ИЦ «Гевисста»

Артем Груздев

КУРС

МАШИННОЕ ОБУЧЕНИЕ В R, PYTHON И H2O 5.9

Модуль 1

Предварительная подготовка данных

https://www.facebook.com/groups/gewissta

Я встал на стол, чтобы напомнить себе, что надо смотреть на вещи с разных точек зрения.

I. Вводная часть

Машинное обучение – раздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться на данных.

Независимые переменные (признаки, предикторы, фичи)

Зависимая переменная (целевая переменная, отклик, таргет)

Обучающая выборка

	employ	retire	gender	reside	churn
1	5	no	f	2	1
2	5	no	f	6	1
3	29	no	<NA>	NA	0
4	NA	no	<NA>	1	1
5	2	no	m	4	0

Учитель

Модель: сжатое описание взаимосвязи между предикторами и зависимой переменной

Регрессия
Классификация
Ранжирование

Независимые переменные (признаки, предикторы, фичи)

	employ	retire	gender	reside
1	5	no	f	2
2	5	no	f	6
3	29	no	<NA>	NA
4	NA	no	<NA>	1
5	2	no	m	4

Модель: сжатое описание групп данных (кластеров), правил группировки данных и т.д.

Кластеризация
Поиск ассоциативных правил
Сокращение размерности

ИЦ «Гевисста»

I. Вводная часть

I.1. Типы данных

Данные

Структурированные

упорядочены в вертикальные столбцы (поля) и горизонтальные строки (записи или наблюдения)

	marital	income	agecat	churn
1	Женат	77680.0	<31	1
2	Женат	37111.5	<31	1
3	Женат	NA	<NA>	1
4	Одинокий	NA	<31	0
5	Одинокий	16829.6	<31	1
6	Одинокий	57272.7	<31	1
7	Женат	NA	<31	1
8	<NA>	NA	<31	1
9	Одинокий	92167.3	<31	1
10	Женат	37135.5	<31	1

Полуструктурированные

тип структурированных данных, у которых нет строгой структуры моделей данных

*MovieList.json

```
{
    "result": [{
        "actor": "Vivien Leigh",
        "title": "Gone with the Wind",
        "director": "Victor Fleming",
        "description": "Going with the wind"
    },
    {
        "actor": "Michael J Fox",
        "title": "Back To The Future",
        "director": "Robert Zemeckis",
        "description": "Going back to the future"
    },
```

Неструктурированные

данные, которые не имеют определенной структуры, не предполагают наличия заранее определенных столбцов конкретного типа

I. Вводная часть

I.2. Типы переменных

I.2.1. Количественная переменная

Количественная переменная (continuous/numerical variable) – это переменная, которая может принимать бесконечное (неисчислимое) количество значений. Все непрерывные переменные являются характеристиками, которые количественно описывают продукт и измеряются в непрерывной шкале. Мы можем узнать, на сколько и во сколько раз одно значение больше/меньше другого, при этом второй тип сравнения не всегда возможен.

Примерами непрерывных переменных являются возраст, температура, доход. Мы можем вычислить средний возраст, среднюю температуру и средний доход. Возьмем доход, мы можем сказать, сколько людей у нас с доходом в 20000 рублей, сколько людей у нас с доходом в 40000 рублей, мы можем упорядочить значения: 20000, 40000, наконец, мы можем сказать, что, например, человек с доходом 40000 рублей на 20000 рублей богаче человека с доходом 20000 рублей.

Среди количественных шкал выделяют:

- шкалу интервалов;
- шкалу отношений;
- абсолютную шкалу.

I. Вводная часть

I.2. Типы переменных

I.2.1. Количественная переменная (продолжение)

Шкала интервалов состоит из одинаковых интервалов и имеет *условную нулевую точку* (точку отсчета). Она позволяет сказать, насколько одно значение больше другого, но не позволяет сказать, во сколько раз оно больше. Например, повысив температуру с 1ºС до 20 ºС, мы можем сказать, что температура 20 ºС на 19 градусов Цельсия больше 1ºС, но не можем сказать, что температура 20ºС в 20 раз больше, чем 1ºС *.
Шкала отношений отличается от шкалы интервалов тем, что имеет *естественную нулевую точку*. Она позволяет сказать, насколько одно значение больше другого и во сколько раз оно больше. Примером шкалы отношений может служить переменная *Возраст*: мы знаем, что расстояние между 25 и 30 в два раза меньше, чем расстояние между 30 и 40, 30-летний на 5 лет старше 25-летнего.
Шкалы большинства физических величин (длина, масса, сила, давление, скорость и др.) являются шкалами отношений. При этом единица измерения в этих шкалах может быть произвольной. Например, возраст можно измерять в годах, месяцах, неделях. Длину мы можем измерять в километрах, милях, лье.

* Из школьного курса физики вспомним, что температура среды (например, воздуха) определяется энергией молекул, составляющих эту среду. Для идеального газа внутренняя энергия равна сумме кинетических энергий его молекул, которая, в свою очередь, пропорциональна абсолютной температуре в кельвинах. Очевидно, что, например, при «двадцатикратном» нагреве с 1ºС до 20ºС абсолютная температура изменится всего в (273 + 20) / (273 + 1) = 1,069 раза. Ноль по шкале Цельсия условен и соответствует 273К.

I. Вводная часть

I.2. Типы переменных

I.2.1. Количественная переменная (продолжение)

Абсолютная шкала помимо естественной нулевой точки имеет еще и *естественную общепринятую единицу измерения*.
Пример абсолютной шкалы – абсолютная шкала температуры или шкала Кельвина. Нуль этой шкалы отвечает полному прекращению движения молекул, т.е. самой низкой температуре, а единицей измерения является кельвин, который равен 1/273,16 части термодинамической температуры тройной точки воды. Как и шкала отношений, абсолютная шкала также позволяет сказать, насколько одно значение больше другого и во сколько раз оно больше.
Резюмируя, можно сказать, что количественная переменная – самая информативная переменная, у нас есть информация о расстояниях между значениями, можем упорядочить значения, можем сказать, сколько наблюдений принадлежат конкретному значению.

I.2.2. Категориальная переменная

Категориальная переменная (categorical variable) – это переменная, которая может принимать одно значение из ограниченного и обычно фиксированного набора возможных значений. Каждое из возможных значений часто называется еще уровнем (level). Примерами категориальных переменных являются пол, штат, социальный класс, уровень благосостояния, группа крови, гражданство, образование.
Категориальные переменные являются качественными характеристиками, которые могут описать продукт, но при этом не измеряются в непрерывной шкале.

I. Вводная часть

I.2. Типы переменных

I.2.2. Категориальная переменная (продолжение)

Допустим, у нас есть переменная *Пол*. Она имеет уровни *Мужчина* и *Женщина*. Человек либо мужчина, либо женщина. Не существует золотой середины между полами, нельзя вычислить среднее значение пола.
Категориальные переменные бывают порядковыми и номинальными.
Порядковыми называют такие категориальные переменные, значения которых мы можем упорядочить. Допустим, у нас есть переменная *Уровень дохода*. Она имеет уровни *Низкий*, *Средний*, *Высокий*. Мы можем сказать, сколько у нас наблюдений в каждом уровне и можем упорядочить уровни: *Низкий*, *Средний*, *Высокий*. Такая переменная является порядковой, в ней есть определенный порядок. Человек с уровнем *Средний* богаче человека с уровнем *Низкий*, а человек с уровнем *Высокий* богаче человека с уровнем *Средний*, но на сколько точно богаче, мы сказать не можем. Таким образом, порядковая переменная будет менее информативной, чем количественная: у нас исчезает информация о расстояниях между значениями, мы можем упорядочить значения, можем сказать, сколько наблюдений принадлежат конкретному значению.
Номинальными называют переменные, у которых есть только названия уровней. Пример – переменные *Пол* и *Сфера деятельности*. Мы не можем сказать, что уровень *Женщина* хуже/лучше/меньше/больше уровня *Мужчина*. Возьмем переменную *Сфера деятельности* с уровнями *Строительство*, *Транспорт* и *Металлургия*. Мы не можем сказать, что уровень *Строительство* хуже/лучше/меньше/больше уровня *Металлургия*. Номинальная переменная будет менее информативной, чем порядковая: мы можем лишь сказать, сколько наблюдений принадлежат конкретному значению.

I. Вводная часть

I.2. Типы переменных

ТИП ПЕРЕМЕННОЙ	Количественная	Порядковая	Номинальная
Пример переменной	Количество лет, потраченных на образование (от 0 до 20 лет)	Уровень образования (начальное, среднее, высшее)	Название университета (МГУ, МИФИ, МФТИ)
Можем сказать, насколько одно значение больше или меньше другого?	Да	Нет	Нет
Можем упорядочить значения?	Да	Да	Нет
Можем сказать, сколько наблюдений для каждого значения?	Да	Да	Да

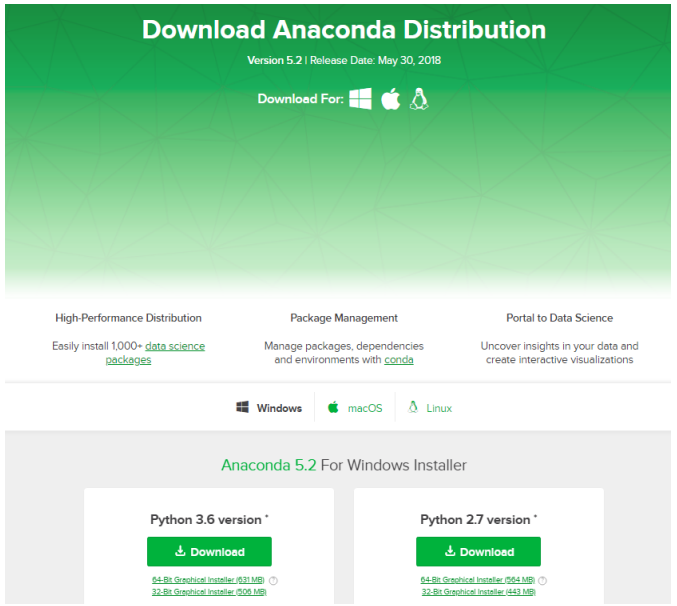
II. Знакомство с Python

Python стал одним из самых популярных языков, применяемых в машинном обучении для выполнения научных и коммерческих проектов. Он объединяет в себе возможности языков программирования общего назначения с простотой использования скриптовых предметно-ориентированных языков типа R. Python предлагает библиотеки для сбора данных из Интернета, построения графиков, статистической обработки и многого другого. Одно из основных преимуществ использования Python – возможность напрямую работать с программным кодом с помощью терминала или других инструментов типа Jupyter Notebook.

Для предварительной подготовки данных и построения моделей в Python нам потребуется ряд библиотек: NumPy, SciPy, matplotlib, pandas, IPython и scikit-learn. Настоятельно рекомендуем воспользоваться дистрибутивом Anaconda, который *уже* включает все необходимые библиотеки. Есть версии для Mac OS, Windows и Linux.

II.1. Установка Anaconda

Anaconda Python можно загрузить с веб-сайта Continuum Analytics по адресу https://www.anaconda.com/download/.
Веб-сервер определит операционную систему вашего браузера и предоставит вам соответствующий вашей системе файл загрузки.
При открытии этого URL-адреса в вашем браузере вы увидите страницу примерно следующего вида:



II. Знакомство с Python

II.1. Установка Anaconda

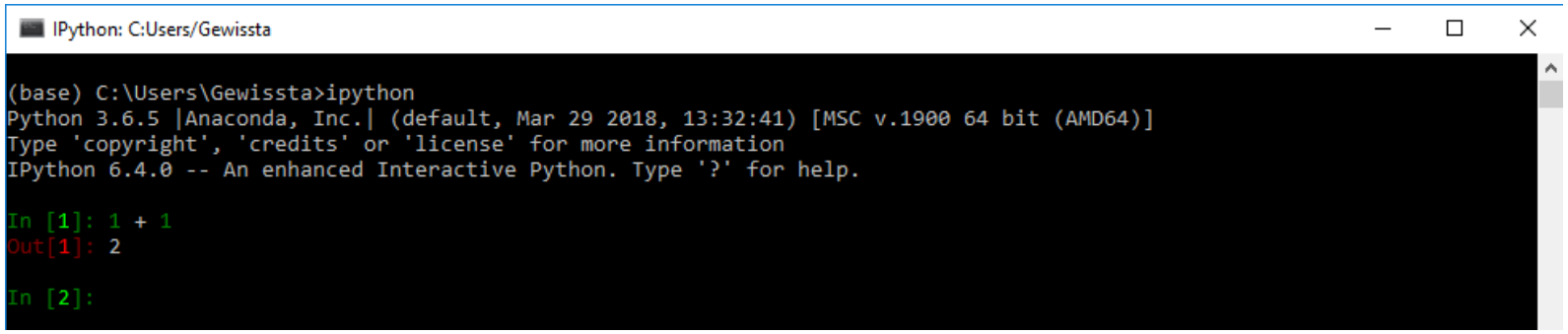
Загрузите инсталлятор для версии 3.6. Текущая версия Anaconda, которая будет использоваться в этом курсе – 5.2 с Python 3.6. Запустите инсталлятор и установите Anaconda Python.
Теперь, когда у нас установлено все необходимое, давайте перейдем к использованию IPython и Jupyter Notebook.

II.2. IPython и Jupyter Notebook

IPython – это альтернативная оболочка для интерактивной работы с Python. Она предлагает несколько усовершенствований для REPL, поставляемой по умолчанию. REPL – это форма организации простой интерактивной среды программирования в рамках средств интерфейса командной строки (REPL, от англ. *read-eval-print loop — цикл «чтение — вычисление — вывод»*), которая поставляется вместе с Python.
Чтобы запустить IPython, просто выполните команду `ipython` из командной строки/терминала.

II. Знакомство с Python

II.2. IPython и Jupyter Notebook

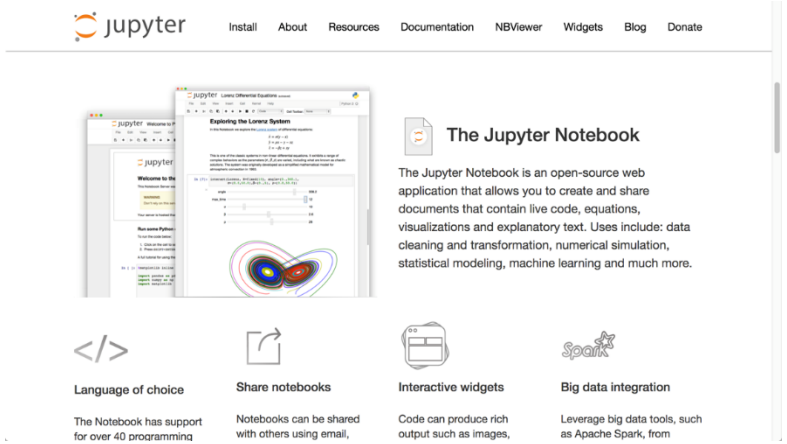


Командная строка ввода показывает `In[1]:`. Каждый раз, когда вы будете вводить инструкцию в REPL IPython, число в командной строке будет увеличиваться.
Аналогично, вывод для какой-либо конкретной записи будет предваряться `Out[x]:`, где x соответствует номеру `In[x]:`.
Данная нумерация операций ввода и выхода будет важна для примеров, поскольку все примеры будут предваряться `In[x]:` и `Out[x]:` и таким образом можно будет отследить последовательность выполнения операций.
Обратите внимание, что эти числа являются строго последовательными. Если вы запускаете программный код и при вводе возникают ошибки или вы вводите дополнительные инструкции, нумерация перестанет быть последовательной (ее можно сбросить, выйдя и перезапустив IPython).

II. Знакомство с Python

II.2. IPython и Jupyter Notebook

Jupyter Notebook – это результат эволюции IPython Notebook. Это веб-приложение с открытым исходным кодом, которое позволяет создавать и обмениваться документами, содержащими живой код, уравнения, визуализацию и разметку.
Первоначально IPython Notebook ограничивался лишь Python в качестве единственного языка. Jupyter Notebook позволил использовать многие языки программирования, включая Python, R, Julia, Scala и F#.
Если вы хотите глубже познакомиться с Jupyter Notebook, перейдите на http://jupyter.org/, где вы увидите страницу следующего вида:

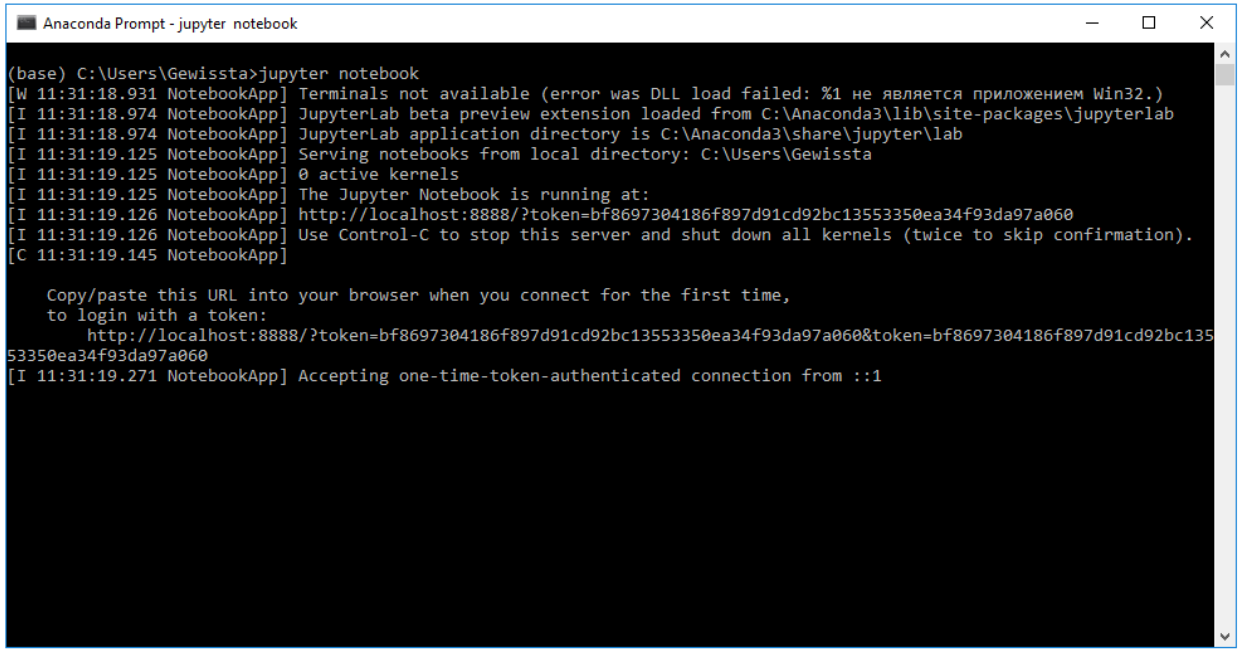


II. Знакомство с Python

II.2. IPython и Jupyter Notebook

Jupyter Notebook можно скачать и использовать независимо от Python. Anaconda устанавливает его по умолчанию. Чтобы запустить Jupyter Notebook, введите в Anaconda Prompt следующую команду:

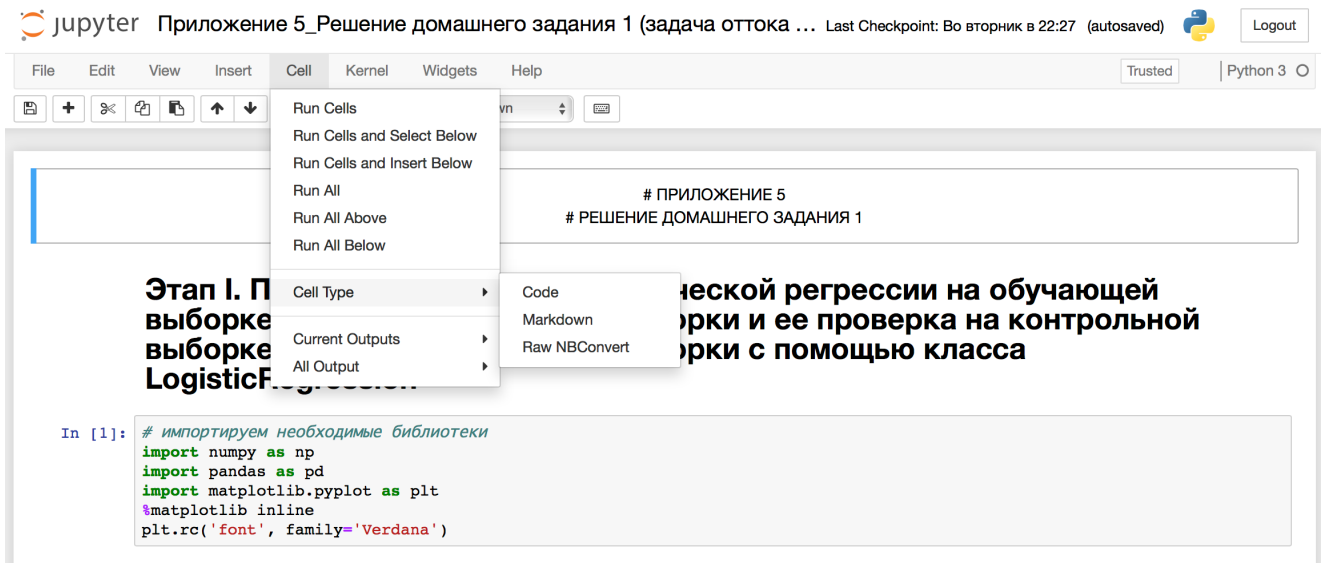
```
jupyter notebook
```



II. Знакомство с Python

II.2. IPython и Jupyter Notebook

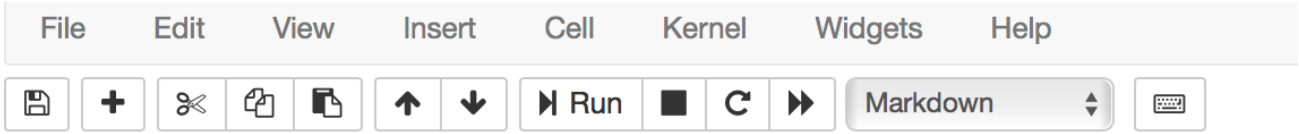
Откроется страница браузера, отображающая домашнюю страницу Jupyter Notebook (http://localhost:8888/tree). Если щелкнуть по файлу с расширением .ipynb, откроется страница с тетрадкой.



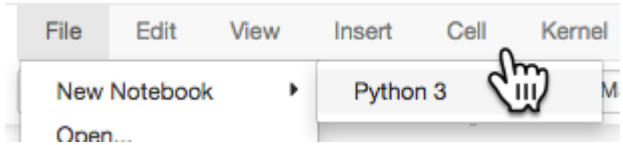
Отображаемая тетрадка представляет собой HTML-документ, который был создан Jupyter и IPython. Он состоит из нескольких ячеек, которые могут быть одного из трех типов: **Code** (активный программный код), **Markdown** (текст, поясняющий код, более развертутый, чем комментарий), **Raw NBConvert** (пассивный программный код). Jupyter запускает ядро IPython для каждой тетрадки. Ячейки, содержащие код Python, выполняются внутри этого ядра и результаты добавляются в тетрадку в формате HTML. Двойной щелчок по любой из этой ячеек позволит отредактировать ее. По завершении редактирования содержимого ячейки, нажмите *Shift* + *Enter*, после чего Jupyter/IPython проанализирует содержимое и отобразит результаты.

II. Знакомство с Python

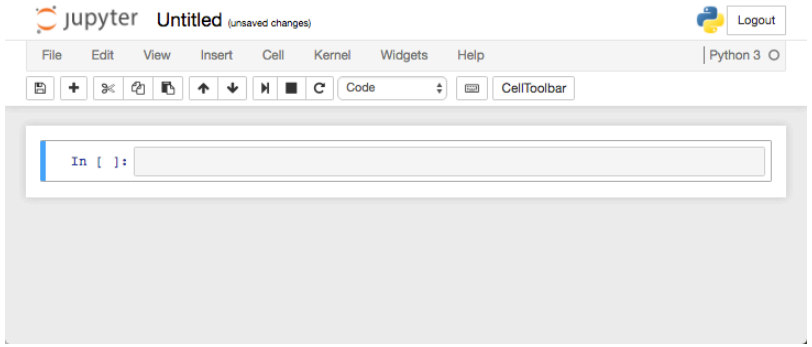
II.2. IPython и Jupyter Notebook



Панель инструментов в верхней части браузера предоставляет ряд возможностей по работе с тетрадкой. К ним относятся добавление, удаление и перемещение ячеек вверх и вниз в тетрадке. Также доступны команды для запуска ячеек, перезапуска ячеек и перезапуска основного ядра IPython.
Чтобы создать новую тетрадку, перейдите в меню **File** | **New Notebook** | **Python 3**:



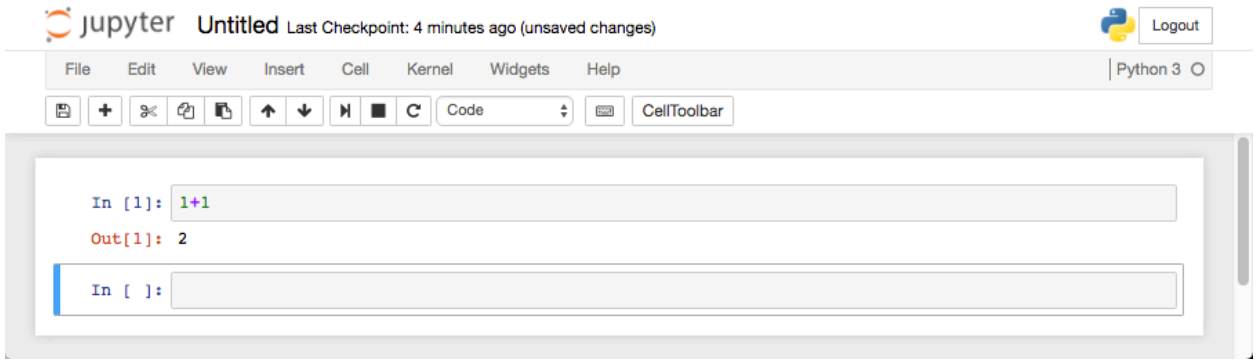
Страница новой тетрадки будет создана в новой вкладке браузера. Ее имя по умолчанию будет Untitled.



II. Знакомство с Python

II.2. IPython и Jupyter Notebook

Тетрадка состоит из одной ячейки Code, которая готова к вводу программного кода Python. Введите 1 + 1 в ячейку и нажмите *Shift* + *Enter* для выполнения.



Ячейка выполнена и результат показан как `Out[1]:`. Jupyter также создал новую ячейку, чтобы вы могли снова ввести код или разметку.

II. Знакомство с Python

II.2. IPython и Jupyter Notebook

В ячейке Markdown мы можем вводить и форматировать текст.

```
<center>
НАЗВАНИЕ
```

НАЗВАНИЕ

In [ ]:

```
# ЗАГОЛОВОК
```

ЗАГОЛОВОК

In [ ]:

```
## ЗАГОЛОВОК
```

ЗАГОЛОВОК

In [ ]:

Установить библиотеку catboost можно в Anaconda Prompt с помощью команды:

```
```
pip install catboost
```
```

Установить библиотеку catboost можно в Anaconda Prompt с помощью команды:

```
pip install catboost
```

In [ ]:

II. Знакомство с Python

II.2. IPython и Jupyter Notebook

`Класс LogisticRegression`

- **`penalty`** – задает тип регуляризации. Значение `l1` соответствует l1-регуляризации (лассо), значение `l2` соответствует l2-регуляризации (гребневой регрессии). Оптимизаторы `newton-cg`, `sag` и `lbfgs` поддерживают только `l2`. По умолчанию используется значение `l2`.

Класс LogisticRegression

- **penalty** – задает тип регуляризации. Значение `l1` соответствует l1-регуляризации (лассо), значение `l2` соответствует l2-регуляризации (гребневой регрессии). Оптимизаторы `newton-cg`, `sag` и `lbfgs` поддерживают только `l2`. По умолчанию используется значение `l2`.

In [ ]:

II. Знакомство с Python

II.3. NumPy

NumPy – это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами и различными математическими функциями.
В Python массив NumPy – это базовая структура данных. Библиотека scikit-learn, с помощью которой мы будем строить модели, требует, чтобы данные были записаны в виде массивов NumPy. Основа NumPy – это класс ndarray, многомерный (*n*-мерный) массив. Все элементы массива должны быть одного и того же типа.
Массив NumPy выглядит следующим образом:

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("массив NumPy:\n{}".format(x))
```

Чтобы получить вывод, запустите этот блок кода помощью кнопки  .

```
массив NumPy:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

II.4. SciPy

SciPy представляет собой набор математических и статистических функций для научных вычислений в Python.

II. Знакомство с Python

II.5. matplotlib

matplotlib – это основная библиотека для построения научных графиков в Python. Она включает функции для создания визуализаций типа линейных диаграмм, круговых диаграмм, гистограмм, диаграмм разброса и т. д. Визуализация данных и результатов анализа может дать вам важную информацию, и мы будем использовать matplotlib для построения некоторых визуализаций. При работе в Jupyter Notebook вы можете вывести рисунок прямо в браузере с помощью встроенных команд `%matplotlib notebook` и `%matplotlib inline`.

II. Знакомство с Python

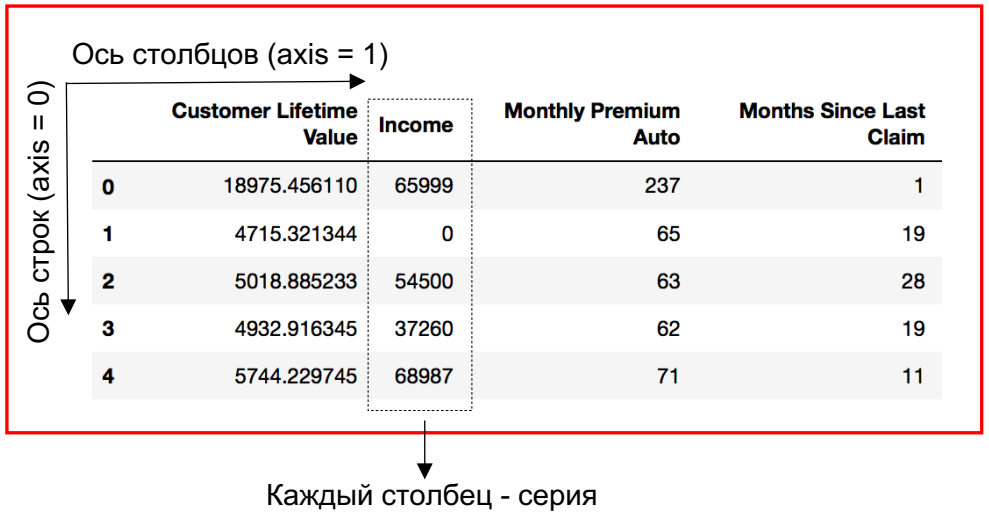
II.6. pandas

pandas – библиотека Python для обработки и анализа данных. Она построена на основе структуры данных, которая называется DataFrame и использует принципы таблицы данных (data frame) среды статистического программирования R. Именно в ней мы будем выполнять предварительную подготовку данных.
Объект `DataFrame` библиотеки pandas – это таблица, похожая на электронную таблицу Excel. Для простоты эту таблицу называют датафреймом. Если же говорить более точно, датафрейм представляет собой проиндексированный многомерный массив. В отличие от NumPy, который требует, чтобы все записи в массиве были одного и того же типа, каждый столбец датафрейма (объект `Series`) может иметь отдельный тип, то есть в столбцах могут быть записаны строковые значения, даты, целые числа, числа с плавающей точкой. Датафрейм имеет две размерности, ось строк 0 (двигаемся по датафрейму сверху вниз) и ось столбцов 1 (двигаемся по датафрейму слева направа).

II. Знакомство с Python

II.6. pandas

```
import pandas as pd
data = pd.read_csv('Data/StateFarm.csv', sep=';')
data.head()
```



	Customer Lifetime Value	Income	Monthly Premium Auto	Months Since Last Claim
0	18975.456110	65999	237	1
1	4715.321344	0	65	19
2	5018.885233	54500	63	28
3	4932.916345	37260	62	19
4	5744.229745	68987	71	11

Датафрейм,
индексация начинается с 0

[Посмотреть программный код к разделу II.6.]

Поговорим о типах переменных в библиотеке pandas.

II. Знакомство с Python

II.6. pandas

В питоновской библиотеке pandas количественным переменным будут соответствовать переменные типа `float` и переменные типа `int`. Для преобразования используются значения `int` и `float` параметра `dtype` метода `.astype()`.

`data.head()`

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
0	<2	<8	Нет	Бюджетный	CC	Мужской	Женат	77680.0	<31	1
1	NaN	<8	Нет	Бесплатный	CC	Мужской	Женат	37111.5	<31	1
2	<2	<8	Нет	NaN	CC	Мужской	Женат	NaN	NaN	1
3	<2	<8	NaN	Бесплатный	CH	Мужской	Одинокий	NaN	<31	0
4	<2	NaN	Нет	Бесплатный	Auto	NaN	Одинокий	16829.6	<31	1

`data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 10 columns):
longdist    4428 non-null object
local       4430 non-null object
int_disc    4430 non-null object
billtype    4429 non-null object
pay         4429 non-null object
gender      4430 non-null object
marital     4430 non-null object
income      4427 non-null float64
agecat      4430 non-null object
churn       4431 non-null int64
dtypes: float64(1), int64(1), object(8)
memory usage: 346.2+ KB
```

II. Знакомство с Python

II.6. pandas

В библиотеке pandas категориальным переменным будут соответствовать переменные типа `object`, также для этого можно использовать тип `str`. Для преобразования используются значения `object` и `str` параметра `dtype` метода `.astype()`.
Есть некоторые тонкости. При преобразовании в тип `object` значения `NaN` так и остаются значениями `NaN` и будут нуждаются в импутации, по итогам преобразования в тип `str` пропуски сформируют отдельную категорию `nan`. В ряде случаев это очень удобно, потому что часто пропуски для категориальных переменных выделяют в отдельную категорию.

`data.head()`

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
0	<2	<8	Нет	Бюджетный	CC	Мужской	Женат	77680.0	<31	1
1	NaN	<8	Нет	Бесплатный	CC	Мужской	Женат	37111.5	<31	1
2	<2	<8	Нет	NaN	CC	Мужской	Женат	NaN	NaN	1
3	<2	<8	NaN	Бесплатный	CH	Мужской	Одинокий	NaN	<31	0
4	<2	NaN	Нет	Бесплатный	Auto	NaN	Одинокий	16829.6	<31	1

`data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 10 columns):
longdist    4428 non-null object
local       4430 non-null object
int_disc    4430 non-null object
billtype    4429 non-null object
pay         4429 non-null object
gender      4430 non-null object
marital     4430 non-null object
income      4427 non-null float64
agecat      4430 non-null object
churn       4431 non-null int64
dtypes: float64(1), int64(1), object(8)
memory usage: 346.2+ KB
```

Обратите внимание, для порядковых переменных не предусмотрено отдельного типа, поэтому такие переменные представляют как переменные типа `int` или переменные типа `object`.

II. Знакомство с Python

II.6. pandas

Если для количественной переменной в качестве десятичного разделителя неверно используется запятая вместо точки, переменная будет неверно записана как категориальная.

`data.head()`

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
0	<2	<8	Нет	Бюджетный	CC	Мужской	Женат	77680	<31	1
1	NaN	<8	Нет	Бесплатный	CC	Мужской	Женат	37111,5	<31	1
2	<2	<8	Нет	NaN	CC	Мужской	Женат	NaN	NaN	1
3	<2	<8	NaN	Бесплатный	CH	Мужской	Одинокий	NaN	<31	0
4	<2	NaN	Нет	Бесплатный	Auto	NaN	Одинокий	16829,6	<31	1

`data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 10 columns):
longdist    4428 non-null object
local       4430 non-null object
int_disc    4430 non-null object
billtype    4429 non-null object
pay         4429 non-null object
gender      4430 non-null object
marital     4430 non-null object
income      4427 non-null object
agecat      4430 non-null object
churn       4431 non-null int64
dtypes: int64(1), object(9)
memory usage: 346.2+ KB
```

Для решения проблемы можно воспользоваться цепочкой методов `.str.replace()` и `.astype()`

```
data['income'] = data['income'].str.replace(',', '.').astype('float')
```

**Значения параметра dtype метода .astype()
для преобразования типов переменных в pandas**

Тип преобразования	Значение параметра dtype метода .astype()
в тип float	`float`
в тип int	`int`
в тип str	`str`
в тип object	`object`

II. Знакомство с Python

II.6. pandas

Для краткого знакомства с библиотекой pandas рекомендуется прочитать первую статью курса ODS https://habr.com/ru/company/ods/blog/322626/ (обновленную тетрадку можно найти здесь https://github.com/Yorko/mlcourse.ai/blob/master/jupyter_russian/topic01_pandas_data_analysis/topic1_habr_pandas.ipynb), затем статью Теда Петру «Минимально достаточный набор средств библиотеки pandas» под номером I в Сборнике переведенных статей к курсу.

Для глубокого изучения возможностей библиотеки pandas рекомендуются прочитать книгу Майкла Хейдта и Артема Груздева «Изучаем pandas» и книгу Ted Petrou «Master Data Analysis with Python Volume 1: Foundations of Data Exploration».

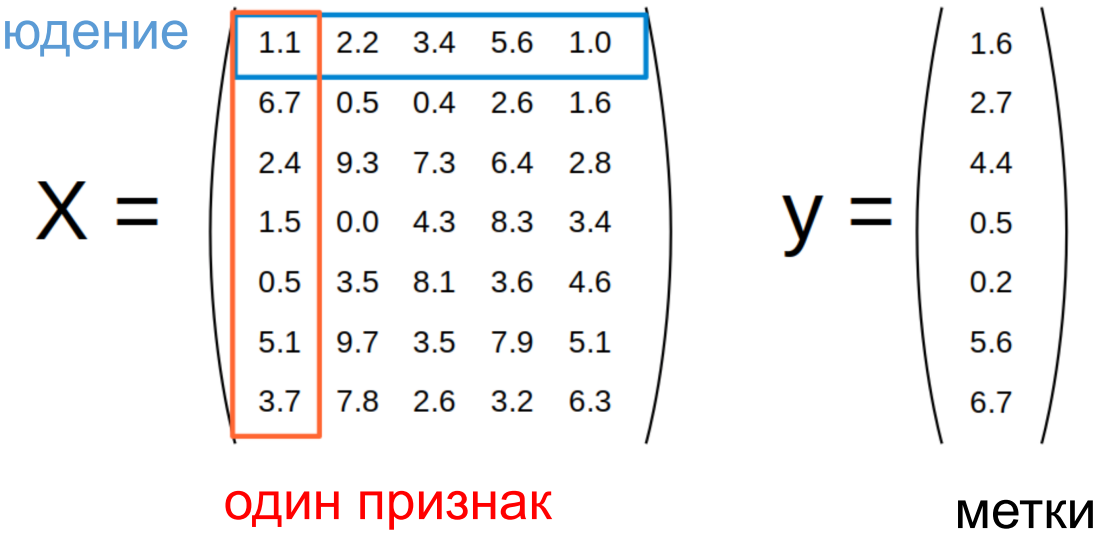
II. Знакомство с Python

II.7. scikit-learn

Библиотека scikit-learn – это проект с открытым исходным кодом. Ее можно свободно использовать и распространять. В ней реализованы методы предварительной обработки данных и методы машинного обучения, каждый метод снабжен подробной документацией.

II.7.1. Понятие массива признаков и массива меток

Для выполнения предварительной подготовки данных и построения модели машинного обучения с учителем нам нужен будет массив признаков – двумерный массив NumPy и массив меток зависимой переменной – одномерный массив NumPy. Все столбцы должны быть количественными признаками. В scikit-learn для массива данных используется заглавная X, а для массива меток – строчная y.



II. Знакомство с Python

II.7. scikit-learn

II.7.1. Понятие массива признаков и массива меток

Давайте попробуем создать массив признаков и массив меток. Для этого нужно импортировать необходимые библиотеки pandas, numpy, а также модуль os, затем загрузить данные. Модуль os содержит функции для работы с операционной системой, не зависящие от используемой операционной системы. Он позволяет взаимодействовать с операционной системой - узнавать/менять файловую структуру, переменные среды, узнавать имя и права пользователя и др.

```
# импортируем библиотеки pandas и numpy, модуль os
import pandas as pd
import numpy as np
import os
```

Взглянем на наш рабочий каталог.

```
# взглянем на наш рабочий каталог
os.getcwd()
```

```
'/Users/artemgruzdev/Documents/Курс/Course_ML/Модуль_1'
```

Если данные лежат в другом каталоге, можем сменить его.

```
# сменим рабочий каталог
os.chdir('/Users/artemgruzdev/Documents/Курс/Course_ML/Модуль_2')
```

В нашем случае смена каталога не нужна, поэтому задаем наш прежний каталог.

```
# вернем обратно
os.chdir('/Users/artemgruzdev/Documents/Курс/Course_ML/Модуль_1')
```

II. Знакомство с Python

II.7. scikit-learn

II.7.1. Понятие массива признаков и массива меток

Данные мы загружаем с помощью функции `read_csv()` библиотеки pandas. Они записаны в файле StateFarm.csv, который находится в каталоге Data нашего рабочего каталога `'/Users/artemgruzdev/Documents/Курс/Course_ML/Модуль_1'`.

```
# записываем CSV-файл в объект DataFrame
data = pd.read_csv('Data/StateFarm.csv', sep=';')
```

Разберем основные параметры функции `read_csv()` библиотеки pandas.

`pandas.read_csv(filepath_or_buffer,` ← задает путь к файлу

`sep=',',` ← задает символ – разделитель полей (по умолчанию `,`)

`header='infer',` ← задает номер строки, содержащей имена столбцов (если имена не передаются, аналогично `header=0` и имена берутся из первой строки файла)

`names=None,` ← задает список с имена столбцов

`index_col=None,` ← задает столбец, значения которого будут использоваться в качестве меток строк датафрейма

`usecols=None,` ← задает подмножество столбцов

`squeeze=False,` ← если спарсенные данные содержат лишь один столбец, возвращает объект `Series`

`decimal='.')` ← задает символ – десятичный разделитель (по умолчанию `.`)

```
# смотрим данные
data.head(3)
```

	Customer Lifetime Value	Income	Monthly Premium Auto	Months Since Last Claim	Months Since Policy Inception	Number of Open Complaints	Number of Policies	Response
0	18975.456110	65999	237	1	14	0	6	0
1	4715.321344	0	65	19	56	0	3	0
2	5018.885233	54500	63	28	17	0	6	0

II. Знакомство с Python

II.7. scikit-learn

II.7.1. Понятие массива признаков и массива меток

Исходный набор содержит данные американской автостраховой компании StateFarm. Он представляет собой записи о 8262 клиентах, классифицированных на два класса: 0 – отклика нет на предложение автостраховки (7435 клиентов) и 1 – отклик есть на предложение автостраховки (827 клиентов). По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- количественный предиктор *Пожизненная ценность клиента* [*Customer Lifetime Value*];
- количественный предиктор *Доход клиента* [*Income*]
- количественный предиктор *Размер ежемесячной автостраховки [Monthly Premium Auto]*;
- количественный предиктор *Количество месяцев со дня подачи последнего страхового требования [Months Since Last Claim]*
- количественный предиктор *Количество месяцев с момента заключения страхового договора [Months Since Policy Inception]*
- количественный предиктор *Количество открытых страховых обращений [Number of Open Complaints]*;
- количественный предиктор *Количество полисов [Number of Policies]*;
- категориальная зависимая переменная *Отклик на предложение автостраховки [Response]*.

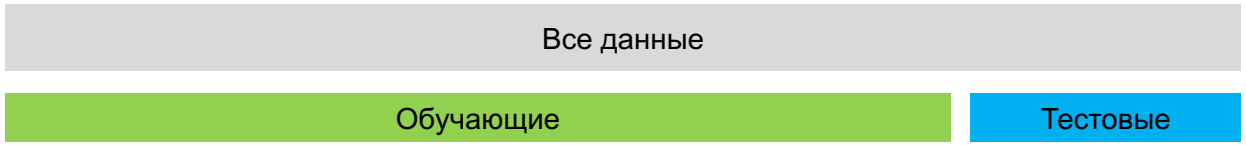
	Customer Lifetime Value	Income	Monthly Premium Auto	Months Since Last Claim	Months Since Policy Inception	Number of Open Complaints	Number of Policies	Response
0	18975.456110	65999	237	1	14	0	6	0
1	4715.321344	0	65	19	56	0	3	0
2	5018.885233	54500	63	28	17	0	6	0

II. Знакомство с Python

II.7. scikit-learn

II.7.2. Валидация

Обязательно нужно выполнить проверку (валидацию) модели, т.е. посмотреть, как модель выдает прогнозы на данных, не участвовавших в обучении. Самый простой способ проверки – случайное разбиение на обучающую и тестовую выборки.



Сначала мы случайным образом разбиваем имеющиеся данные на две выборки: обучающую и тестовую. Формирование тестовой выборки – это способ преодолеть такие несовершенства неидеального мира, как ограничения в объеме данных и ресурсов, а также невозможность получения дополнительных данных из порождающего распределения. В данном случае тестовая выборка должна представлять собой новые, еще неизвестные модели данные. Важно использовать тестовую выборку лишь однократно. Обычно 2/3 доступных данных назначают в обучающую выборку, а оставшуюся 1/3 данных – в тестовую выборку. Другими популярными методами разбиения на обучающую/тестовую выборки являются 60/40, 70/30, 80/20 или даже 90/10, если набор данных относительно велик.

Затем необходимо построить на обучающей выборке (обучить) модели предварительной подготовки – модель импутации, модель стандартизации, модель дамми-кодирования и модель машинного обучения, которая, как мы предполагаем, может оказаться подходящей для решения данной задачи. Модели в библиотеке scikit-learn реализованы в виде классов.

II. Знакомство с Python

II.7. scikit-learn

II.7.2. Валидация

У любой модели есть параметры, которые мы находим в ходе обучения. Например, у нас будет класс `SimpleImputer`, который обучает модель предварительной подготовки – модель импутации пропущенных значений. Здесь параметрами будут статистики, которые мы используем для импутации пропусков (среднее, медиана). Например, для класса `LogisticRegression`, строящего модель машинного обучения – модель логистической регрессии, параметрами будут регрессионные коэффициенты для соответствующих предикторов, для класса `DecisionTreeClassifier`, строящего дерево решений CART, параметрами будут правила расщепления (предиктор расщепления и расщепляющее значение).
Обратите внимание, что для моделей помимо понятия «параметр» есть понятие «гиперпараметр». Параметры мы находим в ходе обучения модели. А вот гиперпараметры нельзя «выучить» в процессе обучения, их задают перед обучением модели и настраивают на тестовой выборке. Модель импутации не может самостоятельно выяснить оптимальную стратегию импутации. Поэтому стратегия импутации – это гиперпараметр модели, который позволяет улучшить качество модели и настраивается на тестовой выборке (для этого у класса `SimpleImputer` есть гиперпараметр `strategy`). Логистическая регрессия не может самостоятельно выяснить оптимальное значение силы регуляризации. Поэтому сила регуляризации – это гиперпараметр модели, который позволяет улучшить качество модели и настраивается на тестовой выборке (для этого у класса `LogisticRegression` есть гиперпараметр `C`). Дерево решений CART не может самостоятельно выяснить оптимальное значение максимальной глубины. Поэтому максимальная глубина – это тоже гиперпараметр, который мы настраиваем на тестовой выборке (для этого у класса `DecisionTreeClassifier`, строящего дерево CART, есть гиперпараметр `max_depth`).

II. Знакомство с Python

II.7. scikit-learn

II.7.2. Валидация (продолжение)

После обучения модели на предыдущем шаге возникает закономерный вопрос: а насколько «хорошо» качество полученной модели? И вот теперь наступает время использовать независимую тестовую выборку. Поскольку модель еще «не видела» эти тестовые данные, такой шаг даст относительно надежную и несмещенную оценку качества на новых, незнакомых данных. Теперь мы берем тестовую выборку и используем модель для прогнозирования меток классов зависимой переменной по наблюдениям. Затем мы берем спрогнозированные метки классов и сравниваем их с фактическими метками классов для оценки обобщающей способности (здесь мы можем использовать правильность – долю правильно спрогнозированных наблюдений от общего количества наблюдений или ошибку классификации). Однако есть сложности.

Когда мы строим модели с разными значениями гиперпараметров на обучающей выборке, а проверяем их качество на тестовой выборке, возникает проблема. Мы используем тестовую выборку и для настройки гиперпараметров и для оценки качества модели. Поскольку мы использовали тестовую выборку для настройки гиперпараметров, мы больше не можем использовать ее для оценки качества модели. Это та же самая причина, по которой нам изначально нужно разбивать данные на обучающую и тестовую выборки. Теперь для оценки качества модели нам необходим независимый набор данных, то есть набор, который не использовался для построения модели и настройки ее параметров. В противном случае мы просто будем настраивать нашу модель под тестовую выборку, ведь любой выбор, сделанный, исходя из метрики на тестовом наборе, «сливает» модели информацию тестового набора. В итоге мы можем получить оптимистичные результаты.

II. Знакомство с Python

II.7. scikit-learn

II.7.2. Валидация (продолжение)

Для простоты пока пренебрежем этим недостатком случайного разбиения на обучающую и тестовую выборки, поскольку наша задача построить базовую модель машинного обучения, не прибегая к оптимизации гиперпараметров. Такое часто бывает, когда, например, дана задача классификации и нужно сопоставить качество несколько алгоритмов, строят базовые модели логистической регрессии, случайного леса и градиентного бустинга и сравнивают.

Итак, мы получили оценку качества модели на тестовых данных. Таким образом, уже нет смысла резервировать тестовую выборку. Теперь мы обучаем модель с оптимальными значениями гиперпараметров, найденными на тестовой выборке, на всех доступных данных и применяем модель, обученную на всех доступных данных, к новым данным.

Давайте сделаем случайное разбиение данных на обучающую и тестовую выборки: сформируем обучающий массив признаков, тестовый массив признаков, обучающий массив меток, тестовый массив меток. Это можно будет сделать с помощью функции `train_test_split()` модуля `model_selection` библиотеки scikit-learn.

```
# импортируем функцию train_test_split(), с помощью
# которой разбиваем данные на обучающие и тестовые
from sklearn.model_selection import train_test_split
```

II. Знакомство с Python

II.7. scikit-learn

II.7.2. Валидация (продолжение)

```
# разбиваем данные на обучающие и тестовые: получаем обучающий
# массив признаков, тестовый массив признаков, обучающий массив
# меток, тестовый массив меток
X_train, X_test, y_train, y_test = train_test_split(data.drop('Response', axis=1),
                                                    data['Response'],
                                                    test_size=0.3,
                                                    stratify=data['Response'],
                                                    random_state=42)
```

1. Указываем массив признаков. Для этого мы используем метод `.drop()` библиотеки pandas. Этот метод удаляет зависимую переменную *Response*, при удалении мы перемещаемся по оси столбцов, поэтому с помощью параметра `axis` указываем ось 1 (по умолчанию задана ось 0). По умолчанию операция не выполняется на месте (регулируется параметром `inplace`), в противном случае мы удалим зависимую переменную *Response* из датафрейма `data` и уже не сможем создать массив меток.

2. Затем создаем массив меток, просто указав зависимую переменную.

3. С помощью параметра `test_size` настраиваем нужный размер тестовой выборки (в процентах). По умолчанию 0,25.

4. С помощью параметра `stratify` (по умолчанию не используется) можно задать стратифицированное разбиение на обучение и контроль, чтобы распределение классов зависимой переменной в тестовой выборке соответствовало распределению классов в обучающей.

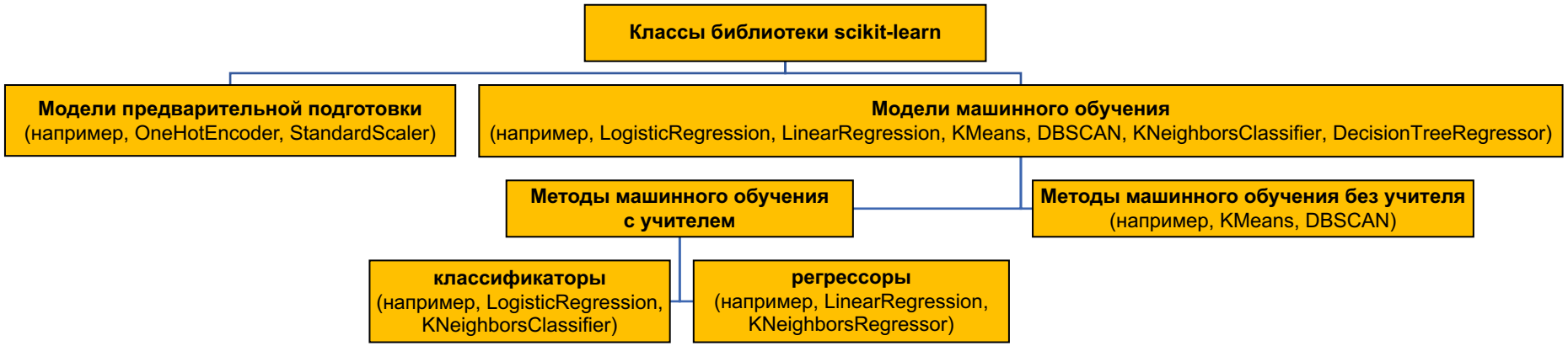
5. Поскольку разбиение является случайным, надо позаботиться о воспроизводимости результатов. Для этого с помощью параметра `random_state` задаем стартовое значение генератора случайных чисел.

Итак, мы получили обучающий массив признаков, тестовый массив признаков, обучающий массив меток, тестовый массив меток. Теперь мы можем использовать методы предварительной подготовки данных и методы машинного обучения, которые предлагает нам библиотека scikit-learn.

II. Знакомство с Python

II.7. scikit-learn

II.7.3. Классы, строящие модели предварительной подготовки данных, и классы, строящие модели машинного обучения



Как мы уже знаем, в библиотеке scikit-learn каждая модель предварительной подготовки данных и каждая модель машинного обучения реализована в собственном классе. При этом классы, в которых реализованы модели машинного обучения с учителям, называются классификаторами (classifier) или регрессорами (regressor). Классы-классификаторы обычно имеют название `[Метод_машинного_обучения]Classifier`, например, `DesicionTreeClassifier`, `RandomForestClassifier`. Классы-регрессоры обычно имеют название `[Метод_машинного_обучения]Regressor`, например, `DesicionTreeRegressor`, `RandomForestRegressor`.

II. Знакомство с Python

II.7. scikit-learn

II.7.3. Классы, строящие модели предварительной подготовки данных, и классы, строящие модели машинного обучения

Если модель машинного обучения с учителем позволяет выполнить только одну задачу – либо задачу регрессии, либо задачу классификации, то класс носит название модели, например, `LogisticRegression`, потому что логистическая регрессия решает только задачу классификации.

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

При работе с классом – моделью предварительной подготовки, мы выполняем следующие операции:

- импортируем из соответствующего модуля класс, в котором реализована соответствующая модель предварительной подготовки;
- создаем экземпляр класса – объект-модель;
- обучаем модель, т.е. вычисляем параметры, с помощью которых будем выполнять преобразование – используем метод `.fit()*` объекта-модели;
- применяем модель, т.е. выполняем преобразование – используем метод `.transform()` объекта-модели;
- либо обучаем и применяем модель сразу – используем метод `.fit_transform()` объекта-модели.

* Метод – функция, «принадлежащая» объекту – экземпляру класса. При вызове этот объект передается в качестве первого аргумента (обычно его называют `self`). Имя_объекта.имя_метода(параметры)

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Давайте воспользуемся классом `StandardScaler`, выполняющим стандартизацию. Самая простая стандартизация подразумевает, что из каждого значения переменной мы вычтем среднее значение и полученный результат разделим на стандартное отклонение.

$$\frac{x_i - mean(x)}{stdev(x)}$$

```
from sklearn.preprocessing import StandardScaler(copy=True,
                                                 with_mean=True,
                                                 with_std=True)
```

`copy=True,` — Если задано `False`, пробует избежать копирования и вместо этого выполняет стандартизацию на месте. Стандартизация на месте не всегда гарантируется. Например, если данные не являются массивом NumPy или CSR матрицей из модуля scipy.sparse, все равно может быть возвращена копия.

`with_mean=True,` — Центрирует данные (вычитает из исходного значения переменной среднее значение) перед тем, как поделить на стандартное отклонение.

`with_std=True)` — Делит на стандартное отклонение.

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Стандартизация необходима для некоторых методов машинного обучения, в частности, для регрессионных моделей (далее мы будем строить логистическую регрессию). Она приводит количественные независимые переменные к единому масштабу. Если не привести признаки к единому масштабу, то прогноз будут определять признаки, имеющие наибольший разряд и соответственно наибольшую дисперсию. Кроме того, единый масштаб позволит нам сравнивать регрессионные коэффициенты при предикторах между собой. В нашем наборе только количественные признаки, если бы здесь были категориальные признаки, мы обязательно превратили бы их в количественные с помощью дамми-кодирования (регрессионные модели работают только с количественными признаками и для моделирования мы используем массивы NumPy, каждый столбец которого должен быть количественным признаком; в ходе моделирования датафреймы pandas внутренне преобразовываются в массивы NumPy). У нас каждый уровень категориальной переменной стал бы отдельным бинарным столбцом со значениями 0 или 1 и такие переменные не нужно стандартизировать.

Мы импортируем класс `StandardScaler` и создаем его экземпляр:

```
# импортируем класс StandardScaler, выполняющий стандартизацию
from sklearn.preprocessing import StandardScaler
# создаем экземпляр класса StandardScaler
standardscaler = StandardScaler()
```

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Затем с помощью метода `.fit()` мы строим модель `standardscaler` на обучающем массиве признаков. В отличие от обычных моделей машинного обучения при вызове метода `.fit()` `standardscaler`, как и большинство классов, выполняющих предварительную подготовку, работает с массивом признаков (`X_train`), а массив меток (`y_train`) не используется. В данном случае метод `.fit()` вычисляет параметры модели – среднее значение и стандартное отклонение для каждой переменной обучающего массива признаков.

```
# обучаем модель стандартизации, т.е. по каждому признаку
# в обучающем массиве признаков вычисляем
# среднее значение признака и стандартное
# отклонение признака для трансформации
standardscaler.fit(X_train)
```

Чтобы применить модель к нашим данным, то есть фактически *отмасштабировать* (*scale*) обучающие и тестовые данные, мы воспользуемся методом `.transform()`. Метод `.transform()` используется в `scikit-learn`, когда модель возвращает новое представление данных.
Значения NaN обрабатываются как пропущенные значения: игнорируются при обучении и сохраняются в ходе применения.

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Фактически метод `.transform()` из каждого значения переменной обучающего и тестового массивов признаков вычитает среднее значение соответствующей переменной в обучающем массиве признаков и делит на стандартное отклонение этой переменной, также взятое по обучающему массиву признаков.

```
# применяем модель стандартизации к обучающему массиву признаков: из исходного
# значения признака вычитаем среднее значение признака, вычисленное
# по ОБУЧАЮЩЕМУ массиву признаков, и результат делим на стандартное
#отклонение признака, вычисленное по ОБУЧАЮЩЕМУ массиву признаков
X_train_standardscaled = standardscaler.transform(X_train)

# применяем модель стандартизации к тестовому массиву признаков: из исходного
# значения признака вычитаем среднее значение признака, вычисленное
# по ОБУЧАЮЩЕМУ массиву признаков, и результат делим на стандартное
# отклонение признака, вычисленное по ОБУЧАЮЩЕМУ массиву признаков
X_test_standardscaled = standardscaler.transform(X_test)
```

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Обратите внимание, что если вы используете такие операции, как укрупнение редких категорий по порогу, импутацию пропусков статистиками, стандартизацию, биннинг и конструирование признаков на основе статистик (frequency encoding, likelihood encoding), т.е. *любые операции, предполагающие вычисления по набору данных*, они должны быть осуществлены после разбиения на обучающую и тестовую выборки. Если мы используем случайное разбиение на обучающую и тестовую выборки и выполняем перечисленные операции до разбиения, получается, что при вычислении статистик для импутации пропусков, среднего и стандартного отклонения для стандартизации, правил биннинга, частот и вероятностей положительного класса зависимой переменной в категориях предиктора для frequency encoding и likelihood encoding соответственно использовались все наблюдения набора, часть из которых потом у нас войдут в тестовую выборку (по сути выборку новых данных). Поэтому получается, что статистики для импутации, статистики для стандартизации, которые мы получили на всем наборе, правила биннинга, частоты и вероятности положительного класса в категориях предиктора пришли к нам частично из «будущего» (из новой, тестовой выборки, которой по факту еще нет). Однако мы должны смоделировать наиболее близкую к реальности ситуацию, когда у нас есть только обучающая выборка, а никаких новых данных еще нет.

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Также обратите внимание, что при выполнении преобразований с помощью статистик мы всегда используем статистики, вычисленные в обучающей выборке. Импутируя переменную в обучающей и тестовой выборках с помощью статистик, мы всегда используем статистики, вычисленные в обучающей выборке. Вычисление статистик для импутации – это модель предварительной подготовки данных, которую мы строим на обучающей выборке и применяем ее к переменным обучающей и тестовой выборок. С помощью тестовой выборки – прообраза новых данных мы можем проверить эффективность той или иной модели импутации и выбрать лучшую (сравнить эффективность импутации средним, минимальным значением, медианой). Масштабируя переменную в обучающей и тестовой выборках, мы всегда используем среднее и стандартное отклонение переменной в обучающей выборке. Вычисление среднего и стандартного отклонения для каждой масштабируемой переменной – это тоже модель предварительной подготовки данных, которую мы строим на обучающей выборке и применяем ее к переменным обучающей и тестовой выборок. С помощью тестовой выборки – прообраза новых данных мы можем проверить эффективность той или иной модели стандартизации и выбрать лучшую (например, есть способ стандартизации, когда мы из каждого значения переменной вычитаем минимальное значение переменной и делим на ширину диапазона этой переменной).

II. Знакомство с Python

II.7. scikit-learn

II.7.4. Работа с классами, строящими модели предварительной подготовки данных

Поэтому нельзя отдельно вычислить статистики для импутации на обучающем наборе, а затем отдельно вычислить статистики для импутации на тестовом наборе и потом использовать эти значения для импутации переменной в соответствующем наборе. Нельзя отдельно вычислить среднее значение и стандартное отклонение переменной на обучающей выборке, а затем отдельно вычислить среднее значение и стандартное отклонение переменной на тестовой выборке и потом использовать эти значения для преобразования переменной в соответствующей выборке. Полезно помнить, что выборка новых данных может состоять из одного наблюдения (так чаще всего и бывает – клиентов, потенциальных заемщиков и т.д. оценивают по одному), и никакие статистики для импутации, статистики для стандартизации вычислить невозможно, все, что у нас есть – это обучающая выборка.