрибута `.cat` к столбцу обусловлено тем, что доступ осуществляется не к объекту `Categorical`, а к столбцу `DataFrame`, тип данных которого `category` (рис. 2.16, 2.17).

```
In [41]: shop_df.dtypes
```

```
Out[41]: gender      category
size      category
totalsum    int64
gender_cat  int8
size_cat    int8
dtype: object
```

```
In [42]: shop_df["size"].cat
```

```
Out[42]: <pandas.core.arrays.categorical.CategoricalAccessor object at 0x000002C5FA347948>
```

```
In [43]: shop_df["size"].cat.categories
```

```
Out[43]: Index(['XS', 'S', 'M', 'L', 'XL'], dtype='object')
```

Рис. 2.16. Столбец набора данных `size`
(категориальный показатель)

```
In [4]: size_val = ["XS", "L", "M", "XL", "XS", "S"]
size_cat = pd.Categorical(size_val, categories = ["XS", "S", "M", "L", "XL"])
size_cat.dtype
```

```
Out[4]: CategoricalDtype(categories=['XS', 'S', 'M', 'L', 'XL'], ordered=False)
```

Рис. 2.17. Категориальная переменная `size_cat`

Попутно познакомимся с методами `.head()`, `.tail()`, `.take()`. Эти методы позволяют получать срезы набора данных: `.head(k)` – первые `k` записей/наблюдений (рис. 2.18), `.tail(k)` – последние `k` записей/наблюдений (рис. 2.19), `.take([k, l])` – записи/наблюдения с номерами `k` и `l` (рис. 2.20).

```
In [26]: shop_df.head(3)
```

```
Out[26]:
```

	gender	size	totalsum	gender_cat	size_cat
0	female	XS	3000	0	0
1	male	L	2456	1	3
2	male	M	8965	1	2

Рис. 2.18. Срез по первым наблюдениям

```
In [27]: shop_df.tail(3)
```

```
Out[27]:
```

	gender	size	totalsum	gender_cat	size_cat
3	female	XL	4567	0	4
4	male	XS	3487	1	0
5	female	S	955	0	1

Рис. 2.19. Срез по последним наблюдениям

```
In [29]: shop_df.take([0, 2, 4])
```

```
Out[29]:
```

	gender	size	totalsum	gender_cat	size_cat
0	female	XS	3000	0	0
2	male	M	8965	1	2
4	male	XS	3487	1	0

Рис. 2.20. Отбор наблюдений по номеру строки

2.2. Импорт данных. Подготовка данных

Так как в большинстве случаев в анализе данных имеют дело именно с табличным представлением данных и не вводят их вручную, а загружают данные из различных источников, то далее будем рассматривать импортированные наборы данных.

Текстовый формат табличных данных часто представлен расширением `.csv` (Comma-Separated Values, значения, разделенные запятыми).

Для считывания данных из файла `.csv` в объект `DataFrame` используется функция библиотеки `pandas` `read_csv()`. Так как это функция библиотеки, то подключив библиотеку `pandas`, сможем обращаться к ней `pd.read_csv()`. Параметры функции представлены в табл. 2.5.

```
import pandas as pd
```

Параметры метода **read_csv()**

Параметр	Описание
<code>filepath_or_buffer</code>	Имя файла, из которого должны быть прочитаны данные. Имя файла указывается относительно текущего рабочего каталога, либо абсолютный путь расположения файла, либо URL-адрес
<code>sep</code> <code>delimiter</code>	Символ разделителя полей (начальные буквы <code>separator</code> , т.е. разделитель). Альтернативное название параметра – <code>delimiter</code> . Значения в каждой строке файла разделяются этим символом
<code>header</code>	Номер строки для использования в качестве имен столбцов и начала данных. Если <code>header=0</code> или имена столбцов не заданы явно, то имена столбцов выводятся из первой строки файла, если имена столбцов передаются явно, то поведение идентично <code>header=None</code>
<code>names</code>	Список имен столбцов
<code>index_col</code>	Столбец (столбцы) для использования в качестве меток строк в <code>DataFrame</code> в виде индекса столбца (или его названия)
<code>dtype</code>	Тип данных для столбцов
<code>parse_dates</code>	В параметр передается имя столбца, значения которого необходимо определить как даты: <code>parse_dates=['name']</code>
<code>encoding</code>	Кодировка

Распространенные кодировки для файлов данных:

- “utf-8” – стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Unicode, используя переменное количество байт;
- “cp1251” – стандартная 8-битная кодировка для русских версий Microsoft Windows до 10-й версии.

Python работает по умолчанию с данными в кодировке 'utf-8'. Если при загрузке данных возвращается сообщение об ошибке: `UnicodeDecodeError: 'utf-8' codec can't decode byte 0xcd in position 0: invalid continuation byte`, то это означает, что загружаемый файл не имеет кодировку 'utf-8' и не может быть считан.

Чтобы корректно считать данные в другой кодировке, можно задать параметр `encoding` функции `read_csv()`, например `encoding='cp1251'`, или изменить кодировку файла данных.

Для изменения кодировки файла данных можно воспользоваться open source текстовым редактором Notepad++ (или другим редактором, который имеет необходимую функцию конвертации в кодировку

UTF-8). Дистрибутив Notepad++ можно скачать по ссылке <https://notepad-plus-plus.org/downloads/>.

Для выделенного текста файла изменяется кодировка Encoding> Convert to UTF-8 (рис. 2.21). Затем файл сохраняется с расширением .csv.

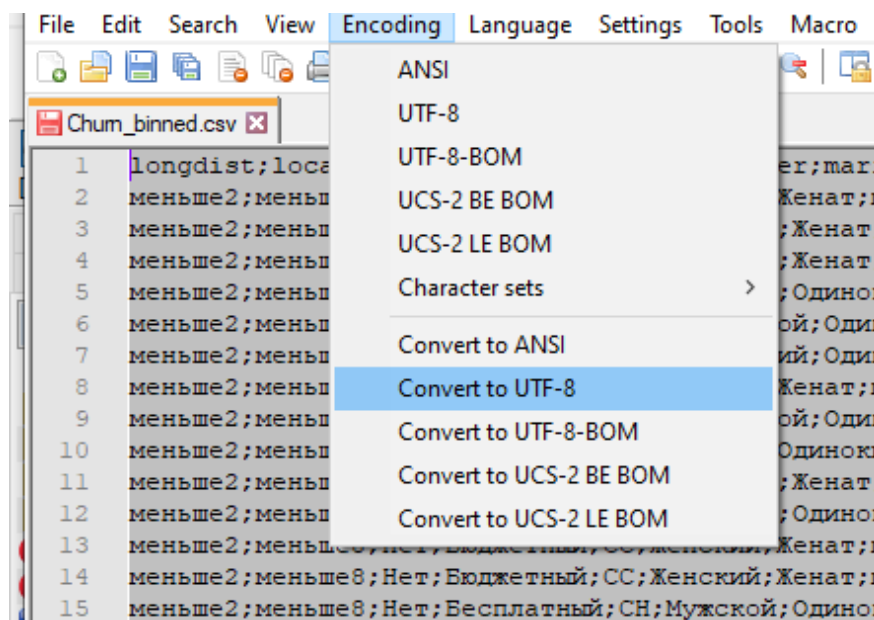


Рис. 2.21. Изменение кодировки средствами Notepad++

Используемые в примерах данные – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку, доступны на ресурсе kaggle: <https://www.kaggle.com/mirichoi0218/insurance>. Для скачивания данных необходимо авторизоваться. Для этого можно использовать ранее созданный аккаунт @gmail.com.

Описание переменных набора:

- age: возраст основного бенефициара;
- sex: пол застрахованного;
- bmi: индекс массы тела;
- children: число детей, охваченных медицинским страхованием / число иждивенцев;
- smoker: курит ли застрахованный;
- region: жилой район получателя страховых выплат в США (северо-восток, юго-восток, юго-запад, северо-запад);
- charges: индивидуальные медицинские расходы, оплачиваемые страховкой.

Используем функцию `read_csv()` для считывания набора данных из файла `insurance.csv`, предварительно размещенного в рабочей директории.

По ссылке <http://localhost:8888/tree> открывается рабочий каталог, в который был установлен дистрибутив Anaconda. На компьютере автора пособия это C:\Users\Svetlana (рис. 2.22).

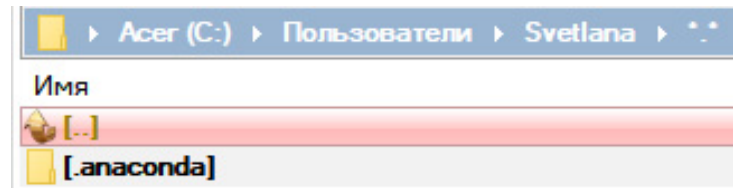


Рис. 2.22. Каталог,
в который установлен дистрибутив Anaconda

Для того чтобы блокноты с кодом и файлы с данными для выполнения лабораторных работ не смешивались с другими файлами в этой папке, рекомендуется создать отдельную директорию, например AoD (сокращение от analysis of data). Необходимо из этой директории создать новый блокнот Jupyter AoD1.ipynb и в эту же папку сохранить файл insurance.csv.

В первом блоке будем подключать необходимые библиотеки. На текущий момент их две: pandas и NumPy (рис. 2.23).

```
#Подключение библиотек
import pandas as pd
import numpy as np
```

```
In [1]: #Подключение библиотек
import pandas as pd
import numpy as np
```

Рис. 2.23. Подключение библиотек

В дальнейшем этот блок будет дополняться, так как все используемые в файле библиотеки лучше подключать в первом блоке.

Считаем файл insurance.csv и выведем содержимое объекта insurance_df на экран (рис. 2.24).

```
#Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
insurance_df
```

С помощью аргумента .dtypes выведем информацию о типе данных в наборе (рис. 2.25).

Как видим, категориальные показатели: sex, smoker, region — имеют тип object.

Для построения срезов по условию используются логические операторы (табл. 2.6).

```
In [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
insurance_df
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Рис. 2.24. Считывание данных из файла

```
In [3]: insurance_df.dtypes
```

```
Out[3]: age          int64
sex            object
bmi           float64
children       int64
smoker         object
region         object
charges       float64
dtype: object
```

Рис. 2.25. Типы данных в объекте insurance_df

Таблица 2.6

Логические операторы

Оператор	Описание
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно
&	Логическое И
	Логическое ИЛИ
!	Логическое НЕ

Извлечем из набора insurance_df все записи, в которых пол респондента female, а возраст от 30 и старше (рис. 2.26).

```
insurance_df[(insurance_df['sex']=='female') & (insurance_df['age']>=30)]
```

```
In [9]: insurance_df[(insurance_df['sex']=='female') & (insurance_df['age']>=30)]
```

```
Out[9]:
```

	age	sex	bmi	children	smoker	region	charges
5	31	female	25.74	0	no	southeast	3756.62160
6	46	female	33.44	1	no	southeast	8240.58960
7	37	female	27.74	3	no	northwest	7281.50560
9	60	female	25.84	0	no	northwest	28923.13692
11	62	female	26.29	0	yes	southeast	27808.72510
...
1323	42	female	40.37	2	yes	southeast	43896.37630
1326	42	female	32.87	0	no	northeast	7050.02130
1330	57	female	25.74	2	no	southeast	12629.16560
1332	52	female	44.70	3	no	southwest	11411.68500
1337	61	female	29.07	0	yes	northwest	29141.36030

461 rows × 7 columns

Рис. 2.26. Отбор данных по логическому условию

Исходный объект `insurance_df` не изменился, а к созданному объекту невозможно обратиться, так как на него не создана ссылка (т.е. не задана переменная, ссылающаяся на этот объект). Сборщик мусора удалит этот объект автоматически. Если он нам нужен в дальнейшей работе, для него нужно задать переменную, которая будет ссылаться на созданный объект.

```
insurance_fm = insurance_df[(insurance_df['sex']=='female') & (insurance_df['age']>=30)]
```

Альтернативный способ обращения к столбцу <имя Data Frame>.<имя столбца>.

```
insurance_df[(insurance_df.sex == 'female') & (insurance_df.age>=30)]
```

Если нужно отобрать несколько столбцов из исходного набора данных, то можно перечислить в квадратных скобках их имена через запятую (рис. 2.27).

```
insurance_df[['age', 'bmi', 'charges']]
```

```
In [12]: insurance_df[['age', 'bmi', 'charges']]
```

```
Out[12]:
```

	age	bmi	charges
0	19	27.900	16884.92400
1	18	33.770	1725.55230
2	28	33.000	4449.46200
3	33	22.705	21984.47061
4	32	28.880	3866.85520
...
1333	50	30.970	10600.54830
1334	18	31.920	2205.98080
1335	18	36.850	1629.83350
1336	21	25.800	2007.94500
1337	61	29.070	29141.36030

1338 rows × 3 columns

Рис. 2.27. Вывод нескольких столбцов исходного набора данных

Метод `.sort_values()` используется для сортировки строк таблицы по значению какого-либо столбца (табл. 2.7).

Таблица 2.7

Параметры метода `.sort_values()`

Параметр	Описание
<code>by</code>	Имя или список имен столбцов для сортировки
<code>ascending</code>	Сортировать по возрастанию (<code>ascending=True</code>) или по убыванию (<code>ascending=False</code>). Можно указать список для нескольких порядков сортировки, но количество элементов списка должно совпадать с числом столбцов выбранных для сортировки в параметре <code>by</code>
<code>inplace</code>	Если <code>inplace=False</code> , то происходит вывод результата сортировки (это значение у параметра по умолчанию), но исходный объект не изменяется. Значение <code>inplace=True</code> приводит к изменению объекта

Проведем упорядочение данных по столбцу `age` по убыванию, а по столбцу `bmi` по возрастанию, и выведем объект `insurance_df` (рис. 2.28). Как видим, объект `insurance_df` остался неизменен.

`insurance_df.sort_values(by=['age', 'bmi'], ascending=[False, True])`

```
In [52]: insurance_df.sort_values(by=['age', 'bmi'], ascending=[False, True]).head(7)
```

Out[52]:

	age	sex	bmi	children	smoker	region	charges
664	64	female	22.990	0	yes	southeast	27037.91410
1265	64	male	23.760	0	yes	southeast	26926.51440
62	64	male	24.700	1	no	northwest	30166.61817
398	64	male	25.600	2	no	southwest	14988.43200
1051	64	male	26.410	0	no	northeast	14394.55790
890	64	female	26.885	0	yes	northwest	29330.98315
378	64	female	30.115	3	no	northwest	16455.70785

```
In [53]: insurance_df.head(7)
```

Out[53]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960

Рис. 2.28. Сортировка строк по значениям столбцов без модификации исходного объекта

Создадим копию объекта `insurance_df` и проведем упорядочение элементов, но уже со значением `inplace=True`. Как видим, объект изменился (рис. 2.29). Проведем переиндексацию строк в этом объекте с помощью метода `.reset_index()`. Так как в этот метод не передавался параметр `inplace=True`, с объектом `insurance_ord` ничего не произойдет. Однако в выведенном на экран объекте с переиндексацией строк можно увидеть столбец, в котором сохранены прежние значения индексов.

```
insurance_ord=insurance_df
insurance_ord.sort_values(by=['age', 'bmi'], ascending=[False,True], inplace=True)
insurance_ord
```

```
In [54]: insurance_ord=insurance_df
insurance_ord.sort_values(by=['age', 'bmi'], ascending=[False,True], inplace=True)
insurance_ord.head(7)
```

Out[54]:

	age	sex	bmi	children	smoker	region	charges
664	64	female	22.990	0	yes	southeast	27037.91410
1265	64	male	23.760	0	yes	southeast	26926.51440
62	64	male	24.700	1	no	northwest	30166.61817
398	64	male	25.600	2	no	southwest	14988.43200
1051	64	male	26.410	0	no	northeast	14394.55790
890	64	female	26.885	0	yes	northwest	29330.98315
378	64	female	30.115	3	no	northwest	16455.70785

```
In [55]: insurance_ord.reset_index().head(7)
```

Out[55]:

	index	age	sex	bmi	children	smoker	region	charges
0	664	64	female	22.990	0	yes	southeast	27037.91410
1	1265	64	male	23.760	0	yes	southeast	26926.51440
2	62	64	male	24.700	1	no	northwest	30166.61817
3	398	64	male	25.600	2	no	southwest	14988.43200
4	1051	64	male	26.410	0	no	northeast	14394.55790
5	890	64	female	26.885	0	yes	northwest	29330.98315

Рис. 2.29. Переиндексация строк (с сохранение старых индексов в новом столбце `index`) без модификации исходного объекта

Проведем упорядочение индексов строк `insurance_ord` с помощью метода `.sort_index()` (рис. 2.30). И так как значение по умолчанию параметра `inplace=False`, то с самым объектом `insurance_ord` ничего не произойдет.

Проведем в самом объекте `insurance_ord` переиндексацию строк, удалив столбец с прежними значениями индексов, для чего передадим, кроме параметра `inplace=True`, еще и параметр `drop=True` в метод `reset_index()`. Результат выполнения кода на рис. 2.31.

```
insurance_ord.reset_index(inplace=True, drop=True)
insurance_ord.head(7)
```

```
In [59]: insurance_ord.sort_index().head(7)
```

```
Out[59]:
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960

```
In [60]: insurance_ord
```

```
Out[60]:
```

	age	sex	bmi	children	smoker	region	charges
664	64	female	22.99	0	yes	southeast	27037.91410
1265	64	male	23.76	0	yes	southeast	26926.51440
62	64	male	24.70	1	no	northwest	30166.61817
398	64	male	25.60	2	no	southwest	14988.43200
1051	64	male	26.41	0	no	northeast	14394.55790

Рис. 2.30. Упорядочение по возрастанию индексов строк без модификации исходного объекта

```
In [61]: insurance_ord.reset_index(inplace=True, drop=True)  
insurance_ord.head(7)
```

```
Out[61]:
```

	age	sex	bmi	children	smoker	region	charges
0	64	female	22.990	0	yes	southeast	27037.91410
1	64	male	23.760	0	yes	southeast	26926.51440
2	64	male	24.700	1	no	northwest	30166.61817
3	64	male	25.600	2	no	southwest	14988.43200
4	64	male	26.410	0	no	northeast	14394.55790
5	64	female	26.885	0	yes	northwest	29330.98315
6	64	female	30.115	3	no	northwest	16455.70785

Рис. 2.31. Модификация объекта – переиндексация строк

Для удаления строк и столбцов в DataFrame используется метод `.drop()`, параметры которого представлены в табл. 2.8.

Удалим из объекта для экспериментов два столбца: пол и регион (рис. 2.32).

```
insurance_ord.drop(['sex', 'region'], axis=1, inplace=True)  
insurance_ord
```


Параметры метода `.drop()`

Параметр	Описание
<code>labels</code>	Метки индексов строк или столбцов, которые необходимо удалить
<code>axis</code>	0 – удаляются строки, 1 – удаляются столбцы
<code>inplace</code>	Если <code>inplace=False</code> , то происходит вывод результата удаления (это значение у параметра по умолчанию), но исходный объект не изменяется. Значение <code>inplace=True</code> приводит к изменению объекта

```
In [67]: insurance_ord.drop(['sex', 'region'], axis=1, inplace=True)
insurance_ord.head(7)
```

Out[67]:

	age	bmi	children	smoker	charges
0	64	22.990	0	yes	27037.91410
1	64	24.700	1	no	30166.61817
2	64	25.600	2	no	14988.43200
3	64	26.410	0	no	14394.55790
4	64	30.115	3	no	16455.70785
5	64	31.300	2	yes	47291.05500
6	64	31.825	2	no	16069.08475

Рис. 2.32. Удаление столбцов с модификацией объекта

Для сохранения DataFrame в файл `.CSV` используется метод `.to_csv()`.

```
insurance_ord.to_csv('tmp.csv')
```

Работа в блокнотах Google Colab будет отличаться только одним моментом – загрузкой файла на сервер.

Для загрузки файла с локального диска в облачный Google Colab воспользуемся кодом:

```
from google.colab import files
uploaded = files.upload()
```

При выполнении кода появится виджет с предложением выбрать файлы на компьютере. После щелчка на кнопке откроется проводник для выбора файлов с диска (рис. 2.33). Если файл доступен по URL-адресу, то этот блок нужно опустить и вместо пути к файлу на компьютере набрать этот адрес в параметрах метода `read_csv()`.

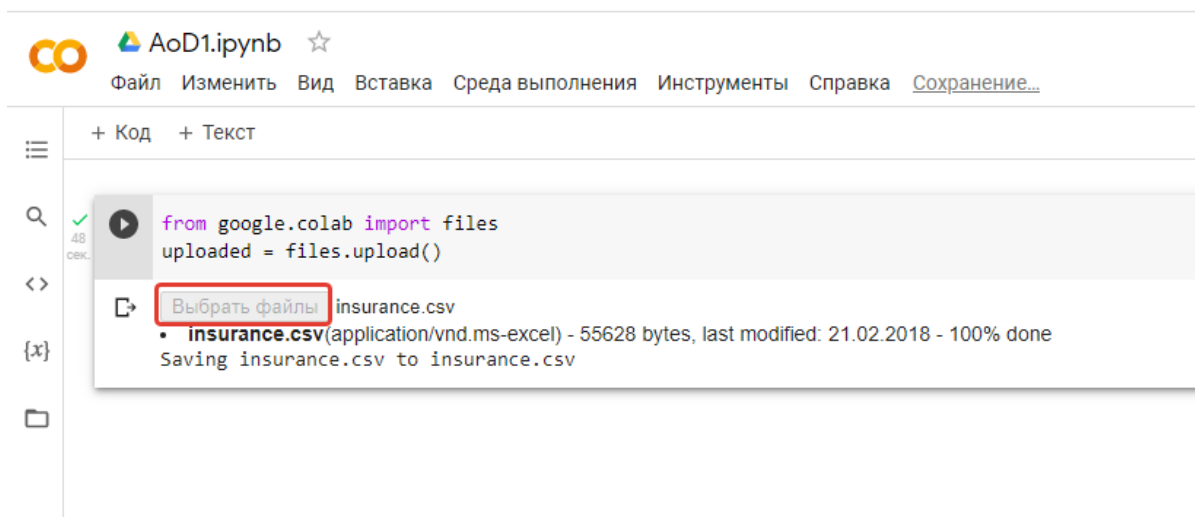


Рис. 2.33. Загрузка файла в облако

В остальном исполнение инструкций будет аналогичным, и далее все инструкции в Jupyter можно использовать в Google Colab (рис. 2.34).

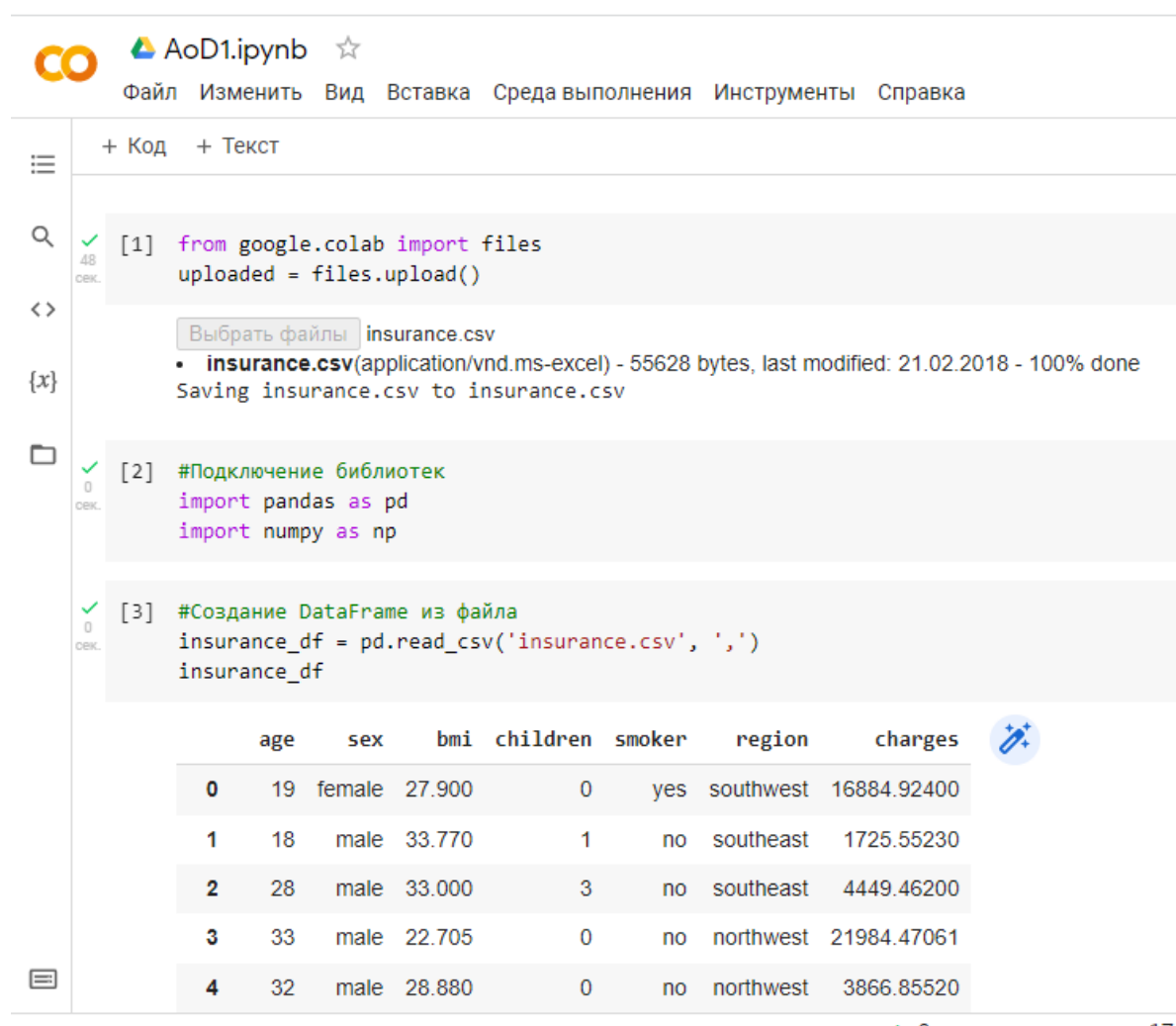


Рис. 2.34. Блоки с кодом и их выполнение в Google Colab

3. Описательный анализ данных. Библиотеки pandas и NumPy

Базовые функции для расчета параметров описательной статистики приведены в табл. 3.1.

Таблица 3.1

Методы для расчета параметров описательной статистики

Метод	Описание
count	Количество не-NA объектов
sum	Сумма
mean	Среднее значение
mad	Среднее абсолютное отклонение
median	Медиана
min	Минимум
max	Максимум
mode	Мода
std	Стандартное отклонение
var	Несмещенная дисперсия
sem	Стандартная ошибка среднего
skew	Скошенность (момент 3-го порядка)
kurt	Экссесс (момент 4-го порядка)
quantile	Квантиль (%)
cumsum	Кумулятивная сумма
cumprod	Кумулятивное произведение
cummax	Кумулятивный максимум
cummin	Кумулятивный минимум

Среди данных могут быть пропуски, которые не позволят рассчитать статистические параметры для столбцов количественных переменных с пропусками без специальных настроек.

Создадим новый блокнот Jupyter. Подключим библиотеки и считаем файл данных в два разных объекта `insurance_df` и `df_nan` (рис. 3.1). То, как мы будем вносить пропуски в `DataFrame`, будет влиять на изменение всех копий исходного объекта, поэтому необходимо создать два разных объекта `DataFrame`.

```
In [1]: #Подключение библиотек
import pandas as pd
import numpy as np

In [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
df_nan = pd.read_csv('insurance.csv', ',')
```

Рис. 3.1. Подключение библиотек и создание объектов `DataFrame`

С помощью метода `.info()` получим сводку по типу данных и наличию пропусков в отдельных столбцах набора данных `df_nan` (рис. 3.2).

`df_nan.info()`

```
In [3]: df_nan.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
age          1338 non-null int64
sex          1338 non-null object
bmi          1338 non-null float64
children     1338 non-null int64
smoker       1338 non-null object
region       1338 non-null object
charges      1338 non-null float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Рис. 3.2. Сводка для набора данных `df_nan`

С помощью свойства `.iat[]` получим доступ к отдельным значениям в `DataFrame` и изменим их значения (в квадратных скобках указывается пара значений – номер строки и номер столбца нужного элемента, нумерация в Python начинается с 0).

```
df_nan.iat[0, 1]=None
df_nan.iat[3, 6]=None
df_nan.iat[2, 2]=None
df_nan.head(7)
```

```
In [4]: df_nan.iat[0, 1]=None
df_nan.iat[3, 6]=None
df_nan.iat[2, 2]=None
df_nan.head(7)
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges
0	19	None	27.900	0	yes	southwest	16884.9240
1	18	male	33.770	1	no	southeast	1725.5523
2	28	male	NaN	3	no	southeast	4449.4620
3	33	male	22.705	0	no	northwest	NaN
4	32	male	28.880	0	no	northwest	3866.8552
5	31	female	25.740	0	no	southeast	3756.6216
6	46	female	33.440	1	no	southeast	8240.5896

Рис. 3.3. Замена отдельных значений `df_nan` пропусками

Теперь запросим информацию о наличии пропусков и увидим, что число наблюдений без пропусков изменилось в трех столбцах (рис. 3.4).

```
df_nan.info()
```

```
In [5]: df_nan.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
age          1338 non-null int64
sex          1337 non-null object
bmi          1337 non-null float64
children     1338 non-null int64
smoker       1338 non-null object
region       1338 non-null object
charges      1337 non-null float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Рис. 3.4. Сводка для набора данных df_nan

Метод .isnull() позволяет получить объект, в котором на соответствующих элементам исходного DataFrame стоят значения False/True, и каждое значение True соответствует пропуску в исходном наборе данных (рис. 3.5).

```
df_nan.isnull()
```

```
In [12]: df_nan.isnull()
```

```
Out[12]:
```

	age	sex	bmi	children	smoker	region	charges
0	False	True	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	True	False	False	False	False
3	False	False	False	False	False	False	True
4	False	False	False	False	False	False	False
...

Рис. 3.5. Определение позиций пропусков в наборе данных

Общее количество пропусков можно получить, дважды просуммировав полученный DataFrame, в начале по столбцам, а потом по строкам (рис. 3.6).

```
df_nan.isnull().sum()
```

```
df_nan.isnull().sum().sum()
```

```
In [13]: df_nan.isnull().sum()
```

```
Out[13]: age          0
sex          1
bmi          1
children     0
smoker       0
region       0
charges      1
dtype: int64
```

```
In [14]: df_nan.isnull().sum().sum()
```

```
Out[14]: 3
```

Рис. 3.6. Число пропусков в каждом столбце и общее число пропусков в наборе данных

Также можно использовать метод `.count()`, который подсчитывает количество непропущенных значений в каждом столбце (рис. 3.7), однако ранее эту информацию по набору данных уже возвратил метод `.info()`. Сводка, выведенная методом `.info()`, была более подробной, так как было известно количество наблюдений/записей.

`df_nan.count()`

```
In [15]: df_nan.count()
```

```
Out[15]: age          1338
sex          1337
bmi          1337
children     1338
smoker       1338
region       1338
charges      1337
dtype: int64
```

Рис. 3.7. Подсчет числа значений без пропусков в каждом столбце набора данных

Самый простой способ работы с пропусками – это удаление записей их содержащих. Метод `.dropna()` удаляет такие строки (если передать в него параметр `axis=1`, то будут удалены столбцы, в которых есть пропуски, а параметр `inplace=True` позволяет модифицировать исходный объект, иначе ссылку на созданную модифицированную копию нужно присвоить какой-либо переменной). Выведем на экран результат применения метода `.dropna()` к `df_nan` без модификации (рис. 3.8).

`df_nan.dropna()`

```
In [16]: df_nan.dropna().head(7)
```

```
Out[16]:
```

	age	sex	bmi	children	smoker	region	charges
1	18	male	33.77	1	no	southeast	1725.55230
4	32	male	28.88	0	no	northwest	3866.85520
5	31	female	25.74	0	no	southeast	3756.62160
6	46	female	33.44	1	no	southeast	8240.58960
7	37	female	27.74	3	no	northwest	7281.50560
8	37	male	29.83	2	no	northeast	6406.41070
9	60	female	25.84	0	no	northwest	28923.13692

Рис. 3.8. Удаление строк с пропусками из набора данных

Были удалены строки с индексами: 0, 2 и 3, и если бы результат сохранился в какую-либо переменную, то с помощью метода `.reset_index()` с параметром `inplace=True` можно было бы провести переиндексацию строк.

Воспользуемся методом `.fillna()` для заполнения пропущенных значений (`fill` – заполнить, `na` – not available, не доступно). Параметры метода `.fillna()` представлены в табл. 3.2.

Таблица 3.2

Параметры метода `.fillna()`

Параметр	Описание
<code>value</code>	Значение, используемое для заполнения пропусков
<code>method</code>	'backfill' / 'bfill' – использование следующего за пропусками действительного наблюдения для заполнения пропуска, 'pad' / 'ffill' – распространение последнего действительного значения перед пропусками для заполнения пропусков
<code>axis</code>	0 – удаляются строки, 1 – удаляются столбцы
<code>inplace</code>	Если <code>inplace=False</code> , то происходит вывод результата заполнения пропусков (это значение у параметра по умолчанию), но исходный объект не изменяется. Значение <code>inplace=True</code> приводит к изменению объекта

Так как параметр `inplace=True` в дальнейшем в метод `.fillna()` не передается, то результат замены пропусков только выводится на экран, а исходный объект не меняется.

Заполним пропуски средним значением (рис. 3.9). Как видим, для категориального показателя `sex` это не сработало. В случае замены категориального показателя на столбец числовых меток, когда 0 – 'female', 1 – 'male', ситуация была бы еще более сомнительной, так как среднее значение в этом случае было бы долей мужчин в наборе данных и было бы равно 0,676 (в случае дискретного показателя такое числовое значение не годится для заполнения пропуска).

```
df_nan.fillna(df_nan.mean())
```

```
In [18]: df_nan.fillna(df_nan.mean())
```

```
Out[18]:
```

	age	sex	bmi	children	smoker	region	charges
0	19	None	27.900000	0	yes	southwest	16884.924000
1	18	male	33.770000	1	no	southeast	1725.552300
2	28	male	30.661649	3	no	southeast	4449.462000
3	33	male	22.705000	0	no	northwest	13263.904652
4	32	male	28.880000	0	no	northwest	3866.855200

Рис. 3.9. Заполнение пропусков средним значением в соответствующем столбце

Используем для заполнения моду (наиболее частотное значение показателя) (рис. 3.10). Ситуация поменялась на противоположную: теперь только пропуск в столбце `sex` заполнен, а для непрерывных количественных показателей `bmi` и `charges` мода не определена.

```
df_nan.fillna(df_nan.mode())
```

```
In [21]: df_nan.fillna(df_nan.mode())
```

```
Out[21]:
```

	age	sex	bmi	children	smoker	region	charges
0	19	male	27.900	0	yes	southwest	16884.9240
1	18	male	33.770	1	no	southeast	1725.5523
2	28	male	NaN	3	no	southeast	4449.4620
3	33	male	22.705	0	no	northwest	NaN
4	32	male	28.880	0	no	northwest	3866.8552

Рис. 3.10. Заполнение пропусков наиболее часто встречающимся значением в соответствующем столбце

Однако если последовательно применить этот метод с двумя разными статистическими показателями для заполнения пропусков, то все пропуски исчезнут (рис. 3.11).

```
df_nan.fillna(df_nan.mean()).fillna(df_nan.mode())
```

```
In [22]: df_nan.fillna(df_nan.mean()).fillna(df_nan.mode())
```

Out[22]:

	age	sex	bmi	children	smoker	region	charges
0	19	male	27.900000	0	yes	southwest	16884.924000
1	18	male	33.770000	1	no	southeast	1725.552300
2	28	male	30.661649	3	no	southeast	4449.462000
3	33	male	22.705000	0	no	northwest	13263.904652
4	32	male	28.880000	0	no	northwest	3866.855200

Рис. 3.11. Заполнение пропусков

Теперь заполним пропуски значением перед пропуском (рис. 3.12). Для столбца `sex` пропуск не ликвидирован, так как этот пропуск в строке с индексом 0, и для него просто нет предшествующего непропущенного значения.

```
df_nan.fillna(method = 'ffill')
```

```
In [23]: df_nan.fillna(method = 'ffill')
```

Out[23]:

	age	sex	bmi	children	smoker	region	charges
0	19	None	27.900	0	yes	southwest	16884.9240
1	18	male	33.770	1	no	southeast	1725.5523
2	28	male	33.770	3	no	southeast	4449.4620
3	33	male	22.705	0	no	northwest	4449.4620
4	32	male	28.880	0	no	northwest	3866.8552

Рис. 3.12. Заполнение пропусков предшествующим значением в столбце

Теперь заполним пропуски значением, следующим за пропуском (рис. 3.13).

```
df_nan.fillna(method = 'bfill')
```

Также можно использовать метод `.fillna()` для заполнения пропусков в каком-либо столбце конкретным значением (рис. 3.14). При использовании параметра `inplace=True` изменения модифицировали набор данных и заполнили пропуск в исходном объекте.


```
df_nan.sex.fillna('male', inplace=True)
df_nan
```

```
In [24]: df_nan.fillna(method = 'bfill')
```

Out[24]:

	age	sex	bmi	children	smoker	region	charges
0	19	male	27.900	0	yes	southwest	16884.9240
1	18	male	33.770	1	no	southeast	1725.5523
2	28	male	22.705	3	no	southeast	4449.4620
3	33	male	22.705	0	no	northwest	3866.8552
4	32	male	28.880	0	no	northwest	3866.8552

Рис. 3.13. Заполнение пропусков последующим значением в столбце

```
In [25]: df_nan.sex.fillna('male', inplace=True)
df_nan.head(7)
```

Out[25]:

	age	sex	bmi	children	smoker	region	charges
0	19	male	27.900	0	yes	southwest	16884.9240
1	18	male	33.770	1	no	southeast	1725.5523
2	28	male	NaN	3	no	southeast	4449.4620
3	33	male	22.705	0	no	northwest	NaN
4	32	male	28.880	0	no	northwest	3866.8552
5	31	female	25.740	0	no	southeast	3756.6216
6	46	female	33.440	1	no	southeast	8240.5896

Рис. 3.14. Заполнение пропусков

Метод `.describe()` выводит сводку по параметрам описательной статистики для всех переменных набора. Для количественных переменных будут рассчитаны максимальное и минимальное значения, среднее и три квартиля (второй квартиль – это медиана), для категориальных данных будет рассчитана абсолютная частота каждого уровня (сколько раз в наблюдениях встречается каждое значение признака).

Выведем сводку описательных статистик для набора `insurance_df` (рис. 3.15). Как видим, сводка включает только количественные показатели.

```
insurance_df.describe()
```

```
In [26]: insurance_df.describe()
```

```
Out[26]:
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Рис. 3.15. Описательные статистики для количественных переменных

Передадим в метод `.describe()` параметр `include='all'`. Теперь выведены статистики по всем переменным набора, однако для категориальных признаков рассчитываются иные статистические показатели (рис. 3.16).

```
insurance_df.describe(include='all')
```

```
In [27]: insurance_df.describe(include='all')
```

```
Out[27]:
```

	age	sex	bmi	children	smoker	region	charges
count	1338.000000	1338	1338.000000	1338.000000	1338	1338	1338.000000
unique	NaN	2	NaN	NaN	2	4	NaN
top	NaN	male	NaN	NaN	no	southeast	NaN
freq	NaN	676	NaN	NaN	1064	364	NaN
mean	39.207025	NaN	30.663397	1.094918	NaN	NaN	13270.422265
std	14.049960	NaN	6.098187	1.205493	NaN	NaN	12110.011237
min	18.000000	NaN	15.960000	0.000000	NaN	NaN	1121.873900
25%	27.000000	NaN	26.296250	0.000000	NaN	NaN	4740.287150
50%	39.000000	NaN	30.400000	1.000000	NaN	NaN	9382.033000
75%	51.000000	NaN	34.693750	2.000000	NaN	NaN	16639.912515
max	64.000000	NaN	53.130000	5.000000	NaN	NaN	63770.428010

Рис. 3.16. Описательные статистики для всех переменных

Можно вывести для категориальных данных описательные статистики отдельной таблицей (рис. 3.17).

```
insurance_df.describe(include=[object])
```

```
In [28]: insurance_df.describe(include=[object])
```

```
Out[28]:
```

	sex	smoker	region
count	1338	1338	1338
unique	2	2	4
top	male	no	southeast
freq	676	1064	364

Рис. 3.17. Описательные статистики для категориальных показателей

Иногда параметры описательной статистики нужно рассчитать по группам. Например, вычислить средние значения возраста (age), индекса массы тела (bmi) и медицинских расходов (charges) отдельно для курящих и для некурящих застрахованных. Можно применить метод `mean()` к сгруппированным с помощью метода `.groupby()` данным (рис. 3.18).

```
insurance_df.groupby('smoker').mean()
```

```
In [32]: insurance_df.groupby('smoker').mean()
```

```
Out[32]:
```

	age	bmi	children	charges
smoker				
no	39.385338	30.651795	1.090226	8434.268298
yes	38.514599	30.708449	1.113139	32050.231832

Рис. 3.18. Расчет среднего значения для срезов данных

Как видим, средний возраст для курящих застрахованных и некурящих практически одинаков, а вот расходы отличаются значительно.

Можно вывести сводку по описательным статистикам для сгруппированных данных (рис. 3.19).

```
insurance_df.groupby('smoker').describe()
```

```
In [29]: insurance_df.groupby('smoker').describe()
```

```
Out [29]:
```

	age							bmi							children		charges									
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max		
smoker																										
no	1064.0	39.385338	14.083410	18.0	26.75	40.0	52.0	64.0	1064.0	30.651795	...	2.0	5.0	1064.0	8434.268298	5993.781819	1121.8739	3986.4387								
yes	274.0	38.514599	13.923186	18.0	27.00	38.0	49.0	64.0	274.0	30.708449	...	2.0	5.0	274.0	32050.231832	11541.547176	12829.4551	20826.2442								

2 rows x 32 columns

Рис. 3.19. Расчет описательных статистик для сгруппированных данных

Также можно предварительно удалить из рассмотрения отдельные столбцы и рассчитать сводку для оставшихся в рассмотрении количественных показателей (рис. 3.20).

```
insurance_df.drop(['bmi', 'children'],axis=1).groupby('smoker').describe()
```

```
In [38]: insurance_df.drop(['bmi', 'children'],axis=1).groupby('smoker').describe()
```

```
Out [38]:
```

	age							charges								
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
smoker																
no	1064.0	39.385338	14.083410	18.0	26.75	40.0	52.0	64.0	1064.0	8434.268298	5993.781819	1121.8739	3986.438700	7345.40530	11362.887050	369
yes	274.0	38.514599	13.923186	18.0	27.00	38.0	49.0	64.0	274.0	32050.231832	11541.547176	12829.4551	20826.244213	34456.34845	41019.207275	637

Рис. 3.20. Расчет описательных статистик для сгруппированных данных

4. Разведочный анализ данных. Библиотеки Matplotlib и Seaborn

Разведочный анализ данных основан на построении визуализаций.

4.1. Библиотека Matplotlib

Начнем с возможностей визуализации данных библиотеки Matplotlib. Matplotlib изначально предназначался для работы с массивами NumPy. Применение функций Matplotlib для визуализации DataFrame, наиболее распространенного представления для набора данных, во многих случаях требует дополнительных преобразований данных.

В первом блоке нового блокнота Jupyter подключим все необходимые библиотеки. Кроме библиотек для работы с данными pandas и NumPy, нам понадобится и модуль библиотеки Matplotlib: pyplot. Модуль matplotlib.pyplot представляет собой менеджер графического интерфейса фигур, подобный MATLAB. Также для вывода графиков после блоков кода в блокноте Jupyter необходимо выполнить специальную *magic* команду после импорта модуля matplotlib.pyplot.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Магические команды – это улучшения, добавляемые поверх обычного кода на Python, их предоставляет ядро IPython, которое использует Jupyter Notebook. Эти команды обычно начинаются с символа %.

Магические команды были добавлены для решения распространенных проблем, с которыми сталкиваются пользователи.

При выполнении кода, связанного с графикой, часто выдаются предупреждения, которые не мешают выполнению кода, и в случае базовых визуализаций их можно просто подавить (т.е. не выводить на экран).

Для этого используется модуль warnings.

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Подключение библиотек и создание объекта DataFrame (рис. 4.1).


```

Ввод [1]: #Подключение библиотек
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

Ввод [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')

```

Рис. 4.1. Подключение библиотек

Основным элементом изображения, которое строит `pyplot`, является фигура, на нее накладывается одно или более поле с графиками, оси координат, текстовые надписи и т.д. Некоторые функции для построения визуализаций приведены в табл. 4.1.

Таблица 4.1

Функции для построения графиков

Функция	Описание
<code>plt.scatter()</code>	Диаграмма рассеяния (точечная)
<code>plt.bar()</code>	Столбчатая диаграмма
<code>plt.boxplot()</code>	График «ящик с усами»

Для размещения на графике дополнительных элементов используются функции, часть которых представлена в табл. 4.2.

Таблица 4.2

Дополнительные элементы графика

Функция	Описание
<code>plt.figure()</code>	Определяет рабочую область графика и ее основные настройки. Например, параметр <code>figsize</code> – определяет размер области графика
<code>plt.title()</code>	Заголовок графика – название визуализации. Параметр <code>fontsize</code> определяет размер шрифта
<code>plt.xlabel()</code>	Название оси абсцисс. Параметр <code>fontsize</code> определяет размер шрифта
<code>plt.ylabel()</code>	Название оси ординат. Параметр <code>fontsize</code> определяет размер шрифта

Построим визуализации для набора данных `insurance_df` с использованием библиотеки `Matplotlib`. Более сложные и детализированные визуализации построим с использованием библиотеки `Seaborn` чуть позднее.

Точечный график зависимости медицинских расходов от индекса массы тела застрахованного (рис. 4.2).

```
plt.figure(figsize=(16, 8))# размер области графика
plt.scatter(insurance_df.bmi, insurance_df.charges)
plt.title('Медицинские расходы', fontsize=20)# заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

```
Ввод [3]: plt.figure(figsize=(16, 8))# размер области графика
plt.scatter(insurance_df.bmi, insurance_df.charges)
plt.title('Медицинские расходы', fontsize=20)# заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

```
Out[3]: Text(0, 0.5, 'Расходы')
```



Рис. 4.2. Точечный график

Столбчатую диаграмму построим для категориального показателя `smoker` (рис. 4.3). Необходимо подсчитать число наблюдений с каждым значением фактора, преобразовать полученный объект в `DataFrame`, переименовать столбец частот встречаемости каждого уровня показателя в выборке, преобразовать индексы строк (уровней показателя `smoker`) в столбец созданного `DataFrame`. И только после этого можно строить столбчатую диаграмму. Конечно, этот путь к цели не единственный, можно предложить и иные варианты достижения подобного результата. Однако в любом случае невозможно непосредственно передать в функцию построения столбчатой диаграммы столбец наблюдений категориальной переменной и сразу получить нужный результат.

```
ins_smok_quant = insurance_df.smoker.value_counts()
ins_smok_quant = ins_smok_quant.to_frame()
ins_smok_quant = ins_smok_quant.rename(columns={'smoker': 'quantity'})
ins_smok_quant['smoker'] = ins_smok_quant.index
```

```
plt.bar(ins_smok_quant.smoker.to_numpy(),
ins_smok_quant.quantity.to_numpy())
plt.title('Распространенность вредной привычки', fontsize=16)# заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

```
Ввод [4]: ins_smok_quant = insurance_df.smoker.value_counts()
ins_smok_quant = ins_smok_quant.to_frame()
ins_smok_quant = ins_smok_quant.rename(columns={'smoker': 'quantity'})
ins_smok_quant['smoker'] = ins_smok_quant.index
```

```
Ввод [5]: plt.bar(ins_smok_quant.smoker.to_numpy(), ins_smok_quant.quantity.to_numpy())
plt.title('Распространенность вредной привычки', fontsize=16)# заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

```
Out[5]: Text(0, 0.5, 'Количество застрахованных')
```

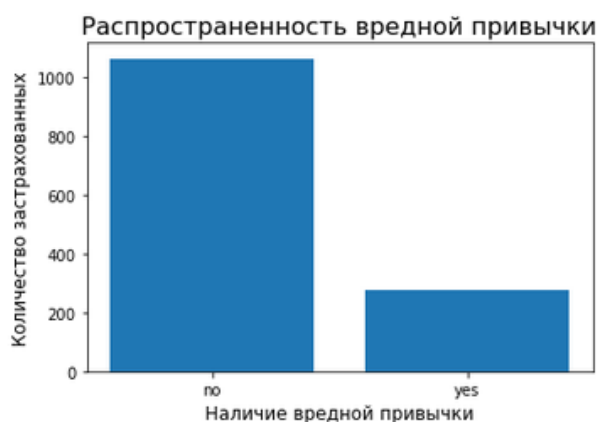


Рис. 4.3. Столбчатая диаграмма

Прделаем те же манипуляции с данными для построения столбчатой диаграммы для категориальной переменной region (рис. 4.4).

```
ins_reg_quant = insurance_df.region.value_counts()
ins_reg_quant = ins_reg_quant.to_frame()
ins_reg_quant = ins_reg_quant.rename(columns={'region': 'quantity'})
ins_reg_quant['region'] = ins_reg_quant.index
plt.bar(ins_reg_quant.region.to_numpy(), ins_reg_quant.quantity.to_numpy())
plt.title('Где проживает застрахованный', fontsize=16)# заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

Построим график «ящик с усами» для количественной переменной bmi (рис. 4.5). Для построения такого графика с группировкой наблюдений по району проживания пришлось бы писать дополнительный код.

```
plt.boxplot(insurance_df.bmi)
plt.title('Индекс массы тела', fontsize=20)# заголовок
```



```

Ввод [6]: ins_reg_quant = insurance_df.region.value_counts()
ins_reg_quant = ins_reg_quant.to_frame()
ins_reg_quant = ins_reg_quant.rename(columns={'region': 'quantity'})
ins_reg_quant['region'] = ins_reg_quant.index

Ввод [7]: plt.bar(ins_reg_quant.region.to_numpy(), ins_reg_quant.quantity.to_numpy())
plt.title('Где проживает застрахованный', fontsize=16) # заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат

```

```

Out[7]: Text(0, 0.5, 'Количество застрахованных')

```

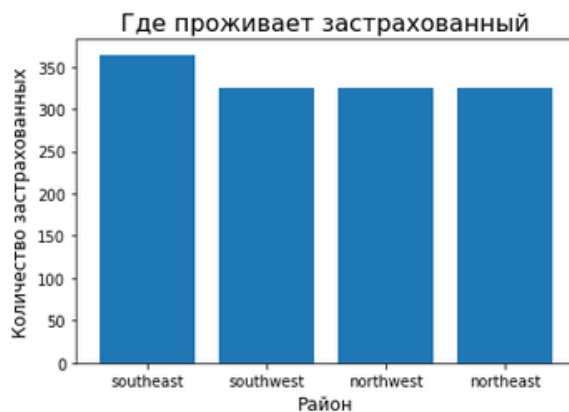


Рис. 4.4. Столбчатая диаграмма

```

Ввод [8]: plt.boxplot(insurance_df.bmi)
plt.title('Индекс массы тела', fontsize=20) # заголовок

```

```

Out[8]: Text(0.5, 1.0, 'Индекс массы тела')

```

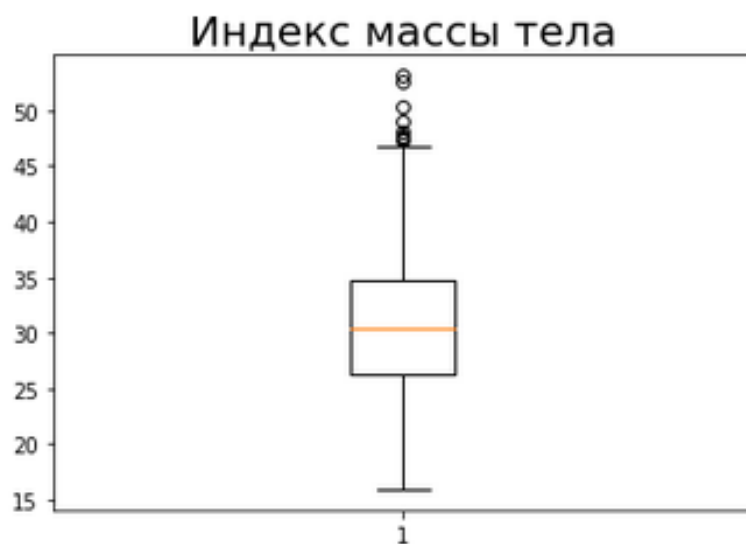


Рис. 4.5. «Ящик с усами»

4.2. Библиотека Seaborn

Seaborn основывается на Matplotlib и тесно взаимодействует со структурами данных pandas.

При импорте seaborn ей задается псевдоним `sns`, он является общепринятым среди пользователей этой библиотеки.

Библиотека имеет высокоуровневый графический интерфейс и позволяет построить большинство графиков одной строчкой кода. Автор библиотеки seaborn – Michael Waskom, PhD, сотрудник Center for Neural Research (Нью-Йорк), выпускник Стэнфорда. Библиотека была названа в честь Сэмюэла Нормана Сиборна (S.N.S. – именно поэтому `import seaborn as sns`), героя сериала The West Wing.

В визуализациях библиотеки seaborn оси автоматически подписываются именами столбцов, по данным которых строятся.

Так как библиотека seaborn построена на основе Matplotlib, с отдельными функциями которой уже познакомились в пункте 4.1, то для подписи осей и создания заголовков в графиках новой библиотеки будем использовать функции Matplotlib из табл. 4.2.

Для того чтобы это было возможно, подключим также модуль `matplotlib.pyplot`.

Внешний вид графика задается с помощью функции `sns.set_style()`

Параметр `style` этой функции может принимать значения: 'darkgrid', 'whitegrid', 'dark', 'white', 'ticks', или может быть задан набор пользовательских настроек стиля с помощью словаря (dict).

Для управления масштабом в библиотеке есть функция `sns.set_context()`

Параметр `context` этой функции может принимать значения 'paper', 'notebook', 'talk', 'poster'. Параметр `font_scale` определяет коэффициент масштабирования для размера шрифта (по умолчанию 1, если выбрать 0.5 – шрифт уменьшится вдвое).

Будем менять эти настройки отображения, чтобы продемонстрировать их визуальный эффект.

Параметры для функции `relplot()` представлены в табл. 4.3.

Таблица 4.3

Параметры функции `relplot()`

Параметр	Описание
<code>x, y</code>	Имена переменных (имена столбцов) из набора <code>data</code>
<code>data</code>	Набор данных в формате <code>pandas.DataFrame</code> , в котором столбцы – это имена переменных, строки – значения

Параметр	Описание
hue	Имя категориальной переменной из набора data , которая будет использоваться для цветового разделения данных
size	Имя переменной (дискретной, с ограниченным числом значений) из набора data , которая будет использоваться для разделения данных по размеру
style	Имя переменной из набора data , которая будет использоваться для разделения данных по стилю (виду маркера и стилю начертания линии)

Подключим библиотеки и создадим объект DataFrame (рис. 4.6).

```
#Подключение библиотек
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
#Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
```

```
Ввод [1]: #Подключение библиотек
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

Ввод [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
```

Рис. 4.6. Подключение библиотек

Построим точечный график зависимости медицинских расходов от индекса массы тела застрахованного (рис. 4.7). Добавим подписи осей и заголовков, а также зададим параметр размера маркеров (с помощью столбца **smoker**) в функции **relplot()**.

```
sns.set_context('paper')
sns.relplot(x="bmi", y="charges", size='smoker', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

```
Ввод [3]: sns.set_context('paper')
sns.relplot(x="bmi", y="charges", size='smoker', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

```
Out[3]: Text(28.2930430277778, 0.5, 'Расходы')
```

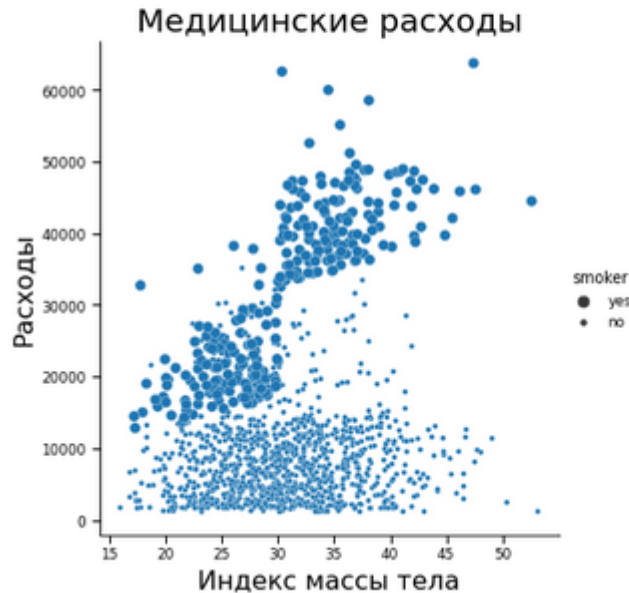


Рис. 4.7. Построение точечного графика

Изменим масштаб, а также зададим параметр для разделения точек по цвету (sex) и по размеру (smoker) для точечного графика зависимости медицинских расходов от индекса массы тела застрахованного (рис. 4.8).

```
sns.set_context('notebook')
sns.relplot(x="bmi", y="charges", hue='sex', size='smoker', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

Изменим масштаб и размер шрифта (он будет применен к легенде), а также зададим параметр для разделения точек по цвету (smoker) и по стилю (sex) для точечного графика зависимости медицинских расходов от индекса массы тела застрахованного (рис. 4.9).

```
sns.set_context('talk', font_scale=0.5)
sns.scatterplot(x="bmi", y="charges", hue='smoker', style='sex', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат
```

```

Ввод [4]: sns.set_context('notebook')
sns.relplot(x="bmi", y="charges", hue='sex', size='smoker', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат

Out[4]: Text(42.08782916666668, 0.5, 'Расходы')

```

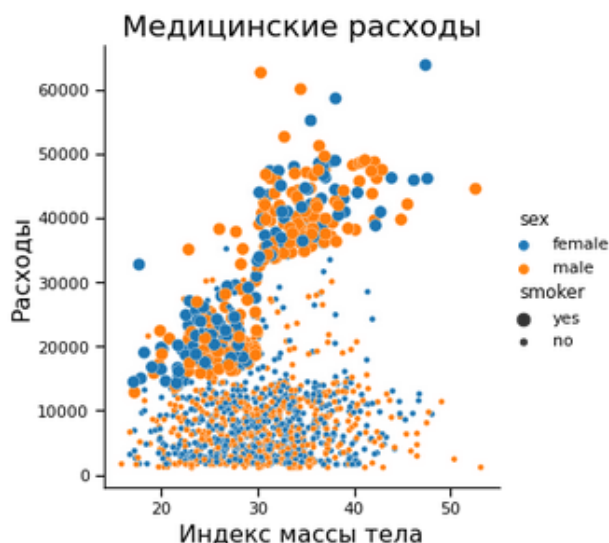


Рис. 4.8. Построение точечного графика

```

Ввод [5]: sns.set_context('talk', font_scale=0.5)
sns.scatterplot(x="bmi", y="charges", hue='smoker', style='sex', data=insurance_df)
plt.title('Медицинские расходы', fontsize=20) # заголовок
plt.xlabel('Индекс массы тела', fontsize=16) # ось абсцисс
plt.ylabel('Расходы', fontsize=16) # ось ординат

Out[5]: Text(0, 0.5, 'Расходы')

```

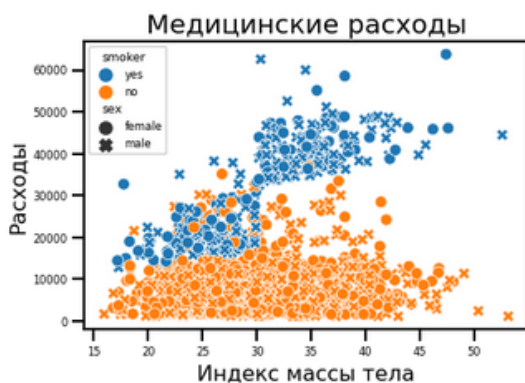


Рис. 4.9. Построение точечного графика

Для визуализации распределений количественной переменной с группировкой по категориальному показателю можно использовать функцию `stripplot()`, где по оси абсцисс (x) – значения категориального показателя, а по оси ординат – распределения количественной переменной (рис. 4.10). Заодно посмотрим, как изменится стиль отображения для варианта `style='whitegrid'`.

```

sns.set_style(style='whitegrid')

```

```
sns.stripplot(x='smoker', y='charges', data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Расходы', fontsize=12) # ось ординат
```

```
Ввод [6]: sns.set_style(style='whitegrid')
sns.stripplot(x='smoker', y='charges', data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Расходы', fontsize=12) # ось ординат

Out[6]: Text(0, 0.5, 'Расходы')
```

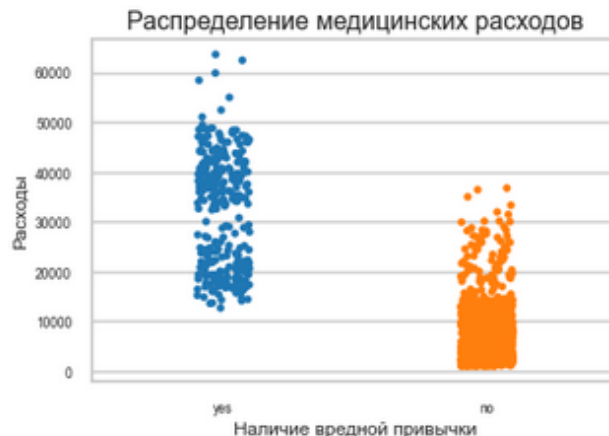


Рис. 4.10. График распределения для переменной charges

Параметры для функции relplot() (табл. 4.3) можно использовать и для stripplot(). Используем показатель sex для разделения данных по цвету (рис. 4.11) и протестируем новый стиль отображения для варианта style='darkgrid'.

```
sns.set_style(style='darkgrid')
sns.stripplot(x='smoker', y='charges', hue='sex', data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Расходы', fontsize=12) # ось ординат
```

Для построения графика «ящик с усами» используется функция boxplot(), по оси абсцисс – категориальная переменная, группирующая данные количественного показателя, по которому строятся «ящики с усами». Для группировки данных выберем показатель smoker, для построения «ящиков с усами» – переменную charges (рис. 4.12).

```
sns.set_style(style='white')
sns.set_context('paper')
sns.boxplot(x="smoker", y="charges", data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
```

```
Ввод [7]: sns.set_style(style='darkgrid')
sns.stripplot(x='smoker', y='charges', hue='sex', data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
plt.ylabel('Расходы', fontsize=12) # ось ординат
```

```
Out[7]: Text(0, 0.5, 'Расходы')
```



Рис. 4.11. График распределения для переменной charges

```
Ввод [8]: sns.set_style(style='white')
sns.set_context('paper')
sns.boxplot(x="smoker", y="charges", data=insurance_df)
plt.title('Распределение медицинских расходов', fontsize=16) # заголовок
plt.xlabel('Наличие вредной привычки', fontsize=12) # ось абсцисс
```

```
Out[8]: Text(0.5, 0, 'Наличие вредной привычки')
```

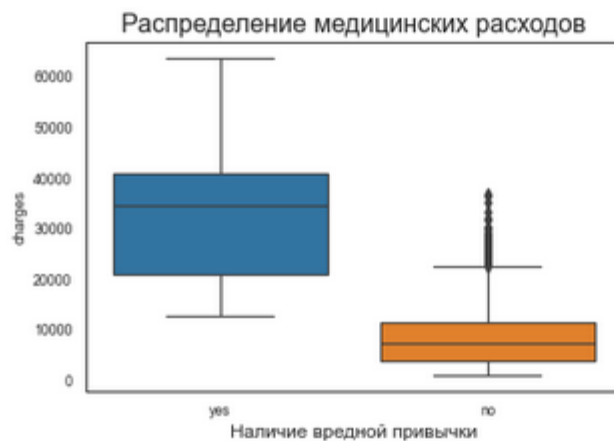


Рис. 4.12. График «ящик с усами»

Чтобы отобразить, какое количество раз в наборе данных встречается то или иное значение категориального признака, используется функция `countplot()`. Имеет вид столбчатой диаграммы. Визуализируем число застрахованных, проживающих в различных районах (рис. 4.13). С помощью параметра `color='b'` мы подавили раскраску столбиков в разные цвета.


```
sns.countplot(x="region", color='b', data=insurance_df)
plt.title('Распределение застрахованных по районам', fontsize=16) # заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

```
Ввод [9]: sns.countplot(x="region", color='b', data=insurance_df)
plt.title('Распределение застрахованных по районам', fontsize=16) # заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат

Out[9]: Text(0, 0.5, 'Количество застрахованных')
```

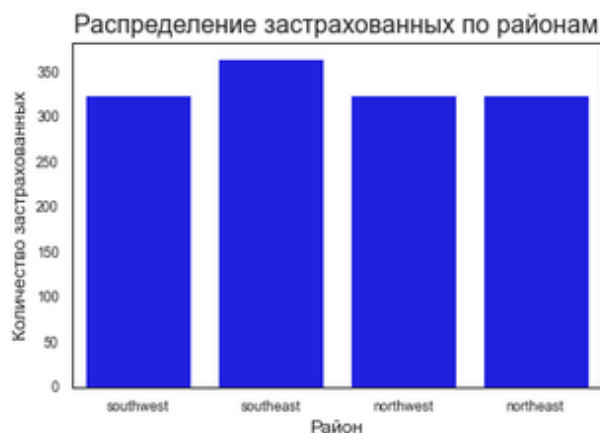


Рис. 4.13. Столбчатая диаграмма

Используем параметр из таблицы 4.3. для разделения данных по цвету с помощью показателя `smoker` (рис. 4.14).

```
sns.countplot(x="region", hue='smoker', data=insurance_df)
plt.title('Распределение застрахованных по районам', fontsize=16) # заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

Визуализацию распределений одномерных наборов данных можно провести с помощью функции `distplot()`. В качестве данных для построения в эту функцию передается столбец `DataFrame` или объект `Series`, т.е. одномерный массив или список, отдельного обращения ко всему набору данных не происходит.

Построим график плотности распределения вероятности для показателя `charges` с 40 бинами (рис. 4.15).

```
sns.distplot(insurance_df["charges"], bins=40)
plt.title('Медицинские расходы', fontsize=16) # заголовок
plt.xlabel('Расходы', fontsize=12) # ось абсцисс
plt.ylabel('Вероятность', fontsize=12) # ось ординат
```



```
Ввод [10]: sns.countplot(x="region", hue='smoker', data=insurance_df)
plt.title('Распределение застрахованных по районам', fontsize=16) # заголовок
plt.xlabel('Район', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат

Out[10]: Text(0, 0.5, 'Количество застрахованных')
```

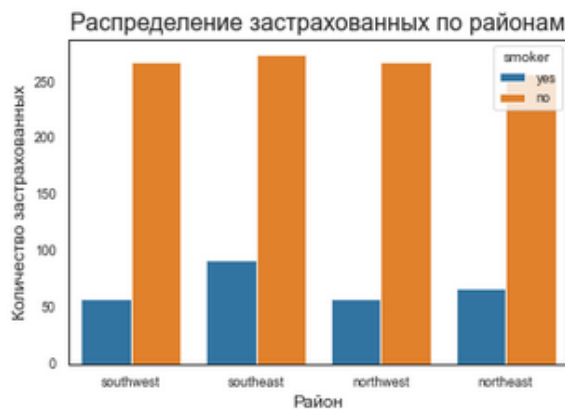


Рис. 4.14. Столбчатая диаграмма

```
Ввод [11]: sns.distplot(insurance_df["charges"], bins=40)
plt.title('Медицинские расходы', fontsize=16) # заголовок
plt.xlabel('Расходы', fontsize=12) # ось абсцисс
plt.ylabel('Вероятность', fontsize=12) # ось ординат

Out[11]: Text(0, 0.5, 'Вероятность')
```

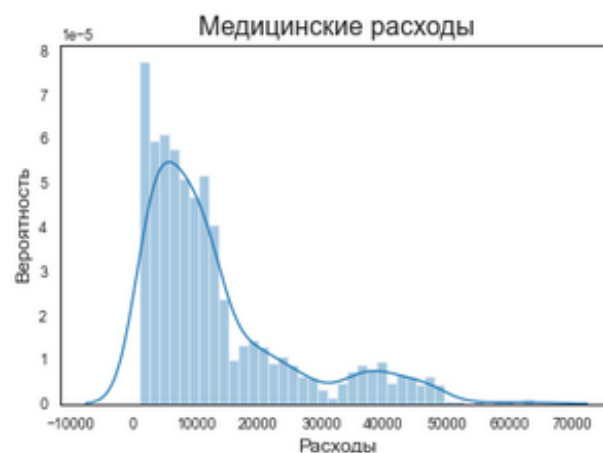


Рис. 4.15. Гистограмма для количественной переменной (вероятность)

Если подавить отображение графика функции плотности распределения вероятности с помощью параметра `kde=False`, то будет изображена гистограмма с частотой попадания наблюдений в интервал значений расходов (рис. 4.16).

```
sns.distplot(insurance_df["charges"], bins=40, kde=False)
plt.title('Медицинские расходы', fontsize=16) # заголовок
plt.xlabel('Расходы', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат
```

```
Ввод [12]: sns.distplot(insurance_df["charges"], bins=40, kde=False)
plt.title('Медицинские расходы', fontsize=16) # заголовок
plt.xlabel('Расходы', fontsize=12) # ось абсцисс
plt.ylabel('Количество застрахованных', fontsize=12) # ось ординат

Out[12]: Text(0, 0.5, 'Количество застрахованных')
```

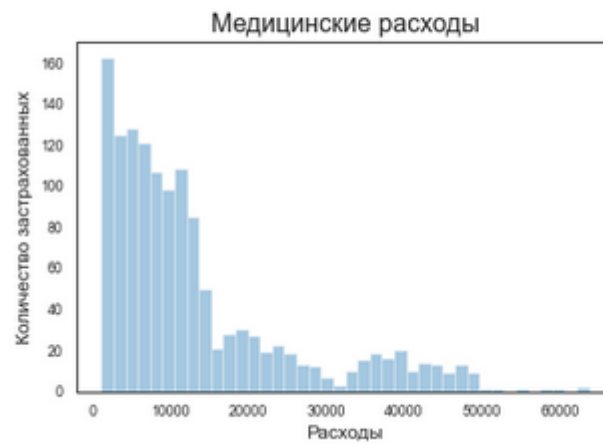


Рис. 4.16. Гистограмма для количественной переменной (частота)

5. Проверка статистических гипотез. Библиотека SciPy, модуль stats

На основе имеющихся данных можно сформулировать гипотезы, подлежащие проверке. Формулируются две гипотезы: нулевая H_0 и альтернативная H_1 .

Нулевая гипотеза – это некоторое утверждение о параметре(-ах) генеральной совокупности или распределении. Обычно нулевая гипотеза описывает ситуацию отсутствия различий, равенства какого-то параметра нулю и т.п. Альтернативная гипотеза – это утверждение, опровергающее нулевую гипотезу, именно альтернативная гипотеза содержит исследовательский вопрос. Альтернативная гипотеза формулируется, но не проверяется.

Важно понимать, что никакие экспериментальные данные не могут подтвердить ни одну из гипотез. Данные могут лишь не противоречить какой-либо гипотезе или показывать крайне маловероятные результаты в предположении, что гипотеза верна, и тогда ее обычно отклоняют.

Проверке подлежит именно нулевая гипотеза. Исходим из начального предположения, что H_0 истинна, и пытаемся это подтвердить, а в случае неудачи придерживаемся альтернативного варианта H_1 .

Очень важно также понимать, можно ли на основе данных проверить исследовательский вопрос.

Например, на основе данных набора `insurance_df` можно сформулировать несколько исследовательских вопросов:

- проверить нулевую гипотезу о равенстве средних медицинских расходов для всех застрахованных независимо от наличия вредной привычки какому-то конкретному значению (альтернативная гипотеза: средние расходы на медицинское обслуживание больше/меньше этого значения);

- проверить нулевую гипотезу о равенстве средних медицинских расходов для курящих/некурящих какому-то конкретному значению (альтернативная гипотеза: средние расходы на медицинское обслуживание для курящих/некурящих больше/меньше этого значения);

- проверить нулевую гипотезу об отсутствии влияния вредной привычки на величину медицинских расходов, т.е. средние значения медицинских расходов для курящих такие же, как и для некурящих

(альтернативная гипотеза: средние расходы на медицинское обслуживание для курящих отличны от средних расходов для некурящих).

Но проверить гипотезу об отсутствии различий в средних медицинских расходах тех, кто был застрахован, и тех, кто обходится без медицинской страховки, по имеющимся данным невозможно, так как известны медицинские расходы только для застрахованных.

Проверить гипотезу об отсутствии различий в средних медицинских расходах тех, кто никогда не имел вредной привычки, и тех, кто бросил курить, по имеющимся данным невозможно, так как известны данные о некурящих в текущий момент времени и нет данных об отсутствии этой привычки или ее наличии в прошлом.

Всегда нужно определять ограничения, которые накладывают на исследования имеющиеся данные. Но это не значит, что такие исследовательские вопросы бесполезны. То, что они выходят за рамки имеющихся данных, не означает отсутствие практической ценности в таких вопросах. Они позволяют понять, какие данные необходимы для проверки таких исследовательских вопросов, и обогатить данные из доступных источников или сформировать проект по сбору данных, определив, какие показатели необходимо регистрировать, чтобы решить подобную исследовательскую задачу в будущем.

В библиотеке SciPy в модуле stats есть методы `.ttest_1samp()` (для одновыборочного t-критерия, т.е. тестирования гипотезы о равенстве среднего выборки конкретному числу) и `.ttest_ind()` (для двухвыборочного t-критерия, проверяющего гипотезу о равенстве средних двух выборок). Параметры `.ttest_1samp()` представлены в табл. 5.1.

Таблица 5.1

Параметры `.ttest_1samp()`

Параметр	Описание
<code>a</code>	Массив наблюдений
<code>popmean</code>	Ожидаемое значение в нулевой гипотезе
<code>nan_policy</code>	Определяет, как обрабатывается массив с пропущенными значениями: 'propagate' – возвращает NaN, 'raise' – выдает ошибку, 'omit' – игнорирует
<code>alternative</code>	Параметр, определяющий, какая альтернативная гипотеза тестируется: "two.sided" – двусторонняя (значение по умолчанию), "greater" (больше, т.е. правосторонняя) или "less" (меньше, т.е. левосторонняя). По умолчанию значение "two.sided"

Важно! Параметр `alternative` есть только в версиях библиотеки SciPy > 1.6.0.

Параметры `.ttest_ind()` представлены в табл. 5.2.

Таблица 5.2

Параметры `.ttest_ind()`

Параметр	Описание
<code>a, b</code>	Массивы наблюдений должны иметь одинаковую форму за исключением размерности
<code>equal_var</code>	Если True (по умолчанию), то выполняется стандартный с двумя независимыми выборками, который предполагает равные дисперсии генеральной совокупности. Если False , то выполняется t-критерий Уэлча, который не предполагает равной дисперсии генеральной совокупности
<code>nan_policy</code>	Определяет, как обрабатывается массив с пропущенными значениями: 'propagate' – возвращает NaN, 'raise' – выдает ошибку, 'omit' – игнорирует
<code>alternative</code>	Параметр, определяющий, какая альтернативная гипотеза тестируется: "two.sided" – двусторонняя (значение по умолчанию), "greater" (больше, т.е. правосторонняя) или "less" (меньше, т.е. левосторонняя). По умолчанию значение "two.sided"

Важно! Параметр `alternative` есть только в версиях библиотеки SciPy > 1.6.0.

Для обращения к методу необходимо подключить из библиотеки `scipy` модуль `stats` и добавлять перед обращением к методу ссылку на модуль `stats`, импортированный из библиотеки `scipy`, т.е. `stats.ttest_1samp()` или `stats.ttest_ind()`.

Подключим необходимые библиотеки (рис. 5.1).

```
#Подключение библиотек
import pandas as pd
import numpy as np
from scipy import stats
```

```
In [1]: #Подключение библиотек
import pandas as pd
import numpy as np
from scipy import stats
```

Рис. 5.1. Подключение библиотек и модулей

Если предполагается тестировать односторонние альтернативы, а версия SciPy это не позволяет, то нужно обновить `conda` через

Anaconda Powershell Prompt. Обновлению подлежат все пакеты, так как в поставке дистрибутива нет конфликтующих версий библиотек и их методы после подключения библиотек могут работать совместно, а обновление до более поздней версии отдельной библиотеки может это нарушить.

В дистрибутиве Anaconda автора пособия библиотека SciPy имела версию 1.3.1, которая не позволяет воспользоваться нужными параметрами методов для t-критериев.

Узнать версию библиотеки можно в блокноте Jupyter:

```
import scipy
scipy.__version__
```

Или в Anaconda Powershell Prompt (рис. 5.2, 5.3).

```
conda list
```

```

Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\Svetlana> conda list
# packages in environment at C:\Users\Svetlana\Anaconda3:
#
# Name          Version          Build          Channel
_ipyw_jlab_nb_ext_conf 0.1.0            py37_0
alabaster        0.7.12           py37_0
altair           4.1.0            py37_0      pypi
anaconda         2019.10          py37_0
anaconda-client  1.7.2            py37_0
anaconda-navigator 1.9.7            py37_0
anaconda-project 0.8.3            py_0
asn1crypto       1.0.1            py37_0
astor            0.8.1            py37_0      pypi
astroid          2.3.1            py37_0
astropy          3.2.1            py37he774522_0
atomicwrites     1.3.0            py37_1
attrs            19.2.0           py_0
babel            2.7.0            py_0
backcall         0.1.0            py37_0
backports        1.0              py_2
backports.functools_lru_cache 1.6.4            pyhd3eb1b0_0
backports.os     0.1.1            py37_0
backports.shutil_get_terminal_size 1.0.0            py37_2
backports.tempfile 1.0              pyhd3eb1b0_1
backports.weakref 1.0.post1        py_1
base58           2.0.1            py37_0      pypi

```

Рис. 5.2. Вывод списка установленных модулей

```

scikit-learn     0.21.3           py37h6288b17_0
scipy            1.3.1            py37h29ff71c_0
seaborn          0.9.0            py37_0

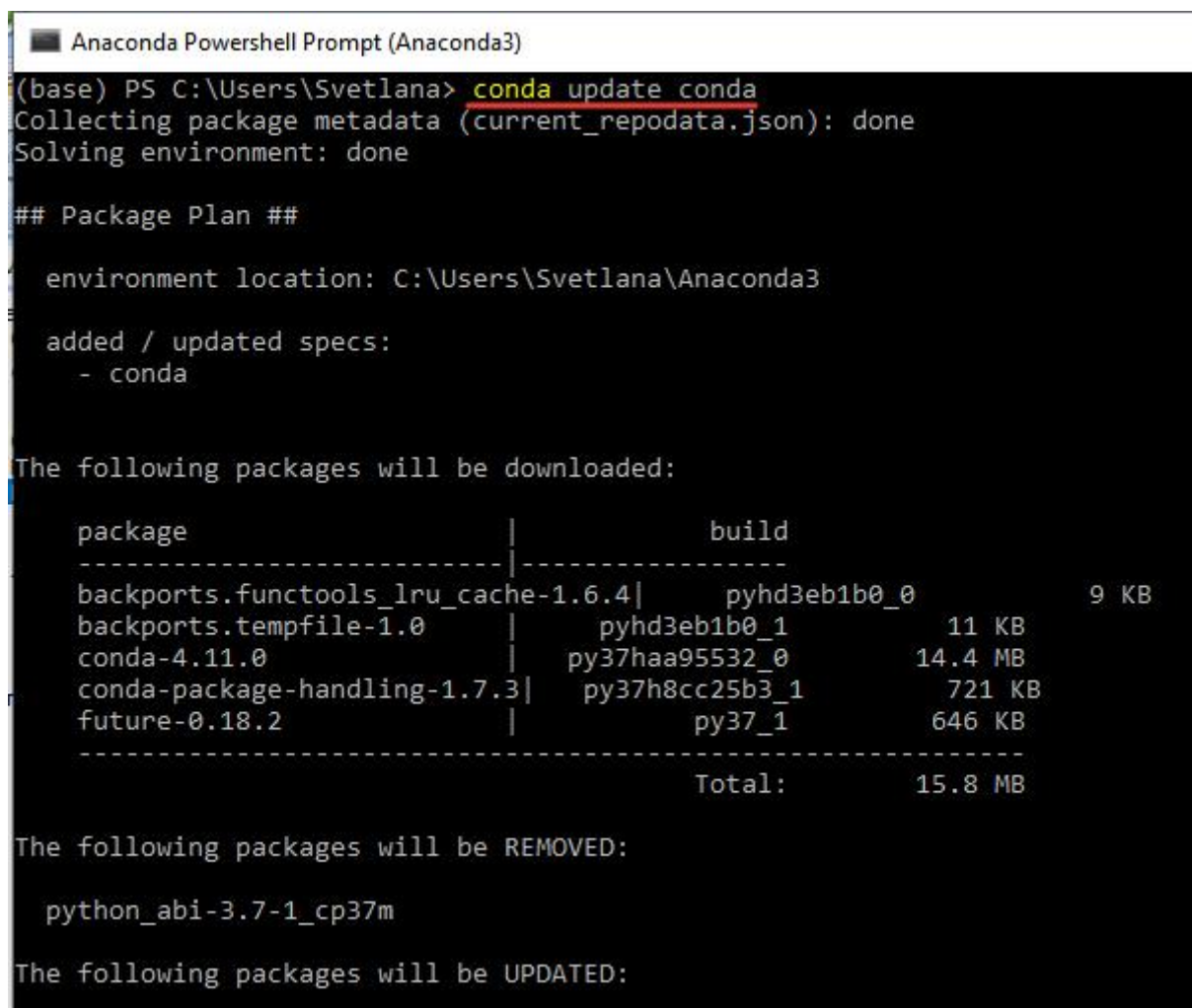
```

Рис. 5.3. Продолжение списка

Важно! Перед запуском обновлений нужно обязательно закрыть Jupyter.

Для корректного обновления вначале используем команду обновления менеджера управления библиотеками conda (рис. 5.4)

```
conda update conda
```



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\Svetlana> conda update conda
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Svetlana\Anaconda3

added / updated specs:
- conda

The following packages will be downloaded:

package                                     build                                     9 KB
-----
backports.functools_lru_cache-1.6.4|      pyhd3eb1b0_0
backports.tempfile-1.0                 |      pyhd3eb1b0_1      11 KB
conda-4.11.0                           |      py37haa95532_0     14.4 MB
conda-package-handling-1.7.3|          py37h8cc25b3_1      721 KB
future-0.18.2                          |          py37_1       646 KB
-----
Total:                                     15.8 MB

The following packages will be REMOVED:

python_abi-3.7-1_cp37m

The following packages will be UPDATED:
```

Рис. 5.4. Обновление менеджера conda

После загрузки файлов обновлений необходимо подтвердить запуск процесса обновления (рис. 5.5).

Далее необходимо обновить все библиотеки (рис. 5.6).

```
conda update --all
```

После загрузки файлов обновлений (это займет некоторое время) необходимо опять подтвердить, что обновление библиотек должно быть запущено (рис. 5.7).


```

Anaconda Powershell Prompt (Anaconda3)

The following packages will be UPDATED:

backports.functool~          1.5-py_2 --> 1.6.4-pyhd3eb1b0_0
conda                        conda-forge::conda-4.8.3-py37hc8dfbb8~ --> pkgs/main::conda-4.11.0-py37haa95532_0
conda-package-han~          1.6.0-py37h62dcd97_0 --> 1.7.3-py37h8cc25b3_1
future                        0.17.1-py37_0 --> 0.18.2-py37_1

The following packages will be DOWNGRADED:

backports.tempfile          1.0-py_1 --> 1.0-pyhd3eb1b0_1

Proceed ([y]/n)? y

Downloading and Extracting Packages
future-0.18.2               | 646 KB | ##### |
100%
conda-4.11.0                | 14.4 MB | ##### |
100%
conda-package-handli        | 721 KB | ##### |
100%
backports.tempfile-1        | 11 KB | ##### |
100%
backports.functools_        | 9 KB | ##### |
100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

```

Рис. 5.5. Подтверждение обновления менеджера conda

```

Anaconda Powershell Prompt (Anaconda3)

(base) PS C:\Users\Svetlana> conda update --all
Collecting package metadata (current_repodata.json): done
Solving environment: |
Warning: 2 possible package resolutions (only showing differing packages):
- defaults/noarch::sphinx-4.2.0-pyhd3eb1b0_1, defaults/win-64::docutils-0.17.1-py37haa95532_1
- defaults/noarch::sphinx-2.2.0-py_0, defaults/win-64::docutils-0.18.1-py37haa95532done

## Package Plan ##

environment location: C:\Users\Svetlana\Anaconda3

The following packages will be downloaded:

package | build | size
-----|-----|-----
_anaconda_depends-2021.11 | py37_0 | 7 KB
anaconda-custom | py37_1 | 3 KB
anaconda-client-1.9.0 | py37haa95532_0 | 170 KB
anaconda-navigator-2.1.1 | py37_0 | 5.4 MB
anaconda-project-0.10.2 | pyhd3eb1b0_0 | 218 KB
anyio-2.2.0 | py37haa95532_2 | 125 KB
appdirs-1.4.4 | pyhd3eb1b0_0 | 12 KB
argcomplete-1.12.3 | pyhd3eb1b0_0 | 35 KB
argh-0.26.2 | py37_0 | 36 KB
argon2-cffi-20.1.0 | py37h2bbff1b_1 | 49 KB
arrow-0.13.1 | py37_0 | 83 KB
asn1crypto-1.4.0 | py_0 | 80 KB
astroid-2.6.6 | py37haa95532_0 | 308 KB
astropy-4.3.1 | py37hc7d831d_0 | 6.0 MB

```

Рис. 5.6. Обновление библиотек


```

Anaconda Powershell Prompt (Anaconda3)

The following packages will be DOWNGRADED:

anaconda                2019.10-py37_0 --> custom-py37_1
backports               1.0-py_2 --> 1.0-pyhd3eb1b0_2
glob2                   0.7-py_0 --> 0.7-pyhd3eb1b0_0
heapdict               1.0.1-py_0 --> 1.0.1-pyhd3eb1b0_0
jdcal                   1.4.1-py_0 --> 1.4.1-pyhd3eb1b0_0
lzo                     2.10-h6df0209_2 --> 2.10-he774522_2
pycosat                0.6.3-py37hfa6e2cd_0 --> 0.6.3-py37h2bbff1b_0
sphinxcontrib-jsm~     1.0.1-py_0 --> 1.0.1-pyhd3eb1b0_0
win_inet_pton          1.1.0-py37_0 --> 1.1.0-py37haa95532_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
networkx-2.6.3      | 1.3 MB | ##### |
100%
yaml-0.2.5         | 62 KB | ##### |
100%
libaec-1.0.4       | 31 KB | ##### |
100%
poyo-0.5.0         | 17 KB | ##### |
100%
pycurl-7.44.1      | 65 KB | ##### |
100%
jupyterlab_server-2. | 48 KB | ##### |
100%

```

Рис. 5.7. Подтверждение обновления библиотек

Обновление займет некоторое время. После приглашения к вводу новой инструкции можно опять запросить список библиотек и убедиться, что версия SciPy подходит для проведения тестирования гипотез (рис. 5.8).

conda list

```

scikit-learn        1.0.1      py37hf11a4ad_0
scikit-learn-intelex 2021.4.0   py37haa95532_0
scipy                1.7.3      py37h0a974cb_0
seaborn              0.11.2     pyhd3eb1b0_0

```

Рис. 5.8. Часть списка библиотек дистрибутива Anaconda

5.1. Тестирование гипотез для одной выборки

Пусть компания, которая занимается оформлением полисов добровольного медицинского страхования, рассчитала годовой взнос по страховке из расчета, что средние расходы на медицинское обслуживание составляют 12 000 д.е. Исследовательский вопрос, который можно проверить на основе имеющихся данных о реальных расходах на медицинское обслуживание застрахованных: соответствует ли средний расход ожидаемому.

Альтернативная гипотеза двусторонняя: средние расходы отличаются от значения, использованного для расчета стоимости страховки (те самые 12 000 д.е.).

Альтернативная гипотеза односторонняя: средние расходы больше, чем значение, использованное для расчета стоимости страховки.

Если мы не учитываем никакие входные данные о клиентах при расчете страхового взноса (т.е. стоимости полиса для клиента), т.е. для всех клиентов она стоит одинаково, то ошибка в значении, выбранном для среднего значения расходов на медицинское обслуживание в генеральной совокупности, может разорить страховую компанию.

Если записать гипотезы с использованием математических символов, то для двусторонней альтернативы

$$H_0 : \mu = \mu_0,$$

$$H_1 : \mu \neq \mu_0.$$

А в случае односторонней альтернативы

$$H_0 : \mu = \mu_0,$$

$$H_1 : \mu > \mu_0.$$

где μ_0 – это выбранное значение 12 000 д.е.

Уровень значимости определим как $\alpha = 0,05$. Для тестирования гипотез будем использовать *одновыборочный t-тест*.

Создадим DataFrame и проведем тестирование двусторонней альтернативы для столбца charges (рис. 5.2).

#Создание DataFrame из файла

```
insurance_df = pd.read_csv('insurance.csv', ',')
```

```
stats.ttest_1samp(insurance_df.charges, 12000)
```

```
In [1]: #Подключение библиотек
import pandas as pd
import numpy as np
from scipy import stats
```

```
In [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
```

```
In [3]: stats.ttest_1samp(insurance_df.charges, 12000)
```

```
Out[3]: Ttest_1sampResult(statistic=3.8373517196585314, pvalue=0.000130171652092872)
```

Рис. 5.9. Результаты тестирования (двусторонняя альтернатива)

Статистический вывод: предположение о том, что средние ожидаемые медицинские расходы составляют 12 000 д.е., не подтвердилось на уровне значимости 0,05 (для двусторонней альтернативной гипотезы). Нулевую гипотезу о равенстве средних расходов 12 000 отклоняем ($p\text{-value} < 0,0001302$).

Для тестирования односторонней альтернативы было проведено обновление всех библиотек. Но теперь выполнение тех же блоков кода дает несколько иной результат (рис. 5.10).

```
Ввод [1]: #Подключение библиотек
import pandas as pd
import numpy as np
from scipy import stats

Ввод [2]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')

C:\Users\Svetlana\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: FutureWarning: In a future version of pandas all arguments of read_csv except for the argument 'filepath_or_buffer' will be keyword-only
  exec(code_obj, self.user_global_ns, self.user_ns)

Ввод [3]: stats.ttest_1samp(insurance_df.charges, 12000)

Out[3]: Ttest_1sampResult(statistic=3.8373517196585314, pvalue=0.000130171652092872)
```

Рис. 5.10. Выполнение кода после обновления библиотек

Для подавления предупреждений, которые не являются ошибками и не мешают выполнению кода, можно использовать следующий код

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Добавим его в первый блок и выполним создание объекта DataFrame еще раз (рис. 5.11).

```
Ввод [8]: #Подключение библиотек
import pandas as pd
import numpy as np
from scipy import stats
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

Ввод [9]: #Создание DataFrame из файла
insurance_df = pd.read_csv('insurance.csv', ',')
```

Рис. 5.11. Подавление вывода предупреждений

Протестируем одностороннюю альтернативу для столбца charges (рис. 5.12).

```
stats.ttest_1samp(insurance_df.charges, 12000, alternative='greater')
```

```
Ввод [4]: stats.ttest_1samp(insurance_df.charges, 12000, alternative='greater')

Out[4]: Ttest_1sampResult(statistic=3.8373517196585314, pvalue=6.5085826046436e-05)
```

Рис. 5.12. Результаты тестирования
(односторонняя альтернатива)

Статистический вывод: средние ожидаемые медицинские расходы превышают 12 000 д.е. с вероятностью 0,95 (для односторонней альтернативной гипотезы). Нулевую гипотезу о равенстве средних расходов 12 000 отклоняем ($p\text{-value} < 6,509 \cdot 10^{-5}$).

5.2. Тестирование гипотез для двух выборок

Исследовательский вопрос, который можно сформулировать для компании, занимающейся медицинским страхованием: одинаковы ли средние расходы на медицинское обслуживание для курящих и некурящих застрахованных.

Однако хотя выборки отдельно по курящим застрахованным и некурящим независимые и количество наблюдений достаточно большое, чтобы приблизить выборки к нормально распределенной случайной величине, то нет оснований полагать, что обе выборки происходят из генеральной совокупности с одинаковыми дисперсиями.

В этом случае используют модификацию Уэлча для исходного теста Стьюдента.

Альтернативная гипотеза двусторонняя: средние расходы для двух выборок различны (не одинаковы).

Если записать гипотезы с использованием математических символов, то для двусторонней альтернативы

$$H_0 : \mu_1 = \mu_2 \text{ или } \mu_1 - \mu_2 = 0,$$

$$H_1 : \mu_1 \neq \mu_2 \text{ или } \mu_1 - \mu_2 \neq 0.$$

Уровень значимости определим как $\alpha = 0,05$. Используем *двухвыборочный t-тест для независимых выборок*.

Предварительно подготовим данные для проведения анализа. Создадим срез данных столбца charges для наблюдений, в которых у застрахованных есть вредная привычка или она отсутствует. Переиндексация строк в созданных наборах для выполнения теста необязательна.

```
ins_charg_nsmk=insurance_df.charges[insurance_df['smoker']=='no']
ins_charg_nsmk.reset_index(inplace=True, drop=True)
ins_charg_ysmk=insurance_df.charges[insurance_df['smoker']=='yes']
ins_charg_ysmk.reset_index(inplace=True, drop=True)
```

Используем метод stats.ttest_ind с параметром equal_var=False (рис. 5.13).

```
stats.ttest_ind(ins_charg_nsmk, ins_charg_ysmk, equal_var=False)
```

Статистический вывод: с вероятностью 95 % средние медицинские расходы для курящих отличаются от средних медицинских рас-

ходов для некурящих. Гипотезу о равенстве средних расходов у этих двух групп отклонили ($p\text{-value} < 5,9 \cdot 10^{-103}$).

```
Ввод [5]: ins_charg_nsmk=insurance_df.charges[insurance_df['smoker']=='no']
ins_charg_nsmk.reset_index(inplace=True, drop=True)
ins_charg_ysmk=insurance_df.charges[insurance_df['smoker']=='yes']
ins_charg_ysmk.reset_index(inplace=True, drop=True)

Ввод [6]: stats.ttest_ind(ins_charg_nsmk, ins_charg_ysmk, equal_var=False)

Out[6]: Ttest_indResult(statistic=-32.751887766341824, pvalue=5.88946444671698e-103)
```

Рис. 5.13. Результаты тестирования

Срезы данных можно было передать в метод, не создавая новые переменные (рис. 5.14).

```
stats.ttest_ind(insurance_df.charges[insurance_df['smoker']=='yes'],
               insurance_df.charges[insurance_df['smoker']=='no'],
               equal_var=False)

Ввод [7]: stats.ttest_ind(insurance_df.charges[insurance_df['smoker']=='yes'],
                           insurance_df.charges[insurance_df['smoker']=='no'],
                           equal_var=False)

Out[7]: Ttest_indResult(statistic=32.751887766341824, pvalue=5.88946444671698e-103)
```

Рис. 5.14. Результаты тестирования

6. Лабораторные работы

6.1. Лабораторная работа «Импорт и преобразование данных»

Цель работы: научиться загружать наборы данных, изменять тип данных, делать срезы данных по условию.

Формируемые знания, умения и навыки: знать основные типы и структуры данных. Уметь загружать данные, преобразовывать данные. Владеть навыками создания срезов данных.

Необходимо:

1. Используя ресурс kaggle: <https://www.kaggle.com/>, выбрать один из наборов данных. Загрузить этот набор в рабочую директорию. Считать данные, определить тип данных.
2. Описать данные набора: какие переменные в нем присутствуют, какой тип данных у этих переменных.

Контрольные вопросы и задания

1. Как выполняется считывание данных?
2. Какие проблемы при импорте данных могут возникнуть, какие способы их решения могут быть использованы?
3. Есть ли в наборе данных количественная дискретная переменная?
4. Есть ли в данных категориальные переменные? Как преобразовать их в числовые векторы?
5. Есть ли в данных количественные переменные? Какой у них тип?
6. Как построить срез данных по условию? Как провести сортировку набора данных по одному из столбцов?

6.2. Лабораторная работа «Описательный анализ данных»

Цель работы: научиться на основе параметров описательных статистик делать выводы о распределении данных.

Формируемые знания, умения и навыки: знать основные меры: центральной тенденции, положения, рассеяния. Уметь определять

тип данных, вычислять статистические параметры для данных выборки с использованием методов библиотеки pandas на языке Python. Владеть навыками использования описательной статистики для определения характера распределения данных.

Необходимо:

1. Рассчитать параметры описательной статистики для переменных набора с использованием методов библиотеки pandas языка Python. Сделать содержательные выводы.

Контрольные вопросы и задания

1. С помощью параметров описательной статистики опишите одну из количественных переменных и одну из категориальных переменных.

2. Есть ли в количественной переменной выбросы? Какое значение является выбросом и почему?

3. По какой категориальной переменной можно группировать данные? Какие группировки представляют интерес и почему?

6.3. Лабораторная работа «Визуализация данных. Разведочный анализ данных»

Цель работы: научиться использовать визуализации для формулирования исследовательских гипотез, проверки предположений относительно распределения данных и зависимостей между показателями.

Формируемые знания, умения и навыки: знать основные визуализации для количественных и категориальных данных, уметь использовать для построения визуализаций методы библиотек для языка Python. Уметь анализировать построенные визуализации для описания распределения данных, которое они графически представляют.

Необходимо:

1. На том же наборе данных провести построение графиков на языке Python.

Контрольные вопросы и задания

1. Какие визуализации построены для количественных переменных? Какие методы библиотек на языке Python и с какими входными параметрами при этом использовались? Какие особенности распределения данных они позволяют наблюдать?

2. Какие визуализации построены для категориальных переменных? Какие методы библиотек на языке Python и с какими входными параметрами при этом использовались? Какие особенности распределения данных они позволяют наблюдать?

3. Какие визуализации построены для пар переменных? Какие методы библиотек на языке Python и с какими входными параметрами при этом использовались? Какие особенности распределения данных они позволяют наблюдать?

6.4. Лабораторная работа

«Исследовательский анализ данных»

Цель работы: научиться формулировать исследовательские вопросы для данных об объектах.

Формируемые знания, умения и навыки: знать, как на основе исследовательского вопроса формулируется нулевая гипотеза и альтернативная, уметь для выборки данных сформулировать гипотезы для тестирования и определить критерий для тестирования, владеть навыками тестирования гипотез на языке Python.

Необходимо:

1. На том же наборе данных поставить исследовательские вопросы, сформулировать нулевую и альтернативную гипотезы, выбрать статистический критерий для проверки гипотез.

2. Провести тестирование гипотез с использованием методов на языке Python. Сделать вывод по результатам тестирования.

Контрольные вопросы и задания

1. Какие исследовательские задачи можно сформулировать для вашего набора данных?

2. Какие гипотезы нужно протестировать для ответа на исследовательский вопрос? Альтернативная гипотеза, сформулированная вами, является односторонней или двусторонней?

3. Какой статистический критерий необходимо использовать при тестировании гипотез?

4. Какой уровень значимости был выбран? Что позволяет определить уровень значимости при тестировании гипотез? Как уровень значимости влияет на вывод?

5. Какой методы на языке Python использовались для тестирования гипотез? Какой содержательный вывод можно сделать по результатам тестирования?

Список литературы

1. Рындина С. В. Базовые возможности языка R для анализа данных. Пенза : Изд-во ПГУ, 2021. 56 с.
2. Официальный сайт Google Colaboratory. URL: <https://colab.research.google.com> (дата обращения: 10.12.2021).
3. Описание библиотеки matplotlib. URL: <https://matplotlib.org/> (дата обращения: 10.12.2021).
4. Описание библиотеки seaborn. URL: <https://seaborn.pydata.org/> (дата обращения: 10.12.2021).
5. Описание библиотеки NumPy. URL: <https://numpy.org/> (дата обращения: 10.12.2021).
6. Описание библиотеки pandas. URL: <https://pandas.pydata.org/> (дата обращения: 10.12.2021).
7. Statistical functions (scipy.stats). URL: <https://docs.scipy.org/doc/scipy/reference/stats.html> (дата обращения: 10.12.2021).
8. Петров Ю. Типы данных. URL: https://www.yuripetrov.ru/edu/python/ch_03.html (дата обращения: 10.12.2021).
9. Обзор типов данных в pandas. URL: <https://russianblogs.com/article/58231188678/> (дата обращения: 10.12.2021).
10. Абдрахманов М. Изучаем pandas. Урок 2. Структуры данных Series и DataFrame. URL: <https://devpractice.ru/pandas-series-and-data-frame-part2/> (дата обращения: 10.12.2021).
11. Рындина С. В. Анализ данных : учеб.-метод. пособие. Пенза : Изд-во ПГУ, 2021. 36 с.

Учебное издание

Рындина Светлана Валентиновна

Базовые возможности языка Python для анализа данных

Редактор *В. В. Устинская*
Технический редактор *М. Б. Жучкова*
Компьютерная верстка *М. Б. Жучковой*

Подписано в печать 30.03.2022.
Формат 60×84¹/₁₆. Усл. печ. л. 3,95.
Тираж 7. Заказ № 178.

Издательство ПГУ.
440026, Пенза, Красная, 40.
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru

