Артем Груздев

	CUSTOMER LIFETIME VALUE	COVERAGE	EDUCATION	EMPLOYMENT STATUS	GENDER	INCOME	MONTHLY PREMIUM AUTO
0	2763.519279	Basic	Bachelor	Employed	F	56274.0	NaN
1	NaN	NaN	Bachelor	Unemployed	F	0.0	NaN
2	NaN	NaN	NaN	Employed	F	48767.0	108.0
3	7645.861827	Basic	Bachelor	NaN	NaN	0.0	106.0
4	2813.692575	Basic	Bachelor	NaN	M	43836.0	73.0

Предварительная подготовка данных в Python

Том 2. План, примеры и метрики качества

ИЦ «Гевисста»

ДМК ПРЕСС
Издательство

А. В. Груздев

Предварительная подготовка данных в Python

Том 2

План, примеры и метрики качества

ДМК Пресс
издательство
Москва, 2023

УДК 004.04Python
ББК 32.372
Г90

Груздев А. В.
Г90 Предварительная подготовка данных в Python. Том 2: План, примеры и метрики качества. – М.: ДМК Пресс, 2023. – 814 с.: ил.

ISBN 978-5-93700-177-1

В двухтомнике представлены материалы по применению классических методов машинного обучения в различных промышленных задачах. Во втором томе рассматривается сам процесс предварительной подготовки данных, а также некоторые метрики качества и ряд полезных библиотек и фреймворков (H2O, Dask, Docker, Google Colab).

Издание рассчитано на специалистов по анализу данных, а также может быть полезно широкому кругу специалистов, интересующихся машинным обучением.

УДК 004.04Python
ББК 32.372

ISBN 978-5-93700-177-1

Оглавление

Введение

Настоящая книга является коллекцией избранных материалов из первого модуля Подписки – обновляемых в режиме реального времени материалов по применению классических методов машинного обучения в различных промышленных задачах, которые автор делает вместе с коллегами и учениками.

Автор благодарит Дмитрия Ларько за помощь в подготовке раздела по конструированию признаков в четвертой части книги, Уилла Керсена за предоставленные материалы к первому разделу шестой части книги.

Во втором томе мы разберем собственно процесс предварительной подготовки данных, обсудим некоторые метрики качества, рассмотрим ряд полезных библиотек и фреймворков.

Часть 4

План предварительной подготовки данных

1. Введение

До этого момента мы знакомились с инструментами – основными питоновскими библиотеками, классами и функциями, необходимыми для предварительной подготовки данных и построения моделей машинного обучения. Мы брали относительно простые примеры, выполняли предварительную подготовку данных и строили модели машинного обучения без глубокого понимания, зачем нужна та или иная операция предварительной подготовки и что происходит «под капотом» этой операции. В реальной практике мы так действовать не можем, нам нужен четкий план действий и глубокое понимание каждого этапа.

План предварительной подготовки данных, как правило, будет состоять из двух этапов. Первый этап – операции, которые можно выполнить до разбиения на обучающую и тестовую выборки / до цикла перекрестной проверки. Второй этап – операции, которые можно выполнить только после разбиения на обучающую и тестовую выборки / внутри цикла перекрестной проверки.

Если используются операции, использующие статистики, например укрупнение редких категорий по порогу, импутация пропусков статистиками, стандартизация, биннинг и конструирование признаков на основе статистик (frequency encoding, likelihood encoding), они должны быть осуществлены после разбиения на обучающую и тестовую выборки или внутри цикла перекрестной проверки.

Если мы используем случайное разбиение на обучающую и тестовую выборки и выполняем перечисленные операции до разбиения, получается, что для вычисления среднего и стандартного отклонения по каждому признаку для стандартизации, правил биннинга, частот и вероятностей положительного класса зависимой переменной в категориях признака использовались все наблюдения набора, часть из которых потом у нас войдет в тестовую выборку (по сути, выборку новых данных).

Если мы используем перекрестную проверку и выполняем перечисленные операции до перекрестной проверки, получается, что в каждом проходе перекрестной проверки для вычисления среднего и стандартного отклонения по каждому признаку для стандартизации, правил биннинга, частот и вероятностей положительного класса зависимой переменной в категориях признака использовались

все наблюдения набора, часть из которых у нас теперь находится в тестовом блоке (по сути, выборке новых данных). В таких случаях в Python используем классы `ColumnTransformer` и `Pipeline`. Случайное разбиение на обучающую и тестовую выборки и перекрестная проверка используются для сравнения конвейеров базовых моделей со значениями гиперпараметров по умолчанию. При подборе гиперпараметров лучшей практикой является комбинированная проверка, сочетающая случайное разбиение на обучающую и тестовую выборки и перекрестную проверку.

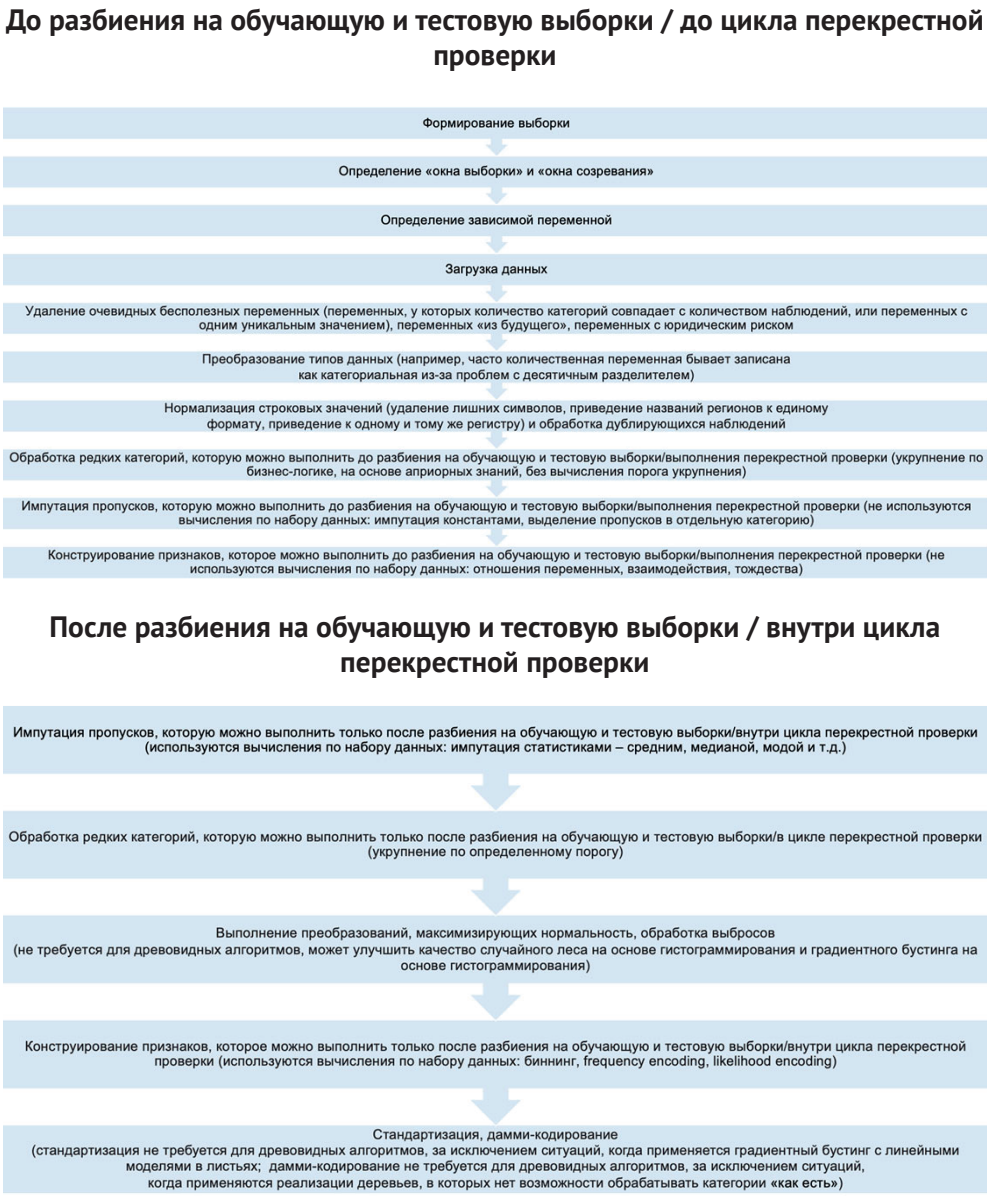


Рис. 1 План предварительной подготовки данных

2. Формирование выборки

2.1. Генеральная и выборочная совокупности

Генеральная совокупность, *или* **популяция (population)**, – совокупность всех объектов (единиц), относительно которых предполагается делать выводы при изучении конкретной задачи.

Генеральная совокупность состоит из всех объектов, которые имеют качества, свойства, интересующие исследователя. Например, в исследованиях телезрительской аудитории, проводимых компанией Mediascope, генеральной совокупностью будет население России в возрасте 4 лет и старше, проживающее в городах от 100 000 и более. А в исследованиях читательской аудитории, проводимых этой же компанией, генеральной совокупностью будет уже население России в возрасте 16 лет и старше, проживающее в городах от 100 000 и более. В исследованиях политических предпочтений в преддверии президентских выборов генеральной совокупностью будет население России в возрасте 18 лет и старше (поскольку право голосовать гражданин получает с 18 лет). В банковском скоринге генеральной совокупностью считаются все потенциально возможные заемщики банка. В таких случаях принято считать, что объем генеральной совокупности у нас неизвестен.

Выборка, *или* **выборочная совокупность (sample)**, – набор объектов, выбранных с помощью определенной процедуры из генеральной совокупности для участия в исследовании.

Цель проведения выборочных обследований – на основе выборки сформировать суждение обо всей генеральной совокупности.

Допустим, нам необходимо провести исследование, цель которого – убедиться, что груши на дереве созрели. Решение заключается в том, чтобы сорвать несколько груш с дерева и попробовать их. Генеральная совокупность – все груши на дереве. Выборочная совокупность – сорванные груши с дерева. Если все сорванные груши созрели, то с большой вероятностью можно сделать вывод, что и все остальные груши на дереве тоже созрели. Если необходимо узнать, все ли груши созрели во всем саду, то это будет уже другая генеральная совокупность – груши во всем саду. Исследование будет состоять в том, чтобы срывать и пробовать груши с разных деревьев.

2.2. Характеристики выборки

Перечень всех единиц наблюдения генеральной совокупности с базовой информацией представляет **основу выборки**. **Базовая информация** – набор характеристик, известных до проведения обследования для каждого элемента основы выборки (например, фамилия, имя и отчество респондента, адрес предприятия, регион проведения интервью и другие характеристики).

Элементы отбора при формировании выборочной совокупности называются **единицами отбора**. Объект, признаки которого подлежат регистрации, называется **единицей наблюдения**. Обычно единицей наблюдения в социологических опросах является конкретный человек, который будет отвечать на вопрос анкеты. Единица наблюдения может совпадать или не совпадать с единицей

отбора. При простой случайной выборке единицы отбора и единицы наблюдения совпадают. В случае использования многоступенчатой выборки сначала отбираются регионы, потом населенные пункты, затем предприятия или адреса проживания семей (все они и будут единицами отбора), и лишь на последнем этапе будут отобраны конкретные единицы наблюдения – респонденты.

Количество элементов выборки называется **объемом** (**размером**) выборки.

Соответствие характеристик выборки характеристикам популяции или генеральной совокупности в целом называется **репрезентативностью**. Репрезентативность определяет, насколько возможно обобщать результаты исследования с привлечением определенной выборки на всю генеральную совокупность, из которой она была отобрана. Корректный вывод обо всей генеральной совокупности можно сделать только на основании репрезентативной выборки. Поэтому при формировании выборки должен быть такой отбор элементов, чтобы выборка была репрезентативной.

В США одним из наиболее известных исторических примеров нерепрезентативной выборки считается случай, произошедший во время президентских выборов в 1936 году.

Журнал «Литрери Дайджест», успешно прогнозировавший события нескольких предшествующих выборов, ошибся в своих предсказаниях, разослав десять миллионов пробных бюллетеней своим подписчикам, а также людям, выбранным по телефонным книгам всей страны и людям из регистрационных списков автомобилей. В 25 % вернувшихся бюллетеней (почти 2,5 миллиона) голоса были распределены следующим образом:

57 % отдавали предпочтение кандидату-республиканцу Альфу Лэндону;

40 % выбрали действующего в то время президента-демократа Франклина Рузвельта.

На действительных же выборах, как известно, победил Рузвельт, набрав более 60 % голосов. Ошибка «Литрери Дайджест» заключалась в следующем: желая увеличить репрезентативность выборки, – так как им было известно, что большинство их подписчиков считают себя республиканцами, – они расширили выборку за счёт людей, выбранных из телефонных книг и регистрационных списков. Однако они не учли современных им реалий и в действительности набрали ещё больше республиканцев: во время Великой депрессии обладать телефонами и автомобилями могли себе позволить в основном представители среднего и высшего класса (то есть большинство республиканцев, а не демократов).

В нашем игрушечном примере, когда нам нужно было убедиться, что груши на дереве созрели, примером нерепрезентативной выборки были бы груши, сорванные только с одной, южной стороны дерева. А если бы нам необходимо было узнать, все ли груши созрели во всем саду, то примером нерепрезентативной выборки были бы груши, сорванные с деревьев, которые росли поблизости (допустим, мы поленились пройти в глубь сада).

Отклонение результатов оценки значений, полученных с помощью выборки, от истинных неизвестных значений в генеральной совокупности называется **ошибкой выборки**.

В выборочных обследованиях мы будем оперировать статистиками. **Статистика** – это некоторая функция от выборочных наблюдений, например минимальное значение, среднее арифметическое, стандартное отклонение и др. Допустим, минимальный вес груши, средний вес груши.

Исследование всех объектов генеральной совокупности называется **сплошным обследованием**. Наиболее точные оценки могут быть получены при сплошном наблюдении, однако могут быть сложности. Основные проблемы, возникающие при сплошном наблюдении: ограничение по времени, ограничение финансовых ресурсов, ограничение человеческих ресурсов (здесь речь идет о физических и интеллектуальных ресурсах как опрашивающих, так и опрошенных).

Понятно, что мы не можем для получения рейтингов кандидатов на пост Президента РФ физически опросить все население России в возрасте от 18 лет и старше. Однако даже если сплошное обследование можно организовать, оно не гарантирует получения надежных результатов. Примером, когда сплошное обследование потерпело неудачу, была сплошная перепись населения России 1897 г. Когда анализировалась численность населения по возрастам, то получалось, что максимальные численности (пики) имели возрасты, кратные 5 и в особенности кратные 10. Большая часть населения в те времена была неграмотна и свой возраст помнила только приблизительно, с точностью до пяти или до десяти лет. Чтобы все-таки узнать, каково было распределение по возрастам на самом деле, нужно было не увеличивать объем данных, а, наоборот, создать выборку из нескольких процентов населения и провести комплексное исследование, основанное на перекрестном анализе нескольких источников: документов, свидетельств и личных показаний. Это дало бы гораздо более точную картину, нежели сплошная перепись. Для решения проблем, возникающих при сплошном обследовании, как раз и используют выборочные обследования.

2.3. Детерминированные и вероятностные выборки

По способу отбора выборки делятся на:

- **детерминированные**;
- **вероятностные**.

Детерминированный отбор – выборочный метод, в котором не применяется процедура случайного отбора единиц генеральной совокупности. Этот метод основан на индивидуальных суждениях исследователя. Примерами являются экспертный отбор, квотный отбор, отбор методом «снежного кома».

Выборка по методу «снежного кома» строится следующим образом. У каждого респондента, начиная с первого, просятся контакты его друзей, коллег, знакомых, которые подходили бы под условия отбора и могли бы принять участие в исследовании. Таким образом, за исключением первого шага, выборка формируется с участием самих объектов исследования. Метод часто применяется, когда необходимо найти и опросить труднодоступные группы респондентов (например, респондентов, имеющих высокий доход, респондентов, принадлежащих к одной профессиональной группе, респондентов, имеющих какие-либо схожие хобби/увлечения и т. д.).

При квотной выборке генеральная совокупность сначала разделяется на непересекающиеся группы. Затем пропорционально из каждой группы выбираются единицы наблюдения на основании предпочтений отбирающего. Например, интервьюер может получить задание отобрать 200 женщин и 300 мужчин возрастом от 45 до 60 лет. Это значит, что внутри каждой квоты интервьюер отбирает респондентов по своим предпочтениям.

Описанный второй шаг формирования квотной выборки относит её к детерминированному типу. Отбор элементов в квотную выборку не является случайным и может быть ненадёжным. Например, интервьюеры могут в первую очередь пытаться опрашивать тех людей, которые выглядят наиболее отзывчивыми или живут поблизости. Соответственно, менее отзывчивые люди или репонденты, живущие в труднодоступных местах, криминогенных районах, в которых интервьер побоится опрашивать, имеют меньше шансов попасть в выборку.

Квотная выборка полезна, когда время ограничено, отсутствует основа для формирования вероятностной выборки, бюджет исследования небольшой или когда точность результатов не слишком важна.

Вероятностный отбор – выборочный метод, в котором состав выборки формируется случайным образом. В вероятностном отборе каждая единица генеральной совокупности имеет определенную вероятность включения в выборку. Нас будут интересовать вероятностные выборочные методы.

2.4. Виды, методы и способы вероятностного отбора

По виду отбора различают следующие вероятностные выборки:

- выборки с индивидуальным отбором;
- выборки с групповым отбором;
- выборки с комбинированным отбором.

Выборки с индивидуальным отбором осуществляют отбор из генеральной совокупности каждой единицы наблюдения в отдельности. Например, при обследовании удовлетворенности сотрудников предприятия размером заработной платы осуществляется отбор сотрудников.

Выборки с групповым отбором осуществляют отбор групп единиц. Например, при обследовании удовлетворенности сотрудников предприятия размером заработной платы осуществляется отбор отделов предприятия.

По методу отбора различают:

- выборки без возвращения (бесповторный отбор);
- выборки с возвращением (повторный отбор).

В выборках без возвращения (бесповторный отбор) отобранный элемент не возвращается в генеральную совокупность, из которой осуществлялся отбор. В выборках с возвращением (повторный отбор) отобранный объект возвращается в генеральную совокупность и имеет шанс быть отобранным повторно. Использование повторного метода дает бóльшую ошибку выборки, чем использование бесповторного.

По способам отбора различают:

- простой случайный отбор;
- систематический отбор;
- вероятностно-пропорциональный отбор;
- расслоенный случайный отбор;
- кластерный (серийный) отбор.

Более подробное обсуждение этих способов выходит за рамки книги, разберем здесь лишь процедуру простого случайного отбора.

При проведении простого случайного отбора каждая единица генеральной совокупности имеет известную и равную вероятность отбора. В простом случайном отборе каждая единица отбирается независимо от другой. Для отбора используется таблица случайных чисел или компьютерная программа.

Здесь отметим, что к повторному отбору приравнивается простой случайный отбор из генеральной совокупности, объем которой неизвестен. При вычислении необходимого объема выборки для построения моделей банковского скоринга как раз предполагают, что имеет место повторный отбор.

2.5. Подходы к определению необходимого объема выборки

Необходимый объем выборки может быть известен по результатам предыдущих аналогичных исследований. Если же объем выборки неизвестен, его необходимо рассчитать.

2.5.1. Определение объема выборки согласно теории выборочных обследований

Согласно теории выборочных обследований объем необходимой выборки зависит от задаваемой точности оценки параметров, дисперсии оцениваемых параметров и способа отбора. Общее правило следующее: чем больше дисперсия оцениваемых параметров, тем больший объем выборки необходим для того, чтобы обеспечить требуемую точность. Поэтому предварительно по отобранным данным необходимо рассчитать дисперсию оцениваемых переменных. В зависимости от величины надежности выбирают значение стандартного нормального распределения.

В банковском скоринге для построения качественной модели данные о «хороших» и «плохих» клиентах максимально должны отражать поток клиентов с улицы.

Предположим, мы хотим быть уверенными на 95 %, что соотношение «хороших» и «плохих» заемщиков в обучающей выборке отражает генеральную совокупность заемщиков. В таких случаях обычно используют следующую формулу определения объема выборки для оценки генеральной доли при повторном случайном отборе (при этом предполагается, что выборка значительно меньше генеральной совокупности):

$$n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2},$$

где:

n – минимальный объем выборки;

z_γ – значение стандартного нормального распределения, определяемое в зависимости от выбранного доверительного уровня (доверительной вероятности);

w – доля «плохих» на предварительной выборке (может быть получена, исходя из опыта имеющихся априорных знаний);

Δ_w – максимально допустимая предельная ошибка оценки доли «плохих» заемщиков (предельная ошибка выборки).

Доверительный уровень (доверительная вероятность) – это вероятность того, что генеральная доля лежит в границах полученного доверительного интервала: выборочная доля (w) ± ошибка выборки (Δ_w). Доверительный уровень устанавливает сам исследователь в соответствии со своими требованиями к надежности полученных результатов. Чаще всего применяются доверительные уровни, равные 0,95 или 0,99.

Допустим, среди 700 клиентов предварительной выборки 50 оказались «плохими». Оценка доли «плохих» клиентов, по имеющимся данным, для построения модели составила около 0,07, или 7 %. При таком значении оценки доли предположим, мы хотим ошибиться не более чем на 10 %, что будет соответствовать допустимой предельной ошибке оценки доли 0,007, т. е. (50 / 700) * 0,1. При этом задаем 95%-ную доверительную вероятность. В этом случае z-значение стандартного нормального закона распределения составит около 1,96. Вычисляем минимальный объем выборки: $n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,07 \cdot 0,93}{0,000049} = 5102.$

На практике чаще всего нет возможности сформировать предварительную выборку и значение w неизвестно. В таком случае w принимается за 0,5 (самый консервативный сценарий). При этом значении размер ошибки выборки будет максимален.

Допустим, оценка доли «плохих» клиентов неизвестна, принимаем ее за 0,5. При таком значении оценки доли предположим, мы хотим ошибиться не более чем на 10 %. При этом задаем 95%-ную доверительную вероятность. Получаем

$$n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,5 \cdot 0,5}{0,000049} = 19592.$$

На собеседованиях часто просят рассчитать объем выборки с определенной предельной ошибкой. Например, рассчитайте объем выборки, предельная ошибка которой составит 4 %. При этом мы принимаем, что доверительный уровень равен 95 %, а генеральная совокупность значительно больше выборки.

Применяем знакомую формулу $n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,5 \cdot 0,5}{0,0016} = 600.$

При этом не забывайте, вам помимо обучающей выборки еще нужно отложить наблюдения для проверки, здесь, соответственно, нужно выделить ситуацию, когда вы строите и проверяете базовую модель (вам нужен тестовый набор), и ситуацию, когда вы строите модели, настраивая гиперпараметры (вам нужен валидационный набор для настройки гиперпараметров, роль валидационного набора могут выполнять проверочные блоки перекрестной проверки и тестовый набор для итоговой оценки качества).

Когда предполагается, что выборка сопоставима с генеральной совокупностью (обычное явление при опросах организаций в B2B-исследованиях), формула определения объема выборки для оценки генеральной доли будет выглядеть несколько иначе, в ней будет фигурировать показатель объема генеральной совокупности.

$$n = \frac{\frac{z_\gamma^2 w(1-w)}{\Delta_w^2}}{1 + \frac{\frac{z_\gamma^2 w(1-w)}{\Delta_w^2} - 1}{N}}.$$

2.5.2. Определение объема выборки согласно правилу NEPV

В практике банковского скоринга для ответа на вопрос об объеме выборки часто используют правило «Number of Events Per Variable» (количество событий на одну переменную, NEPV), сформулированное Фрэнком Харреллом.

Для задачи бинарной классификации оно связывает минимальный объем выборки с количеством «событий» – наблюдений в миноритарной (наименьшей по размеру) категории зависимой переменной и количеством признаков, поданным на вход модели. Согласно этому правилу, необходимо взять количество наблюдений в обучающей выборке, относящихся к миноритарной категории зависимой переменной (в кредитном скоринге это «плохие» заемщики). Это число наблюдений нужно разделить на количество заданных признаков. Для логистической регрессии на один параметр должно приходиться не менее 20 событий, при построении дерева решений CHAID на один признак должно приходиться не менее 50 событий, а для модели случайного леса, градиентного бустинга, SVM и нейронной сети на одну независимую переменную должно приходиться не менее 200 событий.

Для задачи регрессии мы просто берем количество наблюдений и делим на количество признаков, и для линейной регрессии на один параметр должно приходиться не менее 20 наблюдений, для дерева решений CHAID (в тех случаях, когда реализация алгоритма позволяет решать задачу регрессии) на один признак должно приходиться не менее 50 наблюдений, для случайного леса и других сложных моделей на один признак должно приходиться не менее 200 наблюдений. По мнению Фрэнка Харрелла, для дерева решений CART правило NEPV невозможно сформулировать из-за высокой нестабильности метода и склонности к переобучению.

Если правило выполняется для обучающей выборки, то объем выборки для обучения является достаточным. В противном случае необходимо либо увеличить объем выборки, либо сократить количество признаков, подаваемых на вход модели. Затем вся та же самая процедура применяется к тестовой выборке, если правило выполняется, объем выборки для проверки достаточен.

Мы решаем задачу классификации, и у нас есть общая выборка объемом 4424 клиента, классифицированных на два класса: класс *Остается* (2492 клиента) и класс *Уходит* (1932 клиента). Мы разбили выборку на обучающую и тестовую и получили следующее распределение классов в выборках: *Остается* (1746 клиентов) и *Уходит* (1334 клиента).

Выясняем, достаточен ли объем выборки для обучения. У нас 1746 оставшихся клиентов, 1334 ушедших клиента и 9 независимых переменных. Ми-

норитарный класс – класс *Уходит*. Проверяя выполнение правила NEPV, мы получаем 1334 / 9 = 148,2. Наша выборка обеспечивает достаточное количество событий на одну переменную, и мы можем использовать эту выборку для обучения.

Выясняем, достаточен ли объем выборки для проверки. У нас 746 оставшихся клиентов, 598 ушедших клиентов. Проверяя выполнение правила NEPV, мы получаем 598 / 9 = 66,4. Наша выборка обеспечивает достаточное количество событий на одну переменную, и мы можем использовать эту выборку для проверки.

Если выполняется перекрестная проверка, роль обучающей выборки выполняет набор обучающих блоков, а роль тестовой выборки выполняет тестовый блок.

Опять же напомним: данная схема работает для построения базовой модели без подбора гиперпараметров.

2.5.3. Определение объема выборки с помощью кривых обучения и валидации

Кроме того, необходимый объем выборки можно определить с помощью кривых обучения и валидации. Их можно построить с помощью функции `learning_curve()`. Она запускает перекрестную проверку на наборах данных разного объема. Генератор перекрестной проверки разбивает весь набор данных *k* раз на обучающую выборку и тестовую выборку. В итоге для набора соответствующего размера мы получаем метрику для обучающей выборки, усредненную по *k* проходам, и метрику для тестовой выборки, усредненную по *k* проходам.

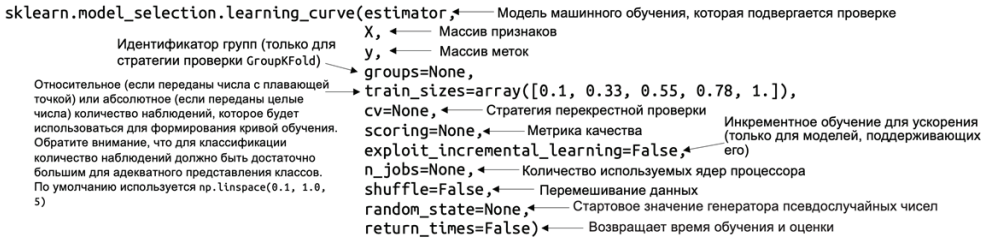


Рис. 2 Параметры функции learning_curve()

В результате функция `learning_curve()` возвращает:

- `train_sizes_abs` – количество наблюдений, использованное для построения кривой обучения;
- `train_scores` – значения метрики на обучающих выборках перекрестной проверки;
- `test_scores` – значения метрики на тестовых выборках перекрестной проверки;
- `fit_times` – время, затраченное на обучение, в секундах;
- `score_times` – время, затраченное на оценку качества, в секундах.

Давайте загрузим необходимые библиотеки, классы и функции.

```
# импортируем библиотеки, классы и функции
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
```

Теперь загрузим данные. Данные записаны в файле *Response.csv*. Исходная выборка содержит записи о 30 259 клиентах, классифицированных на два класса: 0 – отклика нет (17 170 клиентов) и 1 – отклик есть (13 089 клиентов).

По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- категориальный признак *Ипотечный кредит [mortgage]*;
- категориальный признак *Страхование жизни [life_ins]*;
- категориальный признак *Кредитная карта [cre_card]*;
- категориальный признак *Дебетовая карта [deb_card]*;
- категориальный признак *Мобильный банк [mob_bank]*;
- категориальный признак *Текущий счет [curr_acc]*;
- категориальный признак *Интернет-доступ к счету [internet]*;
- категориальный признак *Индивидуальный займ [perloan]*;
- категориальный признак *Наличие сбережений [savings]*;
- категориальный признак *Пользование банкоматом за последнюю неделю [atm_user]*;
- категориальный признак *Пользование услугами онлайн-маркетплейса за последний месяц [markpl]*;
- количественный признак *Возраст [age]*;
- количественный признак *Давность клиентской истории [cus_leng]*;
- категориальная зависимая переменная *Отклик на предложение новой карты [response]*.

```
# загружаем данные
data = pd.read_csv('Data/Response.csv', sep=';')
data.head(3)
```

	mortgage	life_ins	cre_card	deb_card	mob_bank	curr_acc	internet	perloan	savings	atm_user	markpl	age	cus_leng	response
0	No	No	No	No	No	No	No	No	No	No	No	18.0	less than 3 years	No
1	Yes	Yes	NaN	NaN	Yes	No	NaN	NaN	NaN	Yes	No	18.0	NaN	Yes
2	Yes	Yes	NaN	Yes	No	No	No	No	No	No	Yes	NaN	from 3 to 7 years	Yes

Формируем массив меток и массив признаков, создаем списки переменных и трансформеры, с помощью класса `ColumnTransformer` сопоставляем транс-

формеры со списками переменных, создаем конвейер для логистической регрессии и конвейер для градиентного бустинга, каждой модели машинного обучения будет соответствовать своя последовательность моделей предварительной подготовки данных.

```
# создаем массив меток и массив признаков
y = data.pop('response')

# создаем списки категориальных
# и количественных столбцов
categorical_features = data.select_dtypes(
    include='object').columns.tolist()
numeric_features = data.select_dtypes(
    exclude='object').columns.tolist()

# создаем трансформеры
numeric_transformer_logreg = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

numeric_transformer_boost = Pipeline([
    ('imputer', SimpleImputer(strategy='median'))
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant',
                              fill_value='missing')),
    ('onehot', OneHotEncoder(sparse=False,
                             handle_unknown='ignore'))
])

# сопоставляем трансформеры спискам переменных
# для логистической регрессии
preprocessor_logreg = ColumnTransformer([
    ('num', numeric_transformer_logreg, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# сопоставляем трансформеры спискам переменных
# для градиентного бустинга
preprocessor_boost = ColumnTransformer([
    ('num', numeric_transformer_boost, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# формируем итоговый конвейер
pipe_logreg = Pipeline([
    ('preprocessor', preprocessor_logreg),
    ('logreg', LogisticRegression(solver='lbfgs', max_iter=400))
])

pipe_boost = Pipeline([
    ('preprocessor', preprocessor_boost),
                     ('boost', GradientBoostingClassifier(
                         random_state=42))])
```

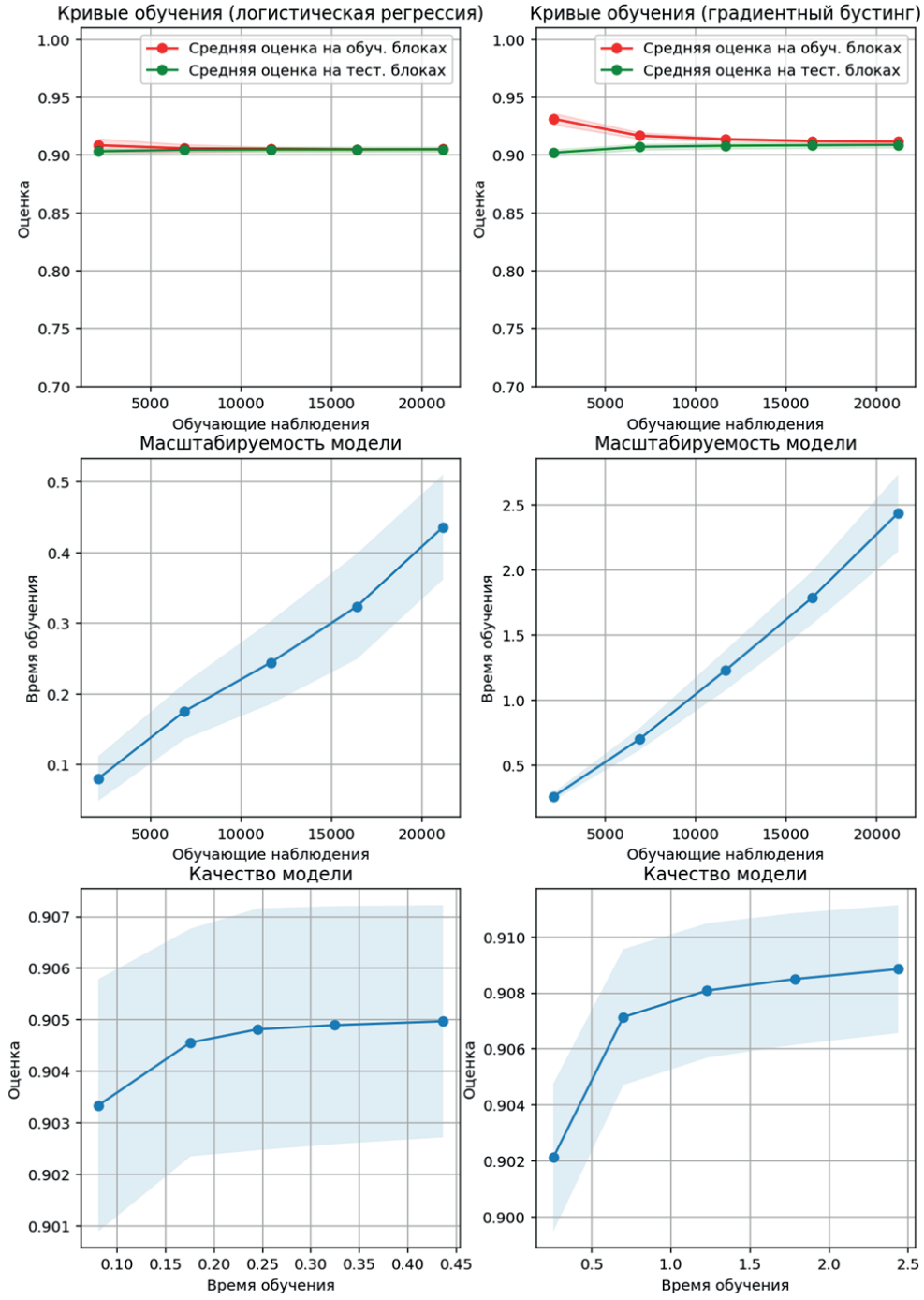
Теперь пишем функцию, которая будет строить графики на основе результатов, возвращенных функцией `learning_curve()`.

```
# пишем функцию, которая строит графики
# по результатам функции learning_curve()
def plot_learning_curve(estimator,
                        title,
                        X,
                        y,
                        axes=None,
                        ylim=None,
                        cv=None,
                        n_jobs=None,
                        train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Строит 3 графика: кривые обучения и валидации, кривую зависимости между
    объемом обучающих данных и временем обучения, кривую зависимости между
    временем обучения и оценкой качества.

    Параметры
    ----------
    estimator : модель машинного обучения для проверки.
    title : заголовок диаграммы.
    X : массив признаков.
    y : массив меток.
    axes : задаем область рисования (Axes) для построения кривых.
    ylim : задает минимальное и максимальное значения по оси y.
    cv : стратегия перекрестной проверки.
    n_jobs : количество используемых ядер процессора.
    train_sizes : абсолютное или относительное количество наблюдений.
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))
    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Обучающие наблюдения")
    axes[0].set_ylabel("Оценка")
    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv,
                       scoring='roc_auc', n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # строим кривые обучения и валидации
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
```

```
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Средняя оценка на обуч. блоках")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Средняя оценка на тест. блоках")
    axes[0].legend(loc="best")

    # строим график зависимости между объемом
    # обучающих данных и временем обучения
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Обучающие наблюдения")
    axes[1].set_ylabel("Время обучения")
    axes[1].set_title("Масштабируемость модели")
    # строим график зависимости между временем
    # обучения и оценкой качества
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("Время обучения")
    axes[2].set_ylabel("Оценка")
    axes[2].set_title("Качество модели")
    return plt
```

А сейчас будем строить графики кривых обучения и валидации.

```
# задаем сетку и размеры графиков
fig, axes = plt.subplots(3, 2, figsize=(10, 15))
# задаем стратегию перекрестной проверки
cv = ShuffleSplit(n_splits=20, test_size=0.3, random_state=42)

# задаем заголовок
title = "Кривые обучения (логистическая регрессия)"
# строим графики для логистической регрессии
plot_learning_curve(pipe_logreg, title, data, y, axes=axes[:, 0],
                    ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

# задаем заголовок
title = "Кривые обучения (градиентный бустинг)"
# строим графики для градиентного бустинга
plot_learning_curve(pipe_boost, title, data, y, axes=axes[:, 1],
                    ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

# выводим графики
plt.show()
```



Для каждой модели машинного обучения мы выводим по три графика: кривые обучения и валидации, кривую зависимости между объемом обучающих

данных и временем обучения, кривую зависимости между временем обучения и оценкой качества.

График кривых обучения и валидации для логистической регрессии показывает, что независимо от размера обучающего набора оценка на обучении практически совпадает с оценкой на тесте.

График кривых обучения и валидации для градиентного бустинга показывает, что по мере увеличения размера обучающего набора оценка на обучении приближается к оценке на тесте (уменьшается гэп между ними), т. е. происходит уменьшение переобучения. При этом мы видим, что обе модели практически эквивалентны по качеству (как метрика качества у нас используется AUC) и увеличения оценки на тесте по мере роста объема данных практически не происходит, что может говорить о достаточном объеме данных, и, скорее всего, улучшения качества можно добиться не за счет увеличения данных, а за счет тщательно продуманного конструирования признаков.

Графики кривой зависимости между объемом обучающих данных и временем обучения показывают, что обучение градиентного бустинга занимает больше времени. В том случае, когда мы используем набор размером 5000 наблюдений, обучение логистической регрессии занимает в среднем 0,12 секунды, а обучение градиентного бустинга – 0,5 секунды.

Графики кривой зависимости между временем обучения и оценкой качества показывают, сколько времени обучения требуется для получения соответствующего качества на тесте. Например, мы можем принять, что разница между AUC 0,905 и AUC 0,908 не столь критична, чтобы обучаться 2,5 секунды вместо 0,4 секунды.

Теперь возьмем другой набор данных. Данные записаны в файле *StateFarm_missing.csv*. Исходная выборка содержит записи о 8293 клиентах, классифицированных на два класса: 0 – отклика нет на предложение автостраховки (7462 клиента) и 1 – отклик есть на предложение автостраховки (831 клиент). По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

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

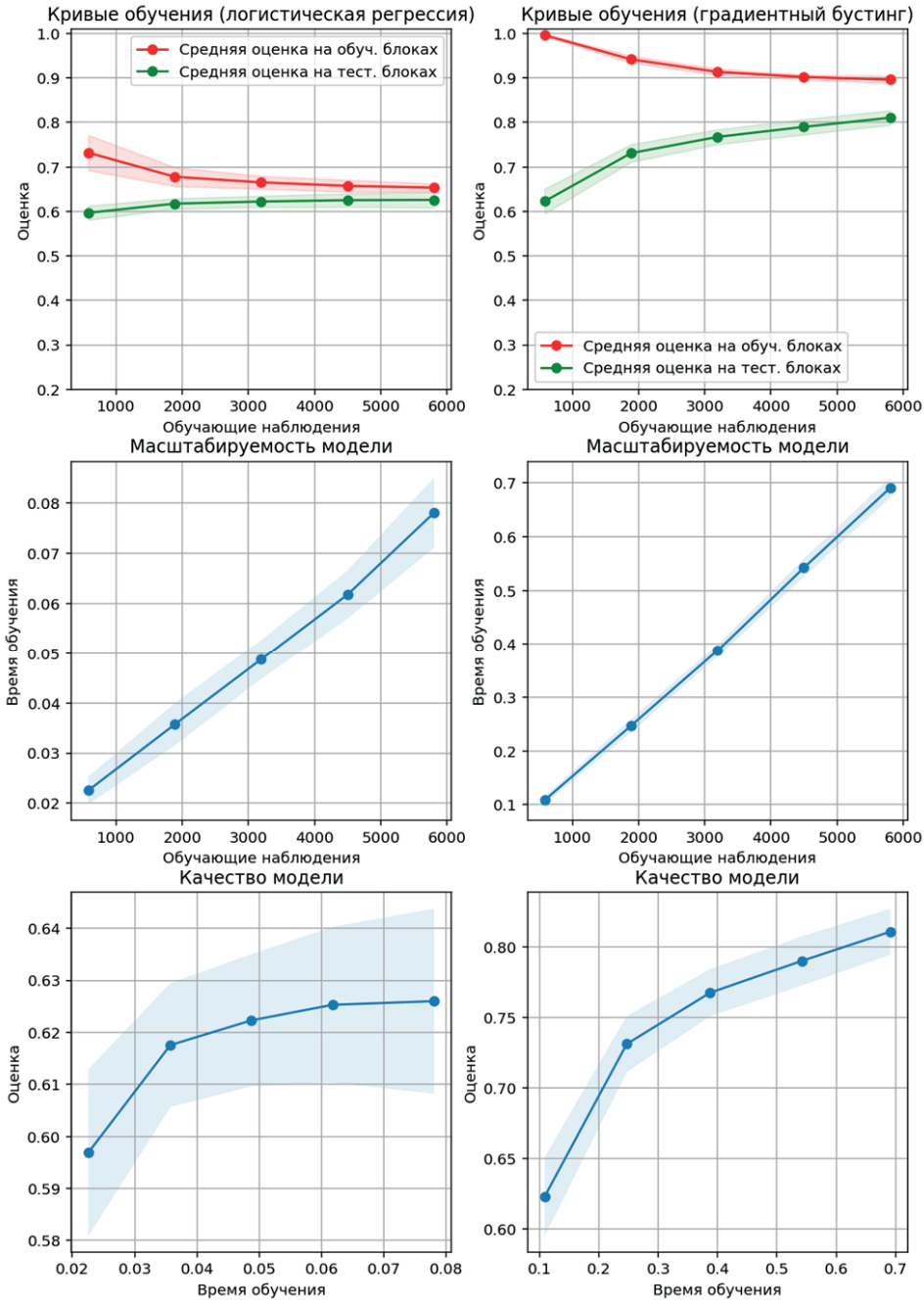
Давайте загрузим данные, опять создадим конвейеры и построим графики кривых обучения и валидации.

```
# загружаем данные
data = pd.read_csv('Data/StateFarm_missing.csv', sep=';')
data.head(3)
```

	Customer Lifetime Value	Coverage	Education	EmploymentStatus	Gender	Income	Monthly Premium Auto	Months Since Last Claim	Months Since Policy Inception	Number of Open Complaints	Number of Policies	Response
0	2763.519279	Basic	Bachelor	Employed	F	56274.0	NaN	32.0	5.0	NaN	1.0	No
1	NaN	NaN	Bachelor	Unemployed	F	0.0	NaN	13.0	42.0	NaN	NaN	No
2	NaN	NaN	NaN	Employed	F	48767.0	108.0	NaN	38.0	0.0	NaN	No

```
# формируем массив меток и массив признаков
y = data.pop('Response')

# создаем списки количественных
# и категориальных столбцов
cat_features = data.select_dtypes(
    include='object').columns.tolist()
num_features = data.select_dtypes(
    exclude='object').columns.tolist()

# сопоставляем трансформеры спискам переменных
# для логистической регрессии
preprocessor_lr = ColumnTransformer([
    ('num', numeric_transformer_logreg, num_features),
    ('cat', categorical_transformer, cat_features)
])

# сопоставляем трансформеры спискам переменных
# для градиентного бустинга
preprocessor_bst = ColumnTransformer([
    ('num', numeric_transformer_boost, num_features),
    ('cat', categorical_transformer, cat_features)
])

# формируем итоговый конвейер
pipe_lr = Pipeline([
    ('preprocessor', preprocessor_lr),
    ('logreg', LogisticRegression(solver='lbfgs',
                                  max_iter=400))
])

pipe_bst = Pipeline([
    ('preprocessor', preprocessor_bst),
    ('boost', GradientBoostingClassifier(
        random_state=42))
])

# задаем сетку и размеры графиков
fig, axes = plt.subplots(3, 2, figsize=(10, 15))

# задаем заголовок
title = "Кривые обучения (логистическая регрессия)"
# строим графики для логистической регрессии
plot_learning_curve(pipe_lr, title, data, y, axes=axes[:, 0],
                    ylim=(0.2, 1.01),
                    cv=cv, n_jobs=4)

# задаем заголовок
title = "Кривые обучения (градиентный бустинг)"
# строим графики для градиентного бустинга
```

```
plot_learning_curve(pipe_bst, title, data, y, axes=axes[:, 1],
                    ylim=(0.2, 1.01),
                    cv=cv, n_jobs=4)

# выводим графики
plt.show()
```

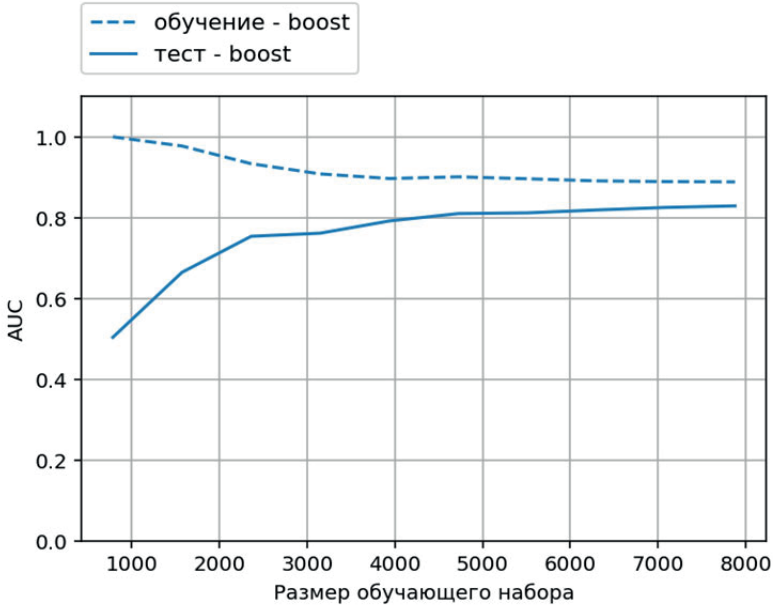


Вновь для каждой модели машинного обучения мы выводим по три графика: кривые обучения и валидации, кривую зависимости между объемом обучающих данных и временем обучения, кривую зависимости между временем обучения и оценкой качества.

Здесь уже графики кривых обучения и валидации для логистической регрессии и бустинга показывают, что по мере увеличения размера обучающего набора оценка на обучении сближается с оценкой на тесте. При этом если для логистической регрессии оценка на обучении практически сближается с оценкой на тесте, то для градиентного бустинга гэп хоть и уменьшается, но все еще остается значительным, что может говорить о недостаточном объеме данных, и, скорее всего, улучшения качества можно добиться за счет увеличения данных. При этом мы видим, что с точки зрения оценки на тесте модель градиентного бустинга существенно лучше модели логистической регрессии.

На практике часто пользуются упрощенной версией функции `plot_learning_curve_simple()`.

```
# импортируем класс KFold
from sklearn.model_selection import KFold

# пишем упрощенную версию plot_learning_curve()
def plot_learning_curve_simple(est, X, y):
    # получаем наборы для обучения, значения метрик
    training_set_size, train_scores, test_scores = learning_curve(
        est, X, y, train_sizes=np.linspace(.1, 1, 10), scoring='roc_auc',
        cv=KFold(20, shuffle=True, random_state=1))
    # извлекаем имя последнего этапа итогового конвейера -
    # название модели машинного обучения
    estimator_name = est.steps[-1][0]
    # строим кривые обучения и валидации
    line = plt.plot(training_set_size, train_scores.mean(axis=1), '--',
                    label="обучение - " + estimator_name)
    plt.plot(training_set_size, test_scores.mean(axis=1), '-',
             label="тест - " + estimator_name, c=line[0].get_color())
    # задаем координатную сетку
    plt.grid()
    # подписываем ось x
    plt.xlabel("Размер обучающего набора")
    # подписываем ось y
    plt.ylabel("AUC")
    # задаем пределы значений оси y
    plt.ylim(0, 1.1)
    # задаем расположение легенды
    plt.legend(loc=(0, 1.05), fontsize=11)
# применяем упрощенную версию plot_learning_curve()
plot_learning_curve_simple(pipe_bst, data, y)
```


обучение - boost
тест - boost
AUC
1.0
0.8
0.6
0.4
0.2
0.0
1000
2000
3000
4000
5000
6000
7000
8000
Размер обучающего набора

3. Определение «окна выборки» и «окна созревания»

Прогнозные модели разрабатываются, исходя из предположения «прошлое отражает будущее». На основе этого предположения мы анализируем поведение прошлых клиентов, чтобы спрогнозировать поведение будущих клиентов. Для того чтобы корректно выполнить этот анализ, нужно собрать необходимые данные о клиентах за определенный период времени, а затем осуществить мониторинг клиентов в течение другого определенного периода времени, оценив, были они «хорошими» или «плохими». Собранные данные (независимые переменные) наряду с соответствующей классификацией (зависимой переменной, которая принимает значение *Хороший* или *Плохой*) составят основу для разработки прогнозной модели. Ключевыми терминами здесь будут «окно выборки» и «окно созревания».

«Окно выборки» – это период времени, в течение которого те или иные клиенты отбираются для анализа (попадают в выборку).

«Окно созревания» – это период времени, в течение которого клиент, собственно говоря, имел возможность себя проявить, и мы присваиваем клиенту соответствующий класс зависимой переменной.

Допустим, сделано предположение, что новый клиент получил кредит в определенный период времени (например, 1 января 2014 г.). В некоторый момент времени в будущем (например, через 90 дней) нам нужно определить, был этот клиент «хорошим» или «плохим» (чтобы классифицировать поведение).

Если мы возьмем все кредиты, выданные в январе 2014 года, и посмотрим на их качество с момента открытия до декабря 2015 года, окном выборки будет январь 2014 года, а окном созревания – 24 месяца, период с января 2014 года по декабрь 2015 года.

В некоторых случаях, таких как мошенничество и банкротство, временной период уже известен или предопределен. Но тем не менее вышеописанный анализ полезно выполнить для того, чтобы определить идеальное окно созревания.

Мы можем попробовать несколько подходов к определению окна выборки и окна созревания.

В ряде случаев окно созревания определяется требованиями регулирующих органов, т. е. горизонтом прогнозирования модели. Например, Базель II требует 12-месячное окно созревания, поэтому вероятность дефолта в моделях, построенных в соответствии с требованиями Базель II, определяется по 12-месячному окну созревания. Более поздние инициативы, такие как МСФО (IFRS) 9.3, предлагают использовать более длительный горизонт прогнозирования убытков, вплоть до срока действия кредита.

Второй подход сопоставляет окно созревания со сроком кредита. Например, если срок автокредита составляет четыре года, оценка заявок по этому кредиту должна основываться на четырехлетнем окне созревания. Логично, что отношения, в которые вступает кредитор, продолжаются четыре года, поэтому риск должен оцениваться в течение четырехлетнего периода. Этот подход хорошо работает для срочных кредитов. Если срок займа очень большой (скажем, более 8–10 лет), то можно применить третий подход, основанный на опреде-

лении окна созревания по данным винтажного анализа, чтобы получить более короткое окно созревания и использовать более свежие данные.

Для револьверных кредитов (например, возобновляемых кредитных карт) и кредитов с длительным сроком (например, ипотека) лучше рассмотреть третий подход. Мы берем ежемесячные или ежеквартальные отчеты по когортному или винтажному анализу, имеющиеся в любом отделе кредитных рисков, анализируем динамику по платежам или просрочкам и строим график появления «плохих» случаев (случаев просрочки 90+, отказа от услуг) с течением времени.

Классический пример винтажного анализа для просроченной задолженности свыше 90 дней и 12-квартального (3-летнего) окна созревания приведен на рисунке ниже. Данные, выделенные жирным шрифтом, показывают текущий статус просрочки платежа на определенный отчетный период времени.

«Плохой» = 90 дней просрочки	Срок кредита (в кварталах)											
Дата открытия кредита	1 Qtr	2 Qtr	3 Qtr	4 Qtr	5 Qtr	6 Qtr	7 Qtr	8 Qtr	9 Qtr	10 Qtr	11 Qtr	12 Qtr
Q1 13	0.00%	0.05%	1.10%	2.40%	2.80%	3.20%	3.50%	3.70%	3.80%	3.85%	3.85%	**3.86%**
Q2 13	0.00%	0.06%	1.20%	2.30%	2.70%	3.00%	3.30%	3.50%	3.60%	3.60%	**3.60%**	
Q3 13	0.00%	0.03%	0.90%	2.80%	3.20%	3.60%	4.10%	4.30%	4.40%	**4.45%**		
Q4 13	0.00%	0.03%	1.00%	2.85%	3.20%	3.50%	3.80%	4.00%	**4.10%**			
Q1 14	0.00%	0.04%	1.00%	2.20%	2.40%	2.70%	2.90%	**4.10%**				
Q2 14	0.00%	0.05%	1.20%	2.50%	2.90%	3.30%	**3.50%**					
Q3 14	0.00%	0.04%	1.30%	2.60%	3.00%	**3.35%**						
Q4 14	0.00%	0.08%	1.40%	2.60%	**3.00%**							
Q1 15	0.00%	0.02%	0.09%	**2.20%**								
Q2 15	0.00%	0.08%	**1.50%**									
Q3 15	0.00%	**0.05%**										
Q4 15	**0.00%**											

Рис. 3 Пример когортного/винтажного анализа

Таблица имеет достаточно простую интерпретацию. Так, на первой строчке 2,8 % заемщиков, получивших кредит в первом квартале 2013 г., выпали в просрочку более 90 дней через 5 кварталов.

Несмотря на то что показатели просрочек схожи, мы видим, что в некоторых когортах показатели просрочек выше при одинаковой зрелости. Это нормальное явление, поскольку маркетинговые кампании, экономические циклы, изменения кредитной политики и другие факторы могут влиять на качество кредитов.

У нас есть несколько сценариев для построения кривой созревания просрочек по представленным данным.

Первый сценарий заключается в использовании значений по диагонали, выделенных жирным шрифтом. Здесь показаны самые последние показатели просроченной задолженности. В портфелях, винтажность которых может отличаться по качеству, это может привести к появлению кривых, которые не очень полезны (винтажная кривая не будет плавно расти, поскольку могут быть «провалы»), и, следовательно, это лучший вариант для продуктов, в которых качество заявителя и кредитного счета довольно стабильно, например ипотека.

В случае колебания показателей есть два дополнительных сценария, которые могут помочь сгладить числа и дать нам более реалистичную диаграмму роста

просрочек 90+ с течением времени. Например, мы можем использовать средние значения по последним четырем когортам, как показано овалами в рисунке, или мы можем выбрать одну отдельную когорту, например кредитные счета, открытые в первом квартале 2013 года, как показано прямоугольной рамкой.

«Плохой» = 90 дней просрочки	Срок кредита (в кварталах)											
Дата открытия кредита	1 Qtr	2 Qtr	3 Qtr	4 Qtr	5 Qtr	6 Qtr	7 Qtr	8 Qtr	9 Qtr	10 Qtr	11 Qtr	12 Qtr
Q1 13	0.00%	0.05%	1.10%	2.40%	2.80%	3.20%	3.50%	3.70%	3.80%	3.85%	3.85%	**3.86%**
Q2 13	0.00%	0.06%	1.20%	2.30%	2.70%	3.00%	3.30%	3.50%	3.60%	3.60%	**3.60%**	
Q3 13	0.00%	0.03%	0.90%	2.80%	3.20%	3.60%	4.10%	4.30%	4.40%	**4.45%**		
Q4 13	0.00%	0.03%	1.00%	2.85%	3.20%	3.50%	3.80%	4.00%	**4.10%**			
Q1 14	0.00%	0.04%	1.00%	2.20%	2.40%	2.70%	2.90%	**4.10%**				
Q2 14	0.00%	0.05%	1.20%	2.50%	2.90%	3.30%	**3.50%**					
Q3 14	0.00%	0.04%	1.30%	2.60%	3.00%	**3.35%**						
Q4 14	0.00%	0.08%	1.40%	2.60%	**3.00%**							
Q1 15	0.00%	0.02%	0.09%	**2.20%**								
Q2 15	0.00%	0.08%	**1.50%**									
Q3 15	0.00%	**0.05%**										
Q4 15	**0.00%**											

Рис. 4 Пример когортного/винтажного анализа: подходы к построению кривой созревания

Ниже показан график накопленного уровня просрочек 90+ для двух когорт: кредитных счетов, открытых в 1-м квартале 2013 года, и кредитных счетов, открытых в 1-м квартале 2014 года.

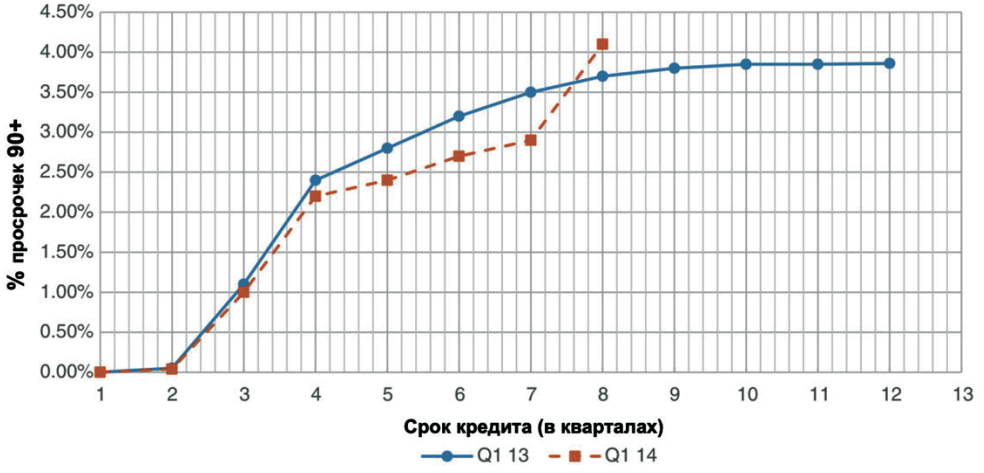


Рис. 5 Кривая винтажей

Перед нами пример типичного портфеля кредитных карт, где просрочки 90+ начинаются уже в самом начале и их количество растет с течением времени. Для счетов, открытых в первом квартале 2013 года, мы видим, что уровень просрочек 90+ быстро рос в первые несколько кварталов, а затем стабилизировался, когда срок кредитного счета стал приближаться к 10 кварталам. Для счетов, открытых во втором квартале 2014 года, мы видим, что уровень просрочек 90+ активно растет и о стабилизации говорить еще рано.

Мы сформируем выборку по такому периоду, когда можно считать, что уровень «плохих» случаев стабилизируется или когорта стала зрелой (т. е. когда накопленный уровень «плохих» случаев начинает выравниваться). В примере на рисунке выше хорошим окном выборки станут кредитные счета, которые были открыты 10–12 кварталов назад и дают 11-квартальное окно созревания.

Проектирование выборки по созревшей когорте осуществляется для того, чтобы минимизировать вероятность неправильной классификации клиентов (мы предоставляем всем клиентам достаточно времени, чтобы они могли стать «плохими») и убедиться в том, что определение «плохого» клиента, полученное по нашей выборке, не будет недооценивать итоговый ожидаемый процент «плохих» случаев. Например, если применительно к нашему примеру выборка будет спроектирована по кредитным счетам, открытым 4 квартала назад, мы увидим, что 2,4 % клиентов классифицированы как «плохие», однако уровень просрочек 90+ по-прежнему растет.

Поэтому некоторые кредитные счета, которые на самом деле являются «плохими», будут ошибочно помечены как «хорошие», если выборка будет спроектирована по 4-квартальному периоду.

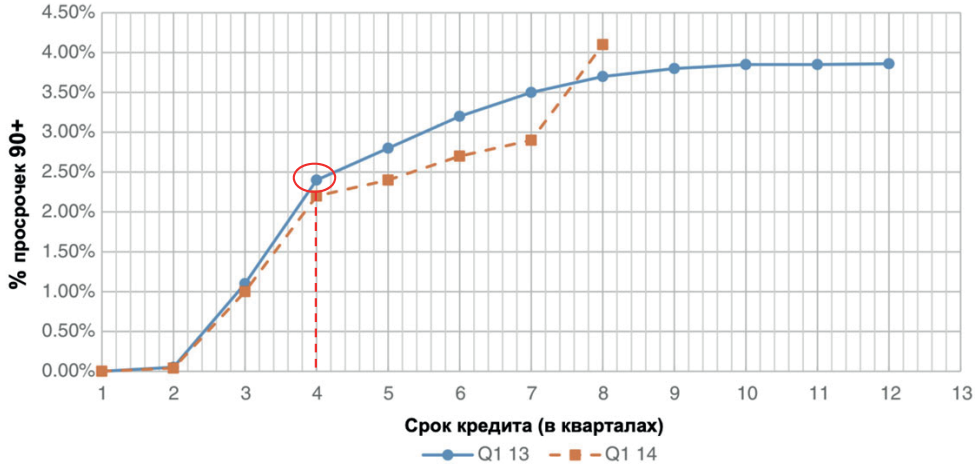


Рис. 6 Кривая винтажей: слишком короткое окно созревания

Временной горизонт «созревания» зависит от продукта, определения «плохого» клиента и конкурентной среды. Счета по кредитным картам считаются «зрелыми» после 12–18 месяцев, счета по автокредитам – обычно через 2–4 года, в то время как счета сроком от 4 до 5 лет считаются минимально допустимыми для разработки скоринговой карты по ипотечным кредитам. Поведенческие скоринговые карты предусматривают окно созревания из расчета 6–12 месяцев. Коллекторские модели строятся, как правило, на данных одного месяца, но все чаще компании строят такие карты для более коротких временных интервалов – до двух недель, чтобы облегчить разработку более подходящих способов взыскания долгов. По очевидным причинам счета по просрочкам 30+ будут становиться зрелыми быстрее, чем счета по просрочкам 90+. В условиях нестабильной макроэкономической ситуации, внутренних социально-экономических кризисов, как правило, счета по просрочкам также становятся зрелыми быстрее.

4. Определение зависимой переменной

Выбор зависимой переменной определяется целью построения прогнозной модели. Цели могут быть общими, например сокращение потерь по новым кредитным счетам, и конкретными, например сокращение числа дефолтов по одобренным заявкам в течение 6 месяцев после принятия положительного решения.

Зависимая переменная может быть количественной и категориальной.

В скоринге заявок зависимая переменная будет категориальной: погасит заемщик кредит («хороший») или не погасит («плохой»). Обычно к категории «плохой» относят клиентов, имеющих просроченную задолженность 90 дней и более. Этот период определяется требованиями банковского надзора. В соответствии с соглашением Базель II дефолт должника считается произошедшим, когда имело место одно или оба из следующих событий: банк считает, что должник не в состоянии полностью погасить свои кредитные обязательства перед банком без принятия банком таких мер, как реализация обеспечения (если таковое имеется); должник более чем на 90 дней просрочил погашение любых существенных кредитных обязательств перед банком.

Разумеется, банк может строить различные скоринговые карты с разными значениями зависимой переменной, вводя дополнительные критерии определения «плохого» и «хорошего» заемщика, а также меняя срок просрочки платежей. Примерами зависимой переменной могут быть наличие просроченной задолженности более 30 дней, 60 дней, 90 дней и более по одному кредиту на текущий момент или худший статус за все время кредитной истории, размер просроченной задолженности, глубина просрочки.

5. Загрузка данных из CSV-файлов и баз данных SQL

Загрузка CSV-файлов выполняется с помощью функции `pd.read_csv()`, а загрузку XLS-файлов можно выполнить с помощью функции `pd.read_excel()`.

Часто при загрузке CSV-файлов у нас возникают проблемы с кодировкой. Питоновский пакет chardet помогает определить тип кодировки прочитываемого файла.

```
# импортируем класс UniversalDetector библиотеки chardet
from chardet.universaldetector import UniversalDetector

# создаем экземпляр класса UniversalDetector
detector = UniversalDetector()

# определяем кодировку файла Credit_OTP.csv
with open('Data/Credit_OTP.csv', 'rb') as fh:
    for line in fh:
        detector.feed(line)
        if detector.done:
            break
    detector.close()
print(detector.result)
```

```
{'encoding': 'windows-1251', 'confidence': 0.8897840802899516, 'language': 'Russian'}
```

Нередко нам приходится работать с большими наборами данных. Мы можем ускорить функцию `pd.read_csv()` за счет распараллеливания. Используем эвристики, используемые во фреймворке AutoML от Лаборатории искусственного интеллекта Сбербанка LAMA. Обратите внимание, что параметры `skiprows`, `nrows`, `index_col`, `header`, `names`, `chunksize` игнорируются и функция будет более требовательна к оперативной памяти. Давайте импортируем необходимые библиотеки, классы, модули и напишем функцию `read_csv()`, которая будет использовать ряд вспомогательных предварительно написанных функций.

```
# импортируем необходимые библиотеки, классы и модули
import numpy as np
import pandas as pd
from joblib import Parallel, delayed
from copy import copy
import warnings
import os

def get_filelen(fname):
    """
    Получает длину csv-файла.

    Параметры:
        fname: имя файла.
    Возвращает:
        длина файла.
    """
    cnt_lines = -1
    with open(fname, "rb") as fin:
        for line in fin:
```

```
            if len(line.strip()) > 0:
                cnt_lines += 1

    return cnt_lines

def get_batch_ids(arr, batch_size):
    """
    Генератор последовательностей, разбитых на батчи.

    Параметры:
        arr: последовательность.
        batch_size: размерность.
    Возвращает:
        Батчи.
    """
    n = 0
    while n < len(arr):
        yield arr[n : n + batch_size]
        n += batch_size

def read_csv_batch(file, offset, cnt, **read_csv_params):
    """
    Читает батч данных из csv-файла.

    Параметры:
        file: путь к файлу.
        offset: отступ.
        cnt: количество строк для чтения.
        **read_csv_params: вспомогательные параметры.
    Возвращает:
        Прочитанные данные.
    """
    read_csv_params = copy(read_csv_params)
    if read_csv_params is None:
        read_csv_params = {}

    try:
        usecols = read_csv_params.pop("usecols")
    except KeyError:
        usecols = None

    header = pd.read_csv(file, nrows=0, **read_csv_params).columns

    with open(file, "rb") as f:
        f.seek(offset)
        data = pd.read_csv(f, header=None, names=header,
                           chunksize=None, nrows=cnt,
                           usecols=usecols, **read_csv_params)

    return data

def get_file_offsets(file, n_jobs=None, batch_size=None):
    """
    Получает отступы.

    Параметры:
        file: путь к файлу.
        n_jobs: количество ядер процессора для распараллеливания.
        batch_size: размер батча.
    Возвращает:
        Кортеж отступов.
    """
```

```
    assert (
        n_jobs is not None or batch_size is not None
    ), "One of n_jobs or batch size should be defined"

    lens = []
    with open(file, "rb") as f:
        header_len = len(f.readline())
        length = 0
        for row in f:
            if len(row.strip()) > 0:
                lens.append(length)
                length += len(row)

    lens = np.array(lens, dtype=np.int64) + header_len

    if batch_size:
        indexes = list(get_batch_ids(lens, batch_size))
    else:
        indexes = np.array_split(lens, n_jobs)

    offsets = [x[0] for x in indexes]
    cnts = [x.shape[0] for x in indexes]

    return offsets, cnts

def _check_csv_params(**read_csv_params):
    """
    Проверяет параметры для функции `read_csv`.

    Параметры:
        **read_csv_params: прочитывает параметры.
    Возвращает:
        Новые параметры.
    """
    for par in ["skiprows", "nrows", "index_col",
                "header", "names", "chunksize"]:
        if par in read_csv_params:
            read_csv_params.pop(par)
            warnings.warn(
                "Parameter {0} will be ignored in parallel mode".format(par),
                UserWarning,
            )

    return read_csv_params

def read_csv(file, n_jobs=4, **read_csv_params):
    """
    Прочитывает данные из csv-файла.

    Параметры:
        file: путь к файлу.
        n_jobs: количество ядер процессора для распараллеливания.
        **read_csv_params: вспомогательные параметры.
    Возвращает:
        Прочитанные данные.
    """
    if n_jobs == 1:
        return pd.read_csv(file, **read_csv_params)

    if n_jobs == -1:
        n_jobs = os.cpu_count()

    _check_csv_params(**read_csv_params)
    offsets, cnts = get_file_offsets(file, n_jobs)
```

```
    with Parallel(n_jobs) as p:
        res = p(
            delayed(read_csv_batch)(file, offset=offset, cnt=cnt,
                                    **read_csv_params)
            for (offset, cnt) in zip(offsets, cnts)
        )

    res = pd.concat(res, ignore_index=True)

    return res
```

Теперь применим нашу функцию.

```
%%time

data = read_csv('Data/paribas_train.csv', n_jobs=1)
```

```
CPU times: user 1.07 s, sys: 120 ms, total: 1.19 s
Wall time: 1.19 s
```

Библиотека pandas может считать данные из любой базы данных SQL, которая поддерживает адаптеры данных Python в рамках интерфейса Python DB-API. Чтение выполняется с помощью функций `pandas.read_sql()` и `pandas.read_sql_query()`, а запись в базу данных SQL выполняется с помощью метода `.to_sql()` объекта DataFrame.

Для иллюстрации программный код, приведенный ниже, считывает данные о котировках акций из файлов *msft.csv* и *aapl.csv*. Затем он подключается к файлу базы данных SQLite3. Если файл не существует, он создается «на лету». Затем программный код записывает данные MSFT в таблицу под названием STOCK_DATA. Если таблица не существует, она также будет создана. Если она уже существует, все данные заменяются данными о котировках акций MSFT. Наконец, программный код добавляет в эту таблицу данные о котировках акций AAPL.

```
# импортируем библиотеку SQLite
import sqlite3

# считываем данные о котировках акций из CSV-файла
msft = pd.read_csv('Data/msft.csv')
msft['Symbol']='MSFT'
aapl = pd.read_csv('Data/aapl.csv')
aapl['Symbol']='AAPL'

# создаем подключение
connection = sqlite3.connect('Data/stocks.sqlite')
# .to_sql() создаст базу SQL для хранения датафрейма в указанной таблице.
# параметр if_exists задает действие, которое нужно выполнить в том случае,
# если таблица уже существует ('fail' - выдать ошибку ValuError, 'replace' - # удалить таблицу перед вставкой новых значений, 'append' - вставить
# новые значения в существующую таблицу)
msft.to_sql('STOCK_DATA', connection, if_exists='replace') aapl.to_sql('STOCK_DATA', connection, if_exists='append')

# подтверждаем отправку данных в базу и закрываем подключение
connection.commit()
connection.close()
```

Чтобы убедиться в создании данных, можно открыть файл базы данных с помощью такого инструмента, как SQLite Database Browser (доступен по адресу *https://github.com/sqlitebrowser/sqlitebrowser*). Рисунок ниже показывает несколько записей в файле базы данных.

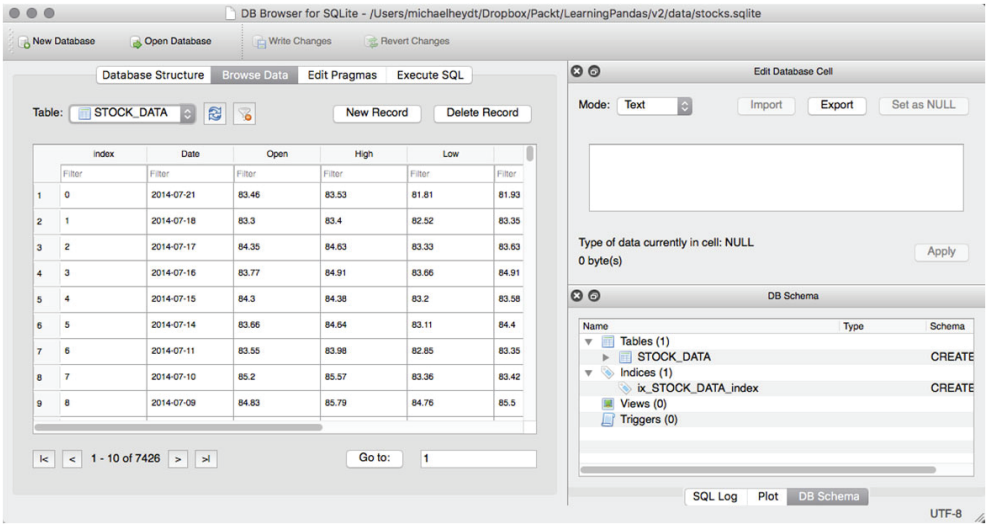


Рис. 7 SQLite Database Browser

Данные из базы данных SQL можно прочитать с помощью функции `pandas.read_sql()`. Следующий программный код демонстрирует выполнение запроса к файлу stocks.sqlite с помощью SQL и сообщает об этом пользователю.

```
# подключаемся к файлу базы данных
connection = sqlite3.connect('Data/stocks.sqlite')

# запрос всех записей в STOCK_DATA # возвращает датафрейм
# index_col задает столбец, который нужно сделать # индексом датафрейма
stocks = pd.read_sql("SELECT * FROM STOCK_DATA;",
   connection, index_col='index')

# закрываем подключение
connection.close()

# выводим первые 5 наблюдений в извлеченных данных
stocks[:5]
```

index	Date	Open	High	Low	Close	Volume	Symbol
0	7/21/2014	83.46	83.53	81.81	81.93	2359300	MSFT
1	7/18/2014	83.30	83.40	82.52	83.35	4020800	MSFT
2	7/17/2014	84.35	84.63	83.33	83.63	1974000	MSFT
3	7/16/2014	83.77	84.91	83.66	84.91	1755600	MSFT
4	7/15/2014	84.30	84.38	83.20	83.58	1874700	MSFT

Кроме того, для отбора столбцов еще можно использовать условие WHERE в SQL. Чтобы продемонстрировать это, следующий программный код отбирает записи, в которых количество проторгованных акций MSFT превышает 29200100.

```
# открываем подключение
connection = sqlite3.connect('Data/stocks.sqlite')
# создаем строку-запрос
query = "SELECT * FROM STOCK_DATA WHERE Volume>29200100 AND Symbol='MSFT';"
# выполняем запрос
items = pd.read_sql(query, connection, index_col='index')

# выводим результат запроса
items
```

index	Date	Open	High	Low	Close	Volume	Symbol
1081	5/21/2010	42.22	42.35	40.99	42.00	33610800	MSFT
1097	4/29/2010	46.80	46.95	44.65	45.92	47076200	MSFT
1826	6/15/2007	89.80	92.10	89.55	92.04	30656400	MSFT
3455	3/16/2001	47.00	47.80	46.10	45.33	40806400	MSFT
3712	3/17/2000	49.50	50.00	48.29	50.00	50860500	MSFT

Для этой же операции можно воспользоваться функцией `pandas.read_sql_query()`.

```
# можно воспользоваться функцией pandas.read_sql_query()
items2 = pd.read_sql_query(
    "SELECT * FROM STOCK_DATA WHERE Volume>29200100 AND Symbol='MSFT';",
    connection,
    index_col='index')
# закрываем подключение
connection.close()

# выводим результат запроса
items2
```

index	Date	Open	High	Low	Close	Volume	Symbol
1081	5/21/2010	42.22	42.35	40.99	42.00	33610800	MSFT
1097	4/29/2010	46.80	46.95	44.65	45.92	47076200	MSFT
1826	6/15/2007	89.80	92.10	89.55	92.04	30656400	MSFT
3455	3/16/2001	47.00	47.80	46.10	45.33	40806400	MSFT
3712	3/17/2000	49.50	50.00	48.29	50.00	50860500	MSFT

Итоговым моментом является то, что большая часть программного кода в этих примерах была программным кодом SQLite3. Библиотека pandas в этих примерах используется лишь тогда, когда нужно применить метод `.to_sql()`, функции `pandas.read_sql()` и `pandas.read_sql_query()`. Они принимают объект подключения, который может быть любым адаптером данных, совместимым с интерфейсом Python DB-API, поэтому вы можете работать с любой информацией базы данных, просто создав соответствующий объект подключения. Программный код на уровне pandas остается неизменным для любой поддерживаемой базы данных.

6. Удаление бесполезных переменных, переменных «из будущего», переменных с юридическим риском

Переменные, у которых количество категорий совпадает с количеством наблюдений, или переменные с одним уникальным значением (переменные-константы) бесполезны для анализа, и поэтому их удаляют самыми первыми.

Также необходимо удалить самыми первыми переменные «из будущего». Простой пример – мы предсказываем стоимость квадратного метра жилья. В нашем распоряжении есть исторические данные о реализованных сделках, среди признаков – время экспозиции квартиры. Однако когда нам нужно будет применить нашу модель для оценки стоимости новой квартиры, выставленной на продажу, у нас по этой квартире не будет данных о ее экспозиции.

Рисунки также несут переменные, в отношении которых мы знаем, что они не фиксировались в течение всего периода сбора исторических данных, или мы предполагаем, что в будущем не сможем получить информацию по этим переменным легальным способом.

Например, МФО фиксировала признак *Служба в армии* при выдаче микрокредита, а затем после корректировки правил кредитной политики отказалась от этого признака. У нас есть данные, где в течение первых 8 месяцев исторических данных признак *Служба в армии* фиксировался, а в последующие 4 месяца исторических данных он перестал фиксироваться (значения записаны как пропуски) и в новых данных его не будет.

Здесь можно привести пример, когда компания DoubleData фиксировала данные о характеристиках пользователей соцсетей, чтобы по ним оценить кредитоспособность или склонность к мошенничеству. Речь шла о социальном капитале (количество друзей, количество друзей друзей, количество неактивных друзей, количество друзей с дефолтом, уже доказано: чем больше у клиента друзей, находящихся в дефолте, тем больше его вероятность дефолта), активности (в рабочее время / в нерабочее время, днём/ночью, в выходные/будние дни), уникальности контента.

Ниже приведен профиль типичного фродстера в сети «ВКонтакте». Человек был осужден по ст. 159 УК РФ. Мошенничество с кредитами, обман клиентов и пр.

Рис. 8 Профиль фродстера в сети «ВКонтакте»

Однако затем законность получения данных оспорила компания Mail.ru, которая владеет сетью «ВКонтакте», и данными DoubleData нельзя было уже воспользоваться.

Также рекомендуется внимательно включать в скоринговые модели переменные сегментации (регионы продаж, канал продаж, тип продукта), поскольку они часто содержат не всегда наблюдаемые факторы (маркетинг, кредитная политика, данные продаж и т. д.) и зависят от них. Мы часто можем отказаться от выдачи кредита в том или ином регионе, отказаться от того или иного продукта.

7. Преобразование типов переменных и знакомство со шкалами переменных

Преобразование типов переменных – один из первых этапов предварительной подготовки данных. Операции по преобразованию типов переменных не будут эффективными, если нет четкого понимания, какой тип шкалы у переменной, поскольку именно шкала определяет тип переменной.

Шкала – правило, определяющее, каким образом в процессе измерения каждому изучаемому объекту ставится в соответствие некоторое число или символы.

Это правило включает в себя три следующих вопроса.

- Можем ли мы вычислить точные расстояния между значениями? Можем ли мы сказать, на сколько и во сколько раз одно значение больше/меньше другого?
- Можем ли мы упорядочить значения по тому или иному критерию?
- Можем ли мы сказать, сколько наблюдений в каждом значении?

Выделяют количественные (непрерывные) и качественные (дискретные) шкалы.

7.1. Количественные (непрерывные) шкалы

Переменная с количественной (непрерывной) шкалой – это переменная, которая может принимать бесконечное (неисчислимое) количество значений. Все переменные с количественными шкалами являются характеристиками, которые количественно описывают продукт. Примерами непрерывных переменных являются возраст, температура, доход. Мы можем вычислить средний возраст, среднюю температуру и средний доход. Количественная шкала позволяет утвердительно ответить на все три вышеприведенных вопроса. Возьмем переменную *Доход* со значениями 90 000, 90 000, 10 000, 10 000, 40 000, 10 000. Мы можем вычислить точные расстояния между значениями. Мы можем узнать, на сколько и во сколько раз одно значение больше/меньше другого, обратите внимание: второй тип сравнения не всегда возможен. Человек с доходом 40 000 рублей богаче человека с доходом 10 000 рублей на 30 000 рублей, или в 4 раза. Мы можем упорядочить значения по возрастанию дохода: 10 000, 40 000, 90 000. Мы можем сказать, что у нас 2 человека с доходом 90 000 рублей, 3 человека с доходом 10 000 рублей, один человек с доходом 40 000 рублей.

Среди количественных шкал выделяют:

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

Шкала интервалов состоит из одинаковых интервалов и имеет *условную нулевую точку* (точку отсчета).

Примером шкалы интервалов могут служить шкалы для измерения температуры по Цельсию и Фаренгейту. Ноль по шкале Фаренгейта определяется

по самоподдерживающейся температуре смеси воды, льда и хлорида аммония (соответствует примерно –17,8 °C). Ноль шкалы Цельсия установлен таким образом, что температура тройной точки воды равна 0,01 °C.

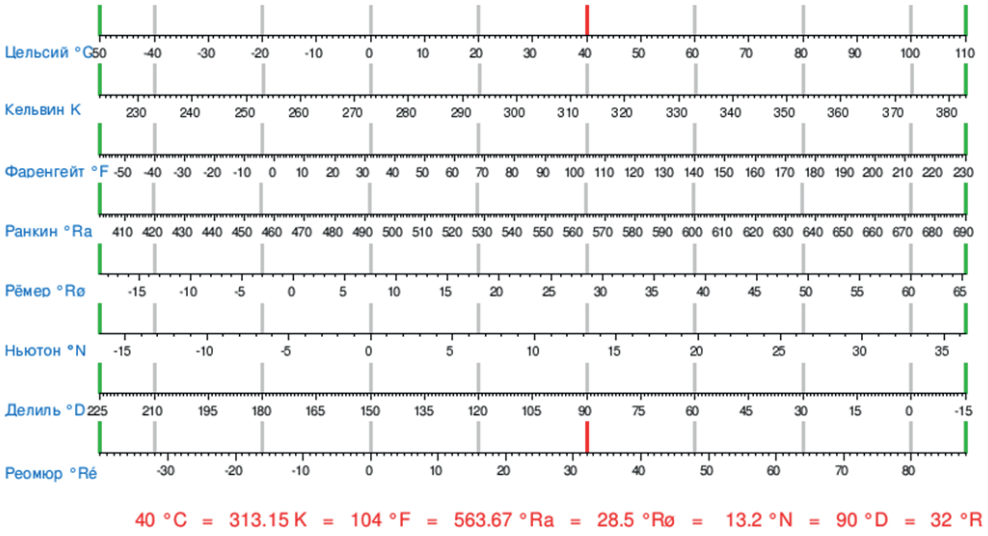


Рис. 9 Диаграмма перевода температур

Шкала интервалов позволяет сказать, насколько одно значение больше другого, но не позволяет сказать, во сколько раз оно больше. Например, повысив температуру с 1 °C до 20 °C, мы можем сказать, что температура 20 °C на 19 °C больше 1 °C, но не можем сказать, что температура 20 °C в 20 раз больше, чем 1 °C. 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 1 °C до 20 °C абсолютная температура изменится всего в (273 + 20) / (273 + 1) = 1,069 раза. Ноль по шкале Цельсия соответствует 273 K. Другим примером шкал этого типа являются шкалы календарного времени.

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.

Абсолютная шкала, помимо естественной нулевой точки, имеет еще и *естественную общепринятую единицу измерения*.

Пример абсолютной шкалы – абсолютная шкала температуры, или шкала Кельвина. Ноль этой шкалы отвечает полному прекращению движения моле-

кул, 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 переменная с количественной шкалой – самая информативная переменная, у нас есть информация о расстояниях между значениями, можем упорядочить значения, можем сказать, сколько наблюдений принадлежат конкретному значению.

7.2. Качественные (дискретные) шкалы

Переменная с качественной (дискретной) шкалой – это переменная, которая может принимать одно значение из ограниченного и обычно фиксированного набора возможных значений. Каждое из возможных значений часто называется еще уровнем (level). Все переменные с качественными шкалами являются характеристиками, которые качественно описывают продукт. Примерами переменных с качественной (дискретной) шкалой являются пол, штат, социальный класс, уровень благосостояния, группа крови, гражданство, образование. Из-за дискретного характера шкалы вычисление среднего значения переменной не имеет смысла, мы не можем вычислить средний пол или среднюю сферу занятости.

Качественные (дискретные) шкалы бывают порядковыми и номинальными.

7.2.1. Порядковая шкала

Порядковая шкала позволяет утвердительно ответить только на два последних вопроса.

Пример переменной с порядковой шкалой – переменная *Уровень дохода*. Она имеет уровни *Низкий*, *Средний*, *Высокий*. Мы можем сказать, сколько у нас наблюдений в каждом уровне. Мы можем упорядочить уровни: *Низкий*, *Средний*, *Высокий*. 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 шкала будет менее информативной, чем количественная: у нас исчезает информация о расстояниях между значениями, но мы по-прежнему можем упорядочить значения, можем сказать, сколько наблюдений принадлежат конкретному значению.

7.2.2. Номинальная шкала

Номинальная шкала позволяет утвердительно ответить только на последний вопрос.

Пример переменной с номинальной шкалой – переменная *Сфера деятельности*. Она имеет уровни *Строительство*, *Транспорт* и *Металлургия*. Мы можем сказать, сколько у нас наблюдений в каждом уровне. Однако мы не можем сказать, что уровень *Строительство* хуже/лучше/меньше/больше уровня *Металлургия*. У нас нет информации о расстояниях между уровнями. Номинальная переменная будет менее информативной, чем порядковая: мы можем лишь сказать, сколько наблюдений принадлежат конкретному значению. К переменным с номинальной шкалой относятся бинарные переменные – переменные с двумя значениями.

Ниже приведена таблица с примером, когда переменную *Образование* можно записать в трех типах шкалы.

ТИП ШКАЛЫ	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

Рис. 10 Три типа шкалы

Из-за неправильного десятичного разделителя количественная переменная может быть ошибочно записана как категориальная.

Бинарная зависимая переменная часто записывается с помощью значений 0 или 1 и прочитывается как целочисленная. В Python мы можем такую переменную не преобразовывать в категориальную (типы `object`), стратегия обработки определяется выбранным классом – классификатором или регрессором.

В Python и H2O порядковые переменные не поддерживаются, обрабатываем их как категориальные номинальные переменные (тип `object` в Python и тип `enum` в H2O) или количественные целочисленные (тип `int` в Python и H2O).

Таблица 1 Представление типов шкал в Python и H2O

	Python	H2O
Количественная	int float	int real
Порядковая	не поддерживается, представляют как `object` или как `int` `Categorical` (только для разведочного анализа)	не поддерживается, представляют как `enum` или как `int`
Номинальная	`object` `Categorical` (только для разведочного анализа)	enum

8. Нормализация строковых значений

Часто строковые значения переменных могут содержать лишние символы типа &*_.

Импортируем библиотеку pandas, прочитываем данные в датафрейм и смотрим первые пять наблюдений.

```
# импортируем библиотеки pandas и numpy
import pandas as pd
import numpy as np

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

	gender	marital
0	Женский	Женат
1	Мужской	Одинокий
2	Женский	Одинокий
3	Мужской	Одинокий
4	Женский	Одинокий

На первый взгляд вроде бы все в порядке, но давайте взглянем на уникальные значения переменных с помощью метода `.unique()`.

```
# создаем список переменных
cols = data.columns
# с помощью метода .unique() выводим уникальные
# значения переменных gender и marital
for col in cols:
    print(data[col].unique())
```

```
['Женский' 'Мужской' 'Женский&*' 'Мужской&*']
['Женат' 'Одинокий' '_Одинокий' '_Женат' 'Же&нат']
```

Видим лишние символы типа &*_.

Теперь удалим лишние символы типа &*_ в переменных *gender* и *marital_status* с помощью метода `.str.replace()` объекта Series библиотеки pandas. Метод имеет общий вид:

```
Series.str.replace(pat, repl)
```

где

`pat` задает символ, который нужно найти

`repl` задает символ, на который нужно заменить найденный символ

```
# с помощью метода .str.replace() удаляем лишние символы
for col in cols:
```

```
        data[col] = data[col].str.replace('[*&_]', '')
    # выводим уникальные значения переменных
    # gender и marital
    for col in cols:
        print(data[col].unique())

    ['Женский' 'Мужской']
    ['Женат' 'Одинокий']
```

Видим, что лишние символы удалены.

Часто бывает необходимо выполнить транслитерацию строковых значений, например передать русские названия латиницей, потому что тот или иной пакет не поддерживает кириллицу. В этом нам поможет функция `translit()` библиотеки `transliterate`.

Давайте выполним транслитерацию строковых значений переменных *gender* и *marital_status* латиницей.

```
# импортируем библиотеку для транслитерации
from transliterate import translit
# выполняем транслитерацию
for col in cols:
    data[col] = data[col].apply(
        lambda x: translit(x, 'ru', reversed=True))
# выводим уникальные значения переменных
# gender и marital
for col in cols:
    print(data[col].unique())

['Zhenskij' 'Muzhskoj']
['Zhenat' 'Odinokij']
```

С помощью методов `str.lower()` и `str.upper()` можем задать нижний и верхний регистры соответственно.

```
# переводим строки (значения переменной gender)
# в нижний регистр
data['gender'] = data['gender'].str.lower()
# переводим строки (значения переменной marital)
# в ВЕРХНИЙ регистр
data['marital'] = data['marital'].str.upper()
data.head()
```

	gender	marital
0	zhenskij	ZHENAT
1	muzhskoj	ODINOKIJ
2	zhenskij	ODINOKIJ
3	muzhskoj	ODINOKIJ
4	zhenskij	ODINOKIJ

Часто при работе с базами данных бывает ситуация, когда у нас есть информация об именах, отчествах и фамилиях клиентов, но нет информации о поле. Давайте рассмотрим случай, когда по отчеству каждого клиента можно определить его пол.

```
# загружаем CSV-файл с ФИО клиентов,
# по которым нужно определить пол
data = pd.read_csv('Data/Gender_based_on_middle_name.csv',
 encoding='cp1251', sep=';')
data.head(20)
```

	Клиент	Возраст	Регион	Статус
0	_Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)
1	_Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)
11	Абдурасулова Наталья Таджиловна	55&	Екатеринбург- 8	Вернул(а)
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)
19	Абрамов Никита Валерьевич	45лет	Екатеринбург- 8	Вернул(а)

Теперь мы создадим переменную *Пол*, которая будет иметь значение `True`, если строковое значение переменной *Клиент* содержит паттерн `вна` (Викторовна, Дмитриевна), и `False` в противном случае. Для этого воспользуемся методом `.str.contains()`.

```
# создаем переменную Пол, которая будет иметь значение True,
# если строковое значение переменной Клиент содержит паттерн
# "вна" (Викторовна, Дмитриевна), и False в противном случае
data['Пол'] = data['Клиент'].str.contains('вна')
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол
0	_Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	False
1	_Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	False
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	False
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	True
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	True
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	True
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	True
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	False
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	False
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	True
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	True
11	Абдурасулова Наталья Таджиловна	55&	Екатеринбург- 8	Вернул(а)	True
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	True
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	True
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	True
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	True
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	False
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	True
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	False
19	Абрамов Никита Валерьевич	45лет	Екатеринбург- 8	Вернул(а)	False

Теперь переименуем категории переменной *Пол* с помощью словаря.

	Клиент	Возраст	Регион	Статус	Пол
0	_Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	Мужской
1	_Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	Мужской
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	Мужской
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	Женский
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	Женский
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	Женский
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	Женский
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	Мужской
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	Мужской
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский
11	Абдурасулова Наталья Таджиловна	55&	Екатеринбург- 8	Вернул(а)	Женский
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	Женский
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	Женский
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	Женский
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	Женский
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	Мужской
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	Женский
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	Мужской
19	Абрамов Никита Валерьевич	45лет	Екатеринбург- 8	Вернул(а)	Мужской

Еще можно было извлечь последние три символа в каждом строковом значении переменной *Клиент* и затем на основе полученных значений создать новую переменную.

```
# извлекаем последние три символа в каждом строковом
# значении переменной Клиент и затем на основе
# полученных значений создаем новую переменную
data['Пол2'] = data['Клиент'].str[-3:]
# переименуем категории переменной Пол2
d = {'вич': 'Мужской', 'вна': 'Женский'}
data['Пол2'] = data['Пол2'].map(d)
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2
0	_Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	Мужской	Мужской
1	_Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	Мужской	Мужской
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	Мужской	Мужской
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	Женский	Женский
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	Женский	Женский
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
11	Абдурасулова Наталья Таджиловна	55&	Екатеринбург- 8	Вернул(а)	Женский	Женский
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	Женский	Женский
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	Женский	Женский
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	Женский	Женский
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	Женский	Женский
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	Мужской	Мужской
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	Женский	Женский
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	Мужской	Мужской
19	Абрамов Никита Валерьевич	45лет	Екатеринбург- 8	Вернул(а)	Мужской	Мужской

Теперь с помощью метода `.str.lstrip()` удалим ненужный символ подчеркивания, с которого начинаются несколько значений переменной *Клиент*. Метод `.str.lstrip()` возвращает копию указанной строки, с начала которой (т. е. слева l – left) устранены указанные символы.

```
# с помощью метода .str.lstrip() удалим ненужный символ
# подчеркивания, с которого начинаются несколько значений
# переменной Клиент, метод .str.lstrip() возвращает
# копию указанной строки, с начала (слева l - left)
# которой устранены указанные символы
data['Клиент'] = data['Клиент'].str.lstrip('_')
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2
0	Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	Мужской	Мужской
1	Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	Мужской	Мужской
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	Мужской	Мужской
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	Женский	Женский
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	Женский	Женский
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
11	Абдурасулова Наталья Таджиловна	55&	Екатеринбург- 8	Вернул(а)	Женский	Женский
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	Женский	Женский
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	Женский	Женский
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	Женский	Женский
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	Женский	Женский
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	Мужской	Мужской
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	Женский	Женский
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	Мужской	Мужской
19	Абрамов Никита Валерьевич	45лет	Екатеринбург- 8	Вернул(а)	Мужской	Мужской

Теперь с помощью метода `.str.rstrip()` удалим ненужные символы, которыми заканчиваются некоторые значения переменной *Возраст*. Метод `.str.rstrip()` возвращает копию указанной строки, с конца которой (справа `r - right`) устранены указанные символы.

```
# с помощью метода .str.rstrip() удалим ненужные символы, которыми
# заканчиваются некоторые значения переменной Возраст, метод
# .str.rstrip() возвращает копию указанной строки, с конца
# (справа r - right) которой устранены указанные символы
data['Возраст'] = data['Возраст'].str.rstrip('&лет')
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2
0	Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	Мужской	Мужской
1	Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	Мужской	Мужской
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	Мужской	Мужской
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	Женский	Женский
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	Женский	Женский
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	Женский	Женский
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский
11	Абдурасулова Наталья Таджиловна	55	Екатеринбург- 8	Вернул(а)	Женский	Женский
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	Женский	Женский
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	Женский	Женский
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	Женский	Женский
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	Женский	Женский
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	Мужской	Мужской
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	Женский	Женский
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	Мужской	Мужской
19	Абрамов Никита Валерьевич	45	Екатеринбург- 8	Вернул(а)	Мужской	Мужской

Теперь создадим переменную *Регион2*, мы просто извлечем цифры из переменной *Регион* с помощью метода `.str.extract()`.

```
# создадим переменную Регион2, извлекая
# цифры из переменной Регион
data['Регион2'] = data['Регион'].str.extract('(\d)', expand=True) data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2	Регион2
0	Колесников Вячеслав Анатольевич	33	Красноярск- 2	Вернул(а)	Мужской	Мужской	2
1	Саймурзанов Михаил Борисович	22	Красноярск- 2	Вернул(а)	Мужской	Мужской	2
2	Абаимов Максим Дмитриевич	43	Москва- 4	Вернул(а)	Мужской	Мужской	4
3	Абакумова Юлия Ивановна	22	Москва- 4	Вернул(а)	Женский	Женский	4
4	Абанова Елена Владимировна	54	Санкт-Петербург- 6	Вернул(а)	Женский	Женский	6
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург- 6	Вернул(а)	Женский	Женский	6
6	Абдугалиева Айгуль Максутовна	27	Москва- 4	Не вернул(а)	Женский	Женский	4
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской	8
8	Абдуллин Евгений Эдуардович	22	Екатеринбург- 8	Не вернул(а)	Мужской	Мужской	8
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
11	Абдурасулова Наталья Таджиловна	55	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
12	Абдурахимова Алена Алимовна	57	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
14	Аблец Юлия Сергеевна	33	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
15	Аболмасова Ирина Олеговна	38	Екатеринбург- 8	Вернул(а)	Женский	Женский	8
16	Абраев Нурлан Мусайбекович	49	Екатеринбург- 8	Вернул(а)	Мужской	Мужской	8
17	Абраменко Екатерина Владимировна	56	Москва- 4	Вернул(а)	Женский	Женский	4
18	Абрамов Дмитрий Владимирович	51	Москва- 4	Вернул(а)	Мужской	Мужской	4
19	Абрамов Никита Валерьевич	45	Екатеринбург- 8	Вернул(а)	Мужской	Мужской	8

Теперь удалим последние три символа в каждом строковом значении переменной *Регион*.

```
# удаляем последние 3 символа в каждом строковом
# значении переменной Регион
data['Регион'] = data['Регион'].map(lambda x: str(x)[:-3])
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2	Регион2
0	Колесников Вячеслав Анатольевич	33	Красноярск	Вернул(а)	Мужской	Мужской	2
1	Саймурзанов Михаил Борисович	22	Красноярск	Вернул(а)	Мужской	Мужской	2
2	Абаимов Максим Дмитриевич	43	Москва	Вернул(а)	Мужской	Мужской	4
3	Абакумова Юлия Ивановна	22	Москва	Вернул(а)	Женский	Женский	4
4	Абанова Елена Владимировна	54	Санкт-Петербург	Вернул(а)	Женский	Женский	6
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург	Вернул(а)	Женский	Женский	6
6	Абдугалиева Айгуль Максутовна	27	Москва	Не вернул(а)	Женский	Женский	4
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург	Не вернул(а)	Мужской	Мужской	8
8	Абдуллин Евгений Эдуардович	22	Екатеринбург	Не вернул(а)	Мужской	Мужской	8
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернул(а)	Женский	Женский	8
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернул(а)	Женский	Женский	8
11	Абдурасулова Наталья Таджиловна	55	Екатеринбург	Вернул(а)	Женский	Женский	8
12	Абдурахимова Алена Алимовна	57	Екатеринбург	Вернул(а)	Женский	Женский	8
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург	Вернул(а)	Женский	Женский	8
14	Аблец Юлия Сергеевна	33	Екатеринбург	Вернул(а)	Женский	Женский	8
15	Аболмасова Ирина Олеговна	38	Екатеринбург	Вернул(а)	Женский	Женский	8
16	Абраев Нурлан Мусайбекович	49	Екатеринбург	Вернул(а)	Мужской	Мужской	8
17	Абраменко Екатерина Владимировна	56	Москва	Вернул(а)	Женский	Женский	4
18	Абрамов Дмитрий Владимирович	51	Москва	Вернул(а)	Мужской	Мужской	4
19	Абрамов Никита Валерьевич	45	Екатеринбург	Вернул(а)	Мужской	Мужской	8

Удалим круглые скобки в переменной *Статус*.

```
# удаляем круглые скобки в переменной Статус
data['Статус'] = data['Статус'].str.replace('[()]', '')
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2	Регион2
0	Колесников Вячеслав Анатольевич	33	Красноярск	Вернула	Мужской	Мужской	2
1	Саймурзанов Михаил Борисович	22	Красноярск	Вернула	Мужской	Мужской	2
2	Абаимов Максим Дмитриевич	43	Москва	Вернула	Мужской	Мужской	4
3	Абакумова Юлия Ивановна	22	Москва	Вернула	Женский	Женский	4
4	Абанова Елена Владимировна	54	Санкт-Петербург	Вернула	Женский	Женский	6
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург	Вернула	Женский	Женский	6
6	Абдугалиева Айгуль Максутовна	27	Москва	Не вернула	Женский	Женский	4
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург	Не вернула	Мужской	Мужской	8
8	Абдуллин Евгений Эдуардович	22	Екатеринбург	Не вернула	Мужской	Мужской	8
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернула	Женский	Женский	8
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернула	Женский	Женский	8
11	Абдурасулова Наталья Таджиловна	55	Екатеринбург	Вернула	Женский	Женский	8
12	Абдурахимова Алена Алимовна	57	Екатеринбург	Вернула	Женский	Женский	8
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург	Вернула	Женский	Женский	8
14	Аблец Юлия Сергеевна	33	Екатеринбург	Вернула	Женский	Женский	8
15	Аболмасова Ирина Олеговна	38	Екатеринбург	Вернула	Женский	Женский	8
16	Абраев Нурлан Мусайбекович	49	Екатеринбург	Вернула	Мужской	Мужской	8
17	Абраменко Екатерина Владимировна	56	Москва	Вернула	Женский	Женский	4
18	Абрамов Дмитрий Владимирович	51	Москва	Вернула	Мужской	Мужской	4
19	Абрамов Никита Валерьевич	45	Екатеринбург	Вернула	Мужской	Мужской	8

Выполняем итоговую нормализацию строковых значений переменной *Статус*.

```
# выполняем итоговую нормализацию строковых
# значений переменной Статус
data['Статус'] = np.where(data['Клиент'].str.contains('вна'),
                          data['Статус'],
                          data['Статус'].map(lambda x: str(x)[:-1]))
data.head(20)
```

	Клиент	Возраст	Регион	Статус	Пол	Пол2	Регион2
0	Колесников Вячеслав Анатольевич	33	Красноярск	Вернул	Мужской	Мужской	2
1	Саймурзанов Михаил Борисович	22	Красноярск	Вернул	Мужской	Мужской	2
2	Абаимов Максим Дмитриевич	43	Москва	Вернул	Мужской	Мужской	4
3	Абакумова Юлия Ивановна	22	Москва	Вернула	Женский	Женский	4
4	Абанова Елена Владимировна	54	Санкт-Петербург	Вернула	Женский	Женский	6
5	Абдрахимова Юлия Рафиковна	23	Санкт-Петербург	Вернула	Женский	Женский	6
6	Абдугалиева Айгуль Максутовна	27	Москва	Не вернула	Женский	Женский	4
7	Абдуллаев Ильгар Эльдарович	44	Екатеринбург	Не вернул	Мужской	Мужской	8
8	Абдуллин Евгений Эдуардович	22	Екатеринбург	Не вернул	Мужской	Мужской	8
9	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернула	Женский	Женский	8
10	Абдуллина Екатерина Анатольевна	63	Екатеринбург	Вернула	Женский	Женский	8
11	Абдурасулова Наталья Таджиловна	55	Екатеринбург	Вернула	Женский	Женский	8
12	Абдурахимова Алена Алимовна	57	Екатеринбург	Вернула	Женский	Женский	8
13	Абельдина Гульпархия Галимжановна	41	Екатеринбург	Вернула	Женский	Женский	8
14	Аблец Юлия Сергеевна	33	Екатеринбург	Вернула	Женский	Женский	8
15	Аболмасова Ирина Олеговна	38	Екатеринбург	Вернула	Женский	Женский	8
16	Абраев Нурлан Мусайбекович	49	Екатеринбург	Вернул	Мужской	Мужской	8
17	Абраменко Екатерина Владимировна	56	Москва	Вернула	Женский	Женский	4
18	Абрамов Дмитрий Владимирович	51	Москва	Вернул	Мужской	Мужской	4
19	Абрамов Никита Валерьевич	45	Екатеринбург	Вернул	Мужской	Мужской	8

Часто данные могут быть некорректно записаны, например несколько полей могут быть записаны в одно, и необходимо извлечь их. Давайте загрузим данные.

```
# загружаем данные
data = pd.read_csv('Data/Raw_text.csv', encoding='cp1251')
data
```

	raw
0	KDR 1 2014-12-23 3242.0
1	MSK 1 2010-02-23 3453.7
2	KRSK 0 2014-06-20 2123.0
3	SPB 0 2014-03-14 1123.6
4	EKB 1 2013-01-15 2134.0

Видим, что несколько переменных записаны в один столбец *raw*. Давайте с помощью метода `.str.extract()` извлечем даты, создав переменную *date*.

```
# с помощью метода .str.extract() извлекаем
# даты из столбца raw, создав переменную date
data['date'] = data['raw'].str.extract(
    '(....-..-..)', expand=True)
data
```

	raw	date
0	KDR 1 2014-12-23 3242.0	2014-12-23
1	MSK 1 2010-02-23 3453.7	2010-02-23
2	KRSK 0 2014-06-20 2123.0	2014-06-20
3	SPB 0 2014-03-14 1123.6	2014-03-14
4	EKB 1 2013-01-15 2134.0	2013-01-15

Извлекаем одиночные цифры из столбца *raw*, создав переменную *gender*.

```
# извлекаем одиночные цифры из столбца raw,
# создав переменную gender
data['gender'] = data['raw'].str.extract(
    '(\d)', expand=True)
data
```

	raw	date	gender
0	KDR 1 2014-12-23 3242.0	2014-12-23	1
1	MSK 1 2010-02-23 3453.7	2010-02-23	1
2	KRSK 0 2014-06-20 2123.0	2014-06-20	0
3	SPB 0 2014-03-14 1123.6	2014-03-14	0
4	EKB 1 2013-01-15 2134.0	2013-01-15	1

Извлекаем числа с плавающей точкой из столбца *raw*, создав переменную *score*.

```
# извлекаем числа с плавающей точкой из столбца raw,
# создав переменную score
data['score'] = data['raw'].str.extract(
    '(\d\d\d\d\.\d)', expand=True)
data
```

	raw	date	gender	score
0	KDR 1 2014-12-23 3242.0	2014-12-23	1	3242.0
1	MSK 1 2010-02-23 3453.7	2010-02-23	1	3453.7
2	KRSK 0 2014-06-20 2123.0	2014-06-20	0	2123.0
3	SPB 0 2014-03-14 1123.6	2014-03-14	0	1123.6
4	EKB 1 2013-01-15 2134.0	2013-01-15	1	2134.0

Наконец, извлекаем текст из столбца *raw*, создав переменную *city*.

```
# извлекаем текст из столбца raw, создав переменную city
data['city'] = data['raw'].str.extract(
    '(\w+)', expand=True)
data
```

	raw	date	gender	score	city
0	KDR 1 2014-12-23 3242.0	2014-12-23	1	3242.0	KDR
1	MSK 1 2010-02-23 3453.7	2010-02-23	1	3453.7	MSK
2	KRSK 0 2014-06-20 2123.0	2014-06-20	0	2123.0	KRSK
3	SPB 0 2014-03-14 1123.6	2014-03-14	0	1123.6	SPB
4	EKB 1 2013-01-15 2134.0	2013-01-15	1	2134.0	EKB

Полезной процедурой для нормализации строк является вычисление расстояния Левенштейна.

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) – минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Таким образом, оно измеряет сходство двух строк.

Например, для слов Smith и Smythe расстояние Левенштейна будет равно 2 и вычисляется следующим образом:

Слово 1	S	m	i	t	h	
Слово 2	S	m	y	t	h	e
Оценка	0	0	**1**	0	0	**1**

Рис. 11 Пример вычисления расстояния Левенштейна

Расстояние Левенштейна активно применяется для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи), в маркетинге для поиска дублей в клиентских базах и унификации товарной номенклатуры, в биоинформатике для сравнения генов, хромосом и белков. В Python расстояние Левенштейна можно вычислить с помощью библиотеки python-Levenshtein (можно установить с помощью команды `pip install python-Levenshtein`).

Давайте вычислим расстояние Левенштейна для нашего игрушечного примера с помощью функции `distance()` библиотеки `python-Levenshtein`.

```
# импортируем функцию distance()
# библиотеки python-Levenshtein
from Levenshtein import distance
# вычисляем расстояние для двух строк
string1 = 'Smith'
string2 = 'Smythe'
distance(string1, string2)
```

```
2
```

С точки зрения применения определение расстояния между словами или текстовыми полями по Левенштейну обладает следующими недостатками:

- при перестановке местами слов или частей слов получаются сравнительно большие расстояния;

- расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными;
- расстояния между очень похожими словами, но в разных регистрах, оказываются большими, поэтому требуется предварительная «нормализация» строк, например приведение к одному и тому же регистру.

В некоторых недостатках мы сейчас убедимся.

Загружаем данные с двумя столбцами адресов.

```
# загружаем данные
df = pd.read_csv('Data/Levenshtein.csv', sep=';')
df
```

	line1	line2
0	9128 LEEWARD CIR, INDIANAPOLIS, IN	9128 Leeward Circle, Indianapolis, IN
1	101 OCEAN LANE DRIVE, KEY BISCAYNE, FL	101 Ocean Lane Drive Unit 1010, Key Biscayne, FL
2	9301 EVERGREEN DRIVE, PARMA, OH	9301 Evergreen Dr, Parma, OH
3	1817 BERTRAND DR, LAFAYETTE, LA	2924 Polo Ridge Ct, Charlotte, NC
4	201 E 87TH ST, NEW YORK, NY	201 E 87th Street 3E, New York, NY
5	799 CARRIGAN AVE, OVIEDO, FL	799 Carrigan Avenue, Oviedo, FL
6	4014 CADDIE DRIVE, ACWORTH, GA	10617 WEYBRIDGE DR, TAMPA, FL

Нетрудно увидеть, что есть очень похожие строки (выделены красными рамками). Вычисляем расстояние Левенштейна.

```
# вычисляем расстояние Левенштейна
df['distance'] = df.apply(
    lambda x: distance(x['line1'], x['line2']), axis=1)
df
```

	line1	line2	distance
0	9128 LEEWARD CIR, INDIANAPOLIS, IN	9128 Leeward Circle, Indianapolis, IN	22
1	101 OCEAN LANE DRIVE, KEY BISCAYNE, FL	101 Ocean Lane Drive Unit 1010, Key Biscayne, FL	30
2	9301 EVERGREEN DRIVE, PARMA, OH	9301 Evergreen Dr, Parma, OH	16
3	1817 BERTRAND DR, LAFAYETTE, LA	2924 Polo Ridge Ct, Charlotte, NC	26
4	201 E 87TH ST, NEW YORK, NY	201 E 87th Street 3E, New York, NY	15
5	799 CARRIGAN AVE, OVIEDO, FL	799 Carrigan Avenue, Oviedo, FL	17
6	4014 CADDIE DRIVE, ACWORTH, GA	10617 WEYBRIDGE DR, TAMPA, FL	22

Видим, что очень похожие строки имеют достаточно большие расстояния из-за того, что имеют разные регистры.

Выполним «нормализацию» строк, т. е. переведем буквы в верхний регистр, и снова вычислим расстояние Левенштейна.

```
# приводим строки к ВЕРХНЕМУ регистру
df['line1'] = df['line1'].str.upper()
```

```
df['line2'] = df['line2'].str.upper()
# снова вычисляем расстояние Левенштейна
df['distance_corr'] = df.apply(
    lambda x: distance(x['line1'], x['line2']), axis=1)
df
```

	line1	line2	distance	distance_corr
0	9128 LEEWARD CIR, INDIANAPOLIS, IN	9128 LEEWARD CIRCLE, INDIANAPOLIS, IN	22	3
1	101 OCEAN LANE DRIVE, KEY BISCAYNE, FL	101 OCEAN LANE DRIVE UNIT 1010, KEY BISCAYNE, FL	30	10
2	9301 EVERGREEN DRIVE, PARMA, OH	9301 EVERGREEN DR, PARMA, OH	16	3
3	1817 BERTRAND DR, LAFAYETTE, LA	2924 POLO RIDGE CT, CHARLOTTE, NC	26	23
4	201 E 87TH ST, NEW YORK, NY	201 E 87TH STREET 3E, NEW YORK, NY	15	7
5	799 CARRIGAN AVE, OVIEDO, FL	799 CARRIGAN AVENUE, OVIEDO, FL	17	3
6	4014 CADDIE DRIVE, ACWORTH, GA	10617 WEYBRIDGE DR, TAMPA, FL	22	22

Теперь мы видим уже более адекватные значения.

9. Обработка дублирующихся наблюдений

Нередко при подготовке выборки допускаются ошибки, в частности в набор данных несколько раз может попасть одно и то же наблюдение. Нужно убедиться, что наш набор не содержит дублирующихся наблюдений (строк). Для идентификации дублей используем метод `.duplicated()`, а для удаления дублей применяем метод `drop_dublicates()`.

```
# импортируем библиотеку pandas
import pandas as pd

# записываем CSV-файл в объект DataFrame
data = pd.read_csv('Data/Verizon.csv', sep=';')
# смотрим первые пять наблюдений
data.head()
```

	longdist	internat	local	int_disc	billtype	pay	age	gender	marital	children	income	churn
0	8.62	NaN	8.49	Нет	Бюджетный	CH	43.0	Мужской	_Женат	0.0	33935.8	0
1	21.27	0.0	218,12	Нет	Бюджетный	CH	60.0	NaN	_Одинокий	2.0	95930,6	1
2	6,13	0.0	NaN	Да	NaN	NaN	25.0	Женский	NaN	2.0	295,34	1
3	16.46	0.0	57,66	Да	Бесплатный	NaN	93.0	Женский	Одинокий	0.0	NaN	1
4	NaN	0.0	16,01	Да	Бесплатный	CC	68.0	Женский&*	NaN	0.0	99832,9	1

```
# посмотрим наличие дублей
data[data.duplicated(keep=False)]
```

	longdist	internat	local	int_disc	billtype	pay	age	gender	marital	children	income	churn
0	8.62	NaN	8.49	Нет	Бюджетный	CH	43.0	Мужской	_Женат	0.0	33935.8	0
13	8.62	NaN	8.49	Нет	Бюджетный	CH	43.0	Мужской	_Женат	0.0	33935.8	0
14	8.62	NaN	8.49	Нет	Бюджетный	CH	43.0	Мужской	_Женат	0.0	33935.8	0

```
# удаляем дубли на месте, оставляя первое
# встретившееся наблюдение в паттерне дубля
data.drop_duplicates(subset=None, keep='first',
                     inplace=True)
# посмотрим наличие дублей
data[data.duplicated(keep=False)]
```

10. Обработка редких категорий

Часто бывает, что наши переменные содержат редкие категории. Редкие категории являются источником шума в данных, который негативно повлияет на качество модели. Кроме того, при разбиении набора данных на обучающую и тестовую выборки может оказаться, что данная категория отсутствует в обучающей выборке, но присутствует в тестовой выборке. Это вызовет проблемы при моделировании. Например, логистическая регрессия, встретив в тестовых данных наблюдение с неизвестной категорией признака, не сможет вычислить прогноз, потому что категория не будет соответствовать схеме дамми-кодирования, полученной для переменной в обучающей выборке, и таким образом не будет вычислен соответствующий регрессионный коэффициент.

Давайте импортируем необходимые библиотеки и класс и загрузим данные с редкими категориями.

```
# импортируем необходимые библиотеки и класс
import pandas as pd
import numpy as np
from collections import defaultdict
# загружаем данные
data = pd.read_csv('Data/Rare_categories.csv', sep=';')
# выводим наблюдения
data.head()
```

	TARGET	GEN_INDUSTRY	GEN_TITLE	ORG_TP_STATE	ORG_TP_FCAPITAL	JOB_DIR
0	0	Торговля	Рабочий	Частная компания	Без участия	Вспомогательный техперсонал
1	0	Торговля	Рабочий	Индивидуальный предприниматель	Без участия	Участие в основ. деятельности
2	0	Информационные технологии	Специалист	Государственная комп./учреж.	Без участия	Участие в основ. деятельности
3	0	Образование	Руководитель среднего звена	Государственная комп./учреж.	Без участия	Участие в основ. деятельности
4	0	Государственная служба	Специалист	Государственная комп./учреж.	Без участия	Участие в основ. деятельности

Набор данных состоит из одних категориальных переменных. Давайте выведем частоты категорий по каждой переменной.

```
# создаем список категориальных переменных
cat_cols = data.dtypes[data.dtypes == 'object'].index.tolist()
# смотрим частоты по категориальным переменным
for col in cat_cols:
    print(data[col].value_counts(dropna=False))
    print('')
```

```
Торговля                                  2385
Другие сферы                              1709
NaN                                       1367
Металлургия/Промышленность/Машиностроение 1356
Государственная служба                    1286
Здравоохранение                           1177
Образование                                998
Транспорт                                  787
Сельское хозяйство                         702
Строительство                              574
Коммунальное хоз-во / Дорожные службы      533
Ресторанный бизнес / Общественное питание  408
```

```
Наука                                          403
Нефтегазовая промышленность                    225
Сборочные производства                         172
Банк/Финансы                                   169
Энергетика                                     145
Развлечения/Искусство                          141
ЧОП / Детективная д-ть                         136
Информационные услуги                          108
Салоны красоты и здоровья                       99
Информационные технологии                       85
Химия/Парфюмерия/Фармацевтика                   63
СМИ/Реклама/PR-агенства                         49
Юридические услуги / нотариальные услуги        47
Страхование                                     28
Туризм                                          20
Недвижимость                                    16
Управляющая компания                            12
Логистика                                       11
Подбор персонала                                 8
Маркетинг                                        4
Name: GEN_INDUSTRY, dtype: int64

Специалист                          7010
Рабочий                             3075
NaN                                 1367
Служащий                             904
Руководитель среднего звена          697
Работник сферы услуг                 563
Высококвалифиц. специалист           549
Руководитель высшего звена           427
Индивидуальный предприниматель       217
Другое                               177
Руководитель низшего звена           136
Военнослужащий по контракту           88
Партнер                               13
Name: GEN_TITLE, dtype: int64

Частная компания                    6523
Государственная комп./учреж.       6112
NaN                                 1367
Индивидуальный предприниматель       957
Некоммерческая организация           243
Частная ком. с инос. капиталом        21
Name: ORG_TP_STATE, dtype: int64

Без участия    13688
NaN             1365
С участием       170
Name: ORG_TP_FCAPITAL, dtype: int64

Участие в основ. деятельности    11452
NaN                               1367
Вспомогательный техперсонал       1025
Бухгалтерия, финансы, планир.      481
Адм-хоз. и трансп. службы          279
Снабжение и сбыт                   217
Служба безопасности                164
```

```
Кадровая служба и секретариат      101
Пр-техн. обесп. и телеком.          75
Юридическая служба                  53
Реклама и маркетинг                  9
Name: JOB_DIR, dtype: int64
```

Обработка редких категорий выполняется либо до разбиения на обучающую и тестовую выборки, либо после него в зависимости от причин, обусловивших появление таких категорий.

Если переменная содержит 2–3 редкие категории небольшой частоты, скорее всего, такие категории случайны и часто могли быть обусловлены очевидными ошибками ввода. В таком случае эти категории, как правило, объединяют с самой часто встречающейся категорией или по смыслу (например, у нас есть категории `John`, `Mike`, `Jack` и редкая категория `Jon`, последняя категория является, скорее всего, результатом ошибки ввода, и ее можно заменить на `John`). Это можно сделать до разбиения на обучающую и тестовую выборки.

Кроме того, у нас могут быть априорные знания, подтвержденные опытом и бизнес-логикой. Например, у нас есть редкие категории-регионы, поскольку мы в них редко выдаем кредиты из-за плохого процента погашений, эти регионы характеризуются нестабильной социально-экономической ситуацией (Республика Дагестан, Республика Адыгея, Чеченская республика) или на их территории находятся зоны, регистрируется высокий уровень преступности (Забайкальский край, Республика Мордовия). Мы можем отнести эти редко встречающиеся регионы к категории *Неблагополучные регионы*. Это тоже можно сделать до разбиения на обучающую и тестовую выборки.

Если таких априорных знаний нет, нам необходимо задать порог укрупнения – минимальное количество наблюдений в категории, ниже которого категория объявляется редкой. Поэтому для объективности решение о выборе такого порога должно приниматься уже после разбиения на обучающую и тестовую выборки. В противном случае получится, что решение о выборе порога мы принимали с учетом информации «из будущего». В данной ситуации чаще всего пишут собственный класс, в котором можно задавать порог укрупнения, и соответственно этот порог можно использовать в качестве гиперпараметра в сетке гиперпараметров при работе с классами `Pipeline` и `GridSearchCV`.

Множественные редкие категории часто объединяют в одну отдельную категорию, если подтверждается гипотеза о том, что редкие категории описывают определенный паттерн. Например, в кредитном скоринге укрупнение редких категорий в отдельную категорию нередко улучшает результат. Редкие типы кредитов могут соответствовать кредитам, выданным на эксклюзивных условиях, подобные кредиты выдаются людям с хорошей кредитной историей, и, таким образом, объединив редкие категории в отдельную группу, мы выделяем группу заемщиков с лучшим кредитным статусом.

Кроме того, применяется случайное присвоение редких категорий уже существующим категориям.

Ниже приведен пример выделения редких категорий признака *JOB_DIR* в отдельную категорию.

```
# записываем указанные категории переменной
# JOB_DIR в отдельную категорию OTHER
lst = ['Реклама и маркетинг', 'Юридическая служба']
data.loc[data['JOB_DIR'].isin(lst), 'JOB_DIR'] = 'OTHER'
# смотрим частоты
data['JOB_DIR'].value_counts(dropna=False)
```

```
Участие в основ. деятельности    11452
NaN                               1367
Вспомогательный техперсонал       1025
Бухгалтерия, финансы, планир.      481
Адм-хоз. и трансп. службы          279
Снабжение и сбыт                   217
Служба безопасности                164
Кадровая служба и секретариат      101
Пр-техн. обесп. и телеком.          75
OTHER                               62
Name: JOB_DIR, dtype: int64
```

Теперь давайте напишем собственный класс `RareGrouper`, с помощью которого можно задать порог укрупнения. Обратите внимание: здесь мы используем словарь со значением по умолчанию (`defaultdict`). Он похож на обычный словарь, за исключением одной особенности – при попытке обратиться к ключу, которого в нем нет, он сперва добавляет для него значение, используя функцию без аргументов, которая предоставляется при его создании (в нашем случае речь пойдет о функции `list()`). В методе `.fit()` для каждой переменной вычисляем относительные частоты категорий и получаем словарь вида

```
defaultdict(list,
            {'var1': ['категория выше порога', 'категория выше порога'],
             'var2': ['категория выше порога', 'категория выше порога']})
```

Обратите внимание, что нас всегда интересуют частоты категорий, вычисленные на обучающей выборке, таким образом, с порогом мы сравниваем частоты категорий, вычисленные на обучающей выборке.

```
# пишем класс, укрупняющий категории по порогу, категории
# с относительными частотами меньше порога запишем в
# отдельную категорию OTHER
class RareGrouper():
    """
    Параметры:
    threshold: int, значение по умолчанию 0.01
        Минимально допустимая относительная частота,
        при которой замены не происходит.
    """

    def __init__(self, threshold=0.01):

        self.d = defaultdict(list)
        self.threshold = threshold

    def fit(self, X, y=None):
```

```
        # для каждой переменной вычисляем относительные
        # частоты категорий и получаем словарь вида
        # defaultdict(
        #     list, {'var1': ['категория выше порога',
        #                     'категория выше порога'],
        #            'var2': ['категория выше порога',
        #                     'категория выше порога']})
        n_obs = len(X)
        for col in X.columns:
            rel_freq = X[col].value_counts(dropna=False) / n_obs
            self.d[col] = rel_freq[rel_freq >= self.threshold].index
        return self

    def transform(self, X):
        # создаем копию датафрейма
        X = X.copy()
        # для каждой переменной категории (строковые значения), которых нет
        # в соответствующем списке словаря (их частоты ниже порога),
        # относим к категории Other
        for col in X.columns:
            X[col] = np.where(X[col].isin(self.d[col]), X[col], 'Other')
        return X
```

Применяем класс `RareGrouper`, для переменной *JOB_DIR* категории менее 200 наблюдений запишем в категорию `OTHER`, порог равен 200 / 15 223 = 0,013.

```
# применяем класс, для переменной JOB_DIR категории
# менее 200 наблюдений запишем в категорию OTHER,
# порог равен 200 / 15 223 = 0.013
raregrouper = RareGrouper(threshold=0.013)
raregrouper.fit(data[['JOB_DIR']])
data['JOB_DIR'] = raregrouper.transform(data[['JOB_DIR']])
# смотрим частоты категорий
# переменной GEN_TITLE
data['JOB_DIR'].value_counts(dropna=False)
```

```
Участие в основ. деятельности    11452
NaN                               1367
Вспомогательный техперсонал       1025
Бухгалтерия, финансы, планир.      481
Other                              402
Адм-хоз. и трансп. службы          279
Снабжение и сбыт                   217
Name: JOB_DIR, dtype: int64
```

```
Участие в основ. деятельности    11452
NaN                               1367
Вспомогательный техперсонал       1025
Бухгалтерия, финансы, планир.      481
Адм-хоз. и трансп. службы          279
Снабжение и сбыт                   217   200
Служба безопасности                164   |
Кадровая служба и секретариат      101   |
Пр-техн. обесп. и телеком.          75   | 402
Юридическая служба                  53   |
Реклама и маркетинг                  9   |
Name: JOB_DIR, dtype: int64
```

С помощью метода CHAID можно выполнить укрупнение категорий и посмотреть, с какими из существующих категорий была объединена та или иная редкая категория.

CHAID (*Chi-square Automatic Interaction Detector – Автоматический обнаружитель взаимодействий*) был разработан Гордоном Каасом в 1980 году и представляет собой метод на основе дерева решений, который исследует взаимосвязь между признаками и зависимой переменной с помощью статистических тестов. Зависимая переменная может быть измерена в категориальной или количественной шкале. Признаки могут быть только категориальными пере-

менными (количественные переменные должны быть предварительно преобразованы в категориальные порядковые с помощью биннинга). На первом этапе категории каждого признака объединяются, если они не имеют между собой статистически значимых отличий по отношению к зависимой переменной (для категориальной зависимой переменной используется критерий хи-квадрат, для количественной зависимой переменной используется F-критерий). Категории, которые дают значимые отличия по зависимой переменной, рассматриваются как отдельные. На втором этапе для разбиения узла выбирается признак, сильнее всего взаимодействующий с зависимой переменной. Получив новые узлы, мы повторяем для каждого из них вышеописанные шаги. Этот процесс продолжается до тех пор, пока есть возможность создания новых узлов, т. е. пока не останется переменных, позволяющих получать узлы, максимально отличающиеся по зависимой переменной, или пока не сработают правила остановки. Если CHAID применять для укрупнения категорий конкретной переменной, то нам по сути требуется только первый этап.

В среде Python укрупнение категорий на основе метода CHAID можно выполнить с помощью пакета CHAID. Этот пакет можно установить с помощью команды `pip install CHAID`.

Выполним для категориальной переменной *GEN_TITLE* укрупнение категорий на основе CHAID.

```
# укрупняем категории с помощью CHAID
from CHAID import Tree
# задаем название признака
independent_variable = 'GEN_TITLE'
# задаем название зависимой переменной
dep_variable = 'TARGET'
# создаем словарь, где ключом будет название
# признака, а значением - тип переменной
dct = {independent_variable: 'nominal'}
# строим дерево CHAID и выводим его
tree = Tree.from_pandas_df(data, dct, dep_variable, max_depth=1) tree.print_tree()
```

```
([], {0: 13411.0, 1: 1812.0}, (GEN_TITLE, p=9.43237916410679e-29, score=133.51911498532897,
groups=[['<missing>'], ['Специалист', 'Руководитель среднего звена', 'Другое'], ['Высоко-
квалифиц. специалист', 'Служащий', 'Работник сферы услуг', 'Рабочий'], ['Военнослужащий по
контракту', 'Индивидуальный предприниматель', 'Руководитель высшего звена', 'Руководитель
низшего звена', 'Партнер']]), dof=3))
|-- (['<missing>'], {0: 1317.0, 1: 50.0}, <Invalid Chaid Split> - the max depth has been
reached)
|-- (['Специалист', 'Руководитель среднего звена', 'Другое'], {0: 6984.0, 1: 900.0}, <In-
valid Chaid Split> - the max depth has been reached)
|-- (['Высококвалифиц. специалист', 'Служащий', 'Работник сферы услуг', 'Рабочий'], {0:
4379.0, 1: 712.0}, <Invalid Chaid Split> - the max depth has been reached)
+-- (['Военнослужащий по контракту', 'Индивидуальный предприниматель', 'Руководитель высшего
звена', 'Руководитель низшего звена', 'Партнер'], {0: 731.0, 1: 150.0}, <Invalid Chaid Split>
- the max depth has been reached)
```

Первая строка вывода начинается с информации о частотах классов зависимой переменной `{0: 13411.0, 1: 1812.0}`, значение *p* показывает статистическую значимость, `score` показывает значение хи-квадрат (меньшее *p*-значение

говорит о более сильной взаимосвязи между признаком и зависимой переменной), `groups` показывает полученные категории. Далее приводятся узлы – укрупненные категории и распределение классов зависимой переменной в каждом узле. Также выводятся предупреждения, сообщающие, какое из правил остановки сработало: для всех узлов приводится предупреждение `the max depth has been reached` – достигнута максимальная глубина, это неудивительно, ведь мы выбрали глубину 1 (`max depth=1`, т. е. дерево будет иметь один уровень, лежащий ниже корневого узла). Мы видим, что редкая категория `Партнер` объединена с категориями `Военнослужащий по контракту`, `Индивидуальный предприниматель`, `Руководитель высшего звена`, `Руководитель низшего звена`. Укрупнение редких категорий с помощью CHAID необходимо выполнять после разбиения на обучающую и тестовую выборки или внутри цикла перекрестной проверки, поскольку CHAID использует для работы биннинг на основе децилей и применяет статистические критерии для принятия решения об объединении категорий.

11. Появление новых категорий в новых данных

Существует еще проблема появления новых категорий в новых данных. Например, мы разработали и внедрили скоринговую модель. К моменту внедрения модели маркетинговая или кредитная политика банка поменялась, и у нас в переменной *Сфера занятости* появилась новая категория `Няни, воспитательницы`.

В банках часто применяется консервативный подход: новая категория приравнивается к категории, демонстрирующей наибольший уровень риска, потому что мы ничего не знаем об этой категории клиентов и их возможном кредитном статусе. В других случаях новую категорию приравнивают к наиболее редкой категории или наиболее частой категории, способ зависит от априорных знаний, накопленного опыта.

Допустим, у нас в исторических данных есть переменная *pay*. У нее есть категории `CC`, `CH` и `AUTO`.

```
CC      2561
CH       977
Auto     889
Name: pay, dtype: int64
```

В новых данных появилась категория `CP`.

В функцию предобработки, которую мы будем применять к новым данным, добавим программный код, заменяющий все новые категории модой, т. е. категорией `CC`.

```
...
# все новые категории переменной pay заменяем модой
lst = ['CC', 'Auto', 'CH']
replace_new_values = lambda s: 'CC' if s not in lst else s
df['pay'] = df['pay'].map(replace_new_values)
...
```

12. Импутация пропусков

Существует три типа возникновения пропусков: MCAR, MAR, MNAR.

MCAR («совершенно случайно пропущенные» – Missing Completely At Random) – тип возникнования пропусков, при котором вероятность пропуска для каждого наблюдения набора одинакова. Вероятность пропуска значения для переменной X не связана ни со значением самой переменной X, ни со значениями других переменных в наборе данных. Например, переменная *Доход* подчиняется условию MCAR, если клиенты, которые не сообщают о своем доходе, имеют в среднем такой же размер дохода, что и клиенты, которые указывают свой доход.

MAR («случайно пропущенные» – Missing At Random) – тип возникновения пропусков, когда данные пропущены не случайно, а ввиду некоторых закономерностей. Вероятность пропуска значения для переменной X может быть объяснена другими имеющимися переменными, не содержащими пропуски. Например, переменная *Доход* подчиняется условию MAR, если вероятность пропуска данных в переменной *Доход* зависит от наблюдаемой переменной, например от переменной *Образование*. Например, клиенты с низким уровнем образования могут иметь большее количество пропущенных значений дохода (т. е. чаще, чем другие респонденты, не отвечают на вопрос о доходе). Необходимо проанализировать взаимосвязь между переменной *Доход* и переменной *Образование*.

MNAR («не случайно пропущенные» – Missing Not At Random) – тип пропущенных данных, когда пропуск значения не является совершенно случайным и не может быть полностью объяснен другими переменными в наборе. Пропущенные значения остаются зависимыми от неизвестных нам факторов, необходимо провести дополнительные исследования. Здесь можно привести вышеописанный случай с пропусками в переменной *Доход*, но только теперь переменная *Образование* у нас отсутствует.

12.1. Способы импутации количественных и бинарных переменных

Пропуски в количественных переменных заменяются вычисленными статистиками, обычно используется среднее или медиана. В случае данных, имеющих асимметричное распределение, предпочитают использовать медиану, а не среднее, так как на нее не влияет небольшое число наблюдений с очень большими или очень маленькими значениями.

Вновь обратите внимание, что импутацию средним, медианой и прочими статистиками необходимо выполнять после разбиения набора данных на обучающую и тестовую выборки (внутри цикла перекрестной проверки).

Помимо импутации средним или медианой, пропуски можно заменить значениями-константами. Для древовидных алгоритмов эффективной может быть импутация значением вне диапазона имеющихся значений. Если импутировать значением, которое будет больше любого имеющегося значения, то в дереве можно будет выбрать такое разбиение по этому признаку, что все наблюдения с известными значениями пойдут в левый узел, а все наблюде-

ния с пропусками – в правый. Если же импутировать значением, которое будет меньше любого имеющегося значения, то в дереве можно будет выбрать такое разбиение по этому признаку, что все наблюдения с пропусками пойдут в левый узел, а все наблюдения с известными значениями – в правый.

Часто применяют индикатор пропусков для соответствующей переменной. Он принимает значение 1, если переменная имеет пропуск, или 0, если переменная не содержит пропуск. В рамках бизнес-подхода индикатор пропусков носит временный характер. Мы строим уравнение регрессии и смотрим, является ли коэффициент для данного индикатора значимым. Если коэффициент значим, то выбор способа импутации признака, для которого создавался индикатор, может существенно повлиять на качество модели. Если коэффициент не является значимым, то выбор способа импутации признака, для которого создавался индикатор, не повлияет существенно на качество модели. Это необходимо для приоритизации операций импутации для десятков-сотен признаков. Допустим, у нас 200 признаков с пропусками, нужно выяснить, для каких признаков имеет смысл пробовать разные способы импутации, а каким признакам будет достаточно импутации медианой или средним.

Выполнить импутацию константами и создать индикаторы пропусков можно (и нужно, чтобы не повторять одни и те же операции для двух наборов данных) до разбиения на обучение и тест (до перекрестной проверки), потому что в рамках этой операции мы не делаем вычислений, охватывающих все наблюдения исходного набора.

Бинарные переменные, у которых есть пропуски, можно превратить в тринарные, где первую категорию можно закодировать как –1, вторую категорию – как 1, а пропуски – как 0.

Количественная переменная	Бинарная переменная
NA	1
6	0
5	1
8	NA
0	0
NA	0
9	1
3	NA
5	0

Количественная переменная				Бинарная переменная
медиана	среднее	-999	индикатор пропусков	
5	5.14	-999	1	1
6	6	6	0	-1
5	5	5	0	1
8	8	8	0	0
0	0	0	0	-1
5	5.14	-999	1	-1
9	9	9	0	1
3	3	3	0	0
5	5	5	0	-1

данные способы импутации можно применить только после разбиения на обучение и тест (внутри цикла перекрестной проверки)

данные способы импутации можно применять до разбиения на обучение и тест (до цикла перекрестной проверки)

Рис. 12 Способ импутации количественных признаков

12.2. Способы импутации категориальных переменных

Пропуски в категориальных переменных можно заменить самой часто встречающейся категорией – модой. Обратите внимание: мод может быть несколько, и в программном коде вы должны указать, какую моду хотите использовать. Также заметьте, что поскольку мы используем вычисления, импутацию

модой можно осуществлять только после разбиения на обучение и тест (внутри цикла перекрестной проверки).

Пропуски в категориальных переменных часто кодируют отдельной категорией для пропусков, а также, как и в случае с количественными переменными, создают индикаторы пропусков (кроме случаев, когда категориальная переменная преобразуется в набор бинарных признаков с помощью дамми-кодирования).

Закодировать пропуски отдельной категорией и создать индикаторы можно (и нужно, чтобы не повторять одни и те же операции для двух наборов данных) до разбиения на обучение и тест (до перекрестной проверки), потому что в рамках этой операции мы не делаем вычислений, охватывающих все наблюдения исходного набора.

Рис. 13 Способ импутации категориальных признаков

В первой части мы уже познакомились с классом `SimpleImputer`, который часто применяется, когда нужно импутировать разные списки признаков. Для импутации пропусков можно также воспользоваться методом `.fillna()`. Для создания индикаторов пропусков используют функцию `np.where()`. Функция имеет три аргумента: первый аргумент – проверяемое условие, второй аргумент – что возвращать, если проверяемое условие верно, третий аргумент – что возвращать, если проверяемое условие неверно. Кроме того, в библиотеке scikit-learn есть класс `MissingIndicator`.

Таблица 2 Методы pandas для импутации пропусков

импутация средним или медианой	`tr['income'].fillna(tr['income'].mean(), inplace=True)`
	`tst['income'].fillna(tr['income'].mean(), inplace=True)`
	
	`tr['income'].fillna(tr['income'].median(), inplace=True)`
	`tst['income'].fillna(tr['income'].median(), inplace=True)`

импутация модой	`tr['pay'].fillna(tr['pay'].value_counts().index[0], inplace=True)` `tst['pay'].fillna(tr['pay'].value_counts().index[0], inplace=True)`
индикатор пропусков	`data['pay_ind'] = np.where(data['pay'].isnull(), 1, 0)`
импутация константой	`data['income'].fillna(-999, inplace=True)`

12.3. Практика

Давайте импортируем необходимые библиотеки, классы и функции и загрузим данные Verizon.

```
# импортируем библиотеки pandas, numpy, missingno,
# SimpleImputer, функции  train_test_split()
# и display()
import pandas as pd
import numpy as np
import missingno as msno
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from IPython.display import display

# включаем режим 'retina', если у вас экран Retina
%config InlineBackend.figure_format = 'retina'

# записываем CSV-файл в объект DataFrame
data = pd.read_csv('Data/Verizon_missing.csv', sep=';')
# выведем первые 3 наблюдения
data.head()
```

	longdist	internat	local	age	income	billtype	pay	churn
0	16.0	0.0	5.0	46.0	34805.5	Бюджетный	CC	0
1	0.0	0.0	5.0	59.0	60111.8	NaN	NaN	1
2	13.0	0.0	NaN	NaN	13126.9	Бюджетный	Auto	0
3	NaN	0.0	10.0	36.0	NaN	NaN	CH	0
4	NaN	NaN	10.0	38.0	17499.2	Бесплатный	Auto	0

Для вывода информации о пропусках используем цепочку методов `.isnull()` и `.sum()`.

```
# смотрим количество пропусков по каждой переменной
data.isnull().sum()
```

```
longdist     8
internat    34
local       14
age          8
income      29
billtype    24
pay         15
churn        0
dtype: int64
```

С помощью цепочки методов `.isnull()`, `.sum()` и `.sum()` можно вывести общее количество пропусков в наборе.

```
# общее количество пропусков
data.isnull().sum().sum()
```

```
132
```

Разбиваем набор на обучающую и тестовую выборки.

```
# разбиваем данные на обучающие и тестовые: получаем обучающий
# массив признаков, тестовый массив признаков, обучающий массив
# меток, тестовый массив меток
train, test, y_train, y_test = train_test_split(
    data.drop('churn', axis=1),
    data['churn'],
    test_size=.3,
    stratify=data['churn'],
    random_state=100)
```

Давайте выведем частоты категорий в признаке *pay* в обучающей и тестовой выборках.

```
# смотрим частоты категорий признака pay
print("TRAIN:")
print(train['pay'].value_counts(dropna=False))
print("")
print("TEST:")
print(test['pay'].value_counts(dropna=False))
```

```
TRAIN:
CC      42
CH      25
Auto    19
NaN     13
CD       1
Name: pay, dtype: int64

TEST:
CC      22
```

```
CH      14
Auto     5
NaN      2
CD       1
Name: pay, dtype: int64
```

Импутируем пропуски в признаке *pay* модой, вычисленной на обучающей выборке, и снова выведем частоты категорий в признаке *pay* в обучающей и тестовой выборках.

```
# выполняем импутацию модой
train['pay'].fillna(train['pay'].value_counts().index[0],
                    inplace=True)
test['pay'].fillna(train['pay'].value_counts().index[0],
                   inplace=True)

# смотрим частоты категорий признака pay
print("TRAIN:")
print(train['pay'].value_counts(dropna=False))
print("")
print("TEST:")
print(test['pay'].value_counts(dropna=False))
```

```
TRAIN:
CC      55
CH      25
Auto    19
CD       1
Name: pay, dtype: int64

TEST:
CC      24
CH      14
Auto     5
CD       1
Name: pay, dtype: int64
```

Видим, что самая частая категория – категория `'CC'` – увеличилась в размере за счет того, что пропускам была присвоена эта категория.

Теперь импутируем пропуски в признаке *Income* средним значением, вычисленным в обучающей выборке.

```
# печатаем среднее признака income
print(f"среднее значение income: {train['income'].mean()}")
# печатаем значение признака income
# в строке с индексом 4
print(train['income'].iloc[4])
print(test['income'].iloc[4])
# выполняем импутацию средним
train['income'].fillna(train['income'].mean(),
                       inplace=True)
test['income'].fillna(train['income'].mean(),
                      inplace=True)
# печатаем значение признака income
# в строке с индексом 4
```

```
print(train['income'].iloc[4])
print(test['income'].iloc[4])
```

```
среднее значение income: 46635.54195121954
nan
nan
46635.54195121954
46635.54195121953
```

С помощью функции `display()` выведем первые пять наблюдений обучающей и тестовой выборок.

```
# взглянем на первые пять наблюдений каждой выборки
display(train.head())
display(test.head())
```

	longdist	internat	local	age	income	billtype	pay
18	19.0	NaN	11.0	88.0	66906.600000	Бесплатный	CH
19	19.0	NaN	96.0	79.0	37571.100000	Бесплатный	CC
66	0.0	0.0	1.0	94.0	13946.800000	Бюджетный	CC
65	9.0	6.0	12.0	93.0	46771.200000	Бюджетный	CC
74	26.0	0.0	NaN	32.0	46635.541951	Бюджетный	Auto

	longdist	internat	local	age	income	billtype	pay
44	3.0	0.0	9.0	45.0	70239.800000	Бесплатный	CC
59	21.0	NaN	NaN	33.0	60170.100000	NaN	CH
51	29.0	NaN	110.0	29.0	18861.600000	Бюджетный	CC
35	12.0	0.0	55.0	62.0	13965.400000	Бесплатный	CH
55	25.0	0.0	77.0	95.0	46635.541951	NaN	Auto

Пропуски в признаке *local* импутируем групповыми средними. Мы воспользуемся средними значениями признака *local*, вычисленными по категориям признака *pay* в обучающей выборке. Для этого напишем класс `GroupImputer`, выполняющий импутацию групповыми статистиками.

```
# пишем класс, выполняющий импутацию
# групповыми статистиками
class GroupImputer():
    """
    Автор: Eryk Lewinson
    https://github.com/erykml

    Класс, выполняющий импутацию групповыми статистиками.

    Параметры
    ----------
    group_cols: list
        Список группирующих столбцов.
    agg_col: str
        Агрегируемый столбец.
```

```
    agg_func: str
        Агрегирующая функция.
    """
    def __init__(self, group_cols, agg_col, agg_func='mean', return_df=True):

        if agg_func not in ['mean', 'median', 'min', 'max']:
            raise ValueError(f"Неизвестная агрегирующая функция {agg_func}")

        if type(group_cols) != list:
            raise ValueError("Задайте список группирующих столбцов")

        if type(agg_func) != str:
            raise ValueError("Агрегирующая функция должна" +
                             "иметь строковое значение")

        self.group_cols = group_cols
        self.agg_col = agg_col
        self.agg_func = agg_func
        self.return_df = return_df

    def fit(self, X, y=None):

        # проверка наличия пропусков в группирующих столбцах
        if pd.isnull(X[self.group_cols]).any(axis=None) == True:
             raise ValueError("Есть пропуски в группирующих столбцах")

        # получение датафрейма с групповыми статистиками
        self.impute_map_ = X.groupby(self.group_cols)[self.agg_col].agg(
            self.agg_func).reset_index(drop=False)

        return self

    def transform(self, X, y=None):

        X = X.copy()

        # заполнение пропусков с помощью датафрейма
        # с групповыми статистиками
        for index, row in self.impute_map_.iterrows():
            ind = (X[self.group_cols] == row[self.group_cols]).all(axis=1)
            X.loc[ind, self.agg_col] = X.loc[ind, self.agg_col].fillna(
                row[self.agg_col])

        if not self.return_df:
            X = X.values

        return X
```

Применяем наш класс.

```
# выполняем импутацию пропусков признака local групповыми
# средними - средними значениями признака local,
# вычисленными по категориям признака pay
imp = GroupImputer(['pay'], agg_col='local', agg_func='mean')
imp.fit(train)
train = imp.transform(train)
test = imp.transform(test)
```

Вновь выведем первые пять наблюдений обучающей и тестовой выборок.

```
# взглянем на первые пять наблюдений каждой выборки
display(train.head())
display(test.head())
```

	longdist	internat	local	age	income	billtype	pay
18	19.0	NaN	11.000000	88.0	66906.600000	Бесплатный	CH
19	19.0	NaN	96.000000	79.0	37571.100000	Бесплатный	CC
66	0.0	0.0	1.000000	94.0	13946.800000	Бюджетный	CC
65	9.0	6.0	12.000000	93.0	46771.200000	Бюджетный	CC
74	26.0	0.0	31.666667	32.0	46635.541951	Бюджетный	Auto

self.impute_map_

	pay	local
0	Auto	31.666667
1	CC	59.551020
2	CD	9.000000
3	CH	65.130435

	longdist	internat	local	age	income	billtype	pay
44	3.0	0.0	9.000000	45.0	70239.800000	Бесплатный	CC
59	21.0	NaN	65.130435	33.0	60170.100000	NaN	CH
51	29.0	NaN	110.000000	29.0	18861.600000	Бюджетный	CC
35	12.0	0.0	55.000000	62.0	13965.400000	Бесплатный	CH
55	25.0	0.0	77.000000	95.0	46635.541951	NaN	Auto

Для импутации пропусков в категориальных признаках можно применить вышеупомянутый метод CHAID. Он рассматривает пропуски как отдельную категорию и сравнивает ее с существующими категориями. Если категория пропусков и существующая категория не имеют между собой статистически значимых отличий по отношению к зависимой переменной, они объединяются. Таким образом, можно попробовать заменить пропуски той категорией, с которой они были объединены.

Давайте взглянем на частоты категорий переменной *billtype* в обучающей выборке.

```
# смотрим частоты категорий признака billtype
train['billtype'].value_counts(dropna=False)
```

```
Бюджетный      50
Бесплатный     37
NaN            13
Name: billtype, dtype: int64
```

```
# выполняем импутацию с помощью CHAID
from CHAID import Tree
# сконкатенируем обучающий массив признаков и массив меток
train_data = pd.concat([train, y_train], axis=1)
# задаем название признака
independent_variable = 'billtype'
# задаем название зависимой переменной
dep_variable = 'churn'
# создаем словарь, где ключом будет название
# признака, а значением - тип переменной
dct = {independent_variable: 'nominal'}
```

```
# строим дерево CHAID и выводим его
tree = Tree.from_pandas_df(data, dct, dep_variable, max_depth=1)
tree.print_tree()
```

```
([], {0: 78.0, 1: 66.0}, (billtype, p=0.01635323632289041, score=5.764489380588449,
groups=[['<missing>', 'Бесплатный'], ['Бюджетный']]), dof=1))
|-- (['<missing>', 'Бесплатный'], {0: 34.0, 1: 42.0}, <Invalid Chaid Split> - the max depth
has been reached)
+-- (['Бюджетный'], {0: 44.0, 1: 24.0}, <Invalid Chaid Split> - the max depth has been
reached)
```

Видим, что пропуски были объединены с категорией `'Бесплатный'`, поэтому пропуски можно импутировать данной категорией.

Импутацию пропусков с помощью CHAID необходимо выполнять после разбиения на обучающую и тестовую выборки или внутри цикла перекрестной проверки, поскольку CHAID использует для работы биннинг на основе децилей и статистические критерии.

Основная рекомендация по работе с пропусками сводится к тому, чтобы всегда руководствоваться здравым смыслом и бизнес-логикой, не надеясь на сложные методы импутации. Приведем примеры применения логического контроля при работе с пропусками.

Необходимые нам данные записаны в файле *Credit_OTP_short.csv*. Исходная выборка содержит записи о 15 223 клиентах, классифицированных на два класса: 0 – отклика не было (13 411 клиентов) и 1 – отклик был (1812 клиентов). По каждому наблюдению (клиенту) фиксируются следующие переменные.

Список исходных переменных включает в себя:

- категориальный признак *Уникальный идентификатор объекта в выборке [AGREEMENT_RK]*;
- бинарная зависимая переменная *Отклик на маркетинговую кампанию [TARGET]*;
- количественный признак *Возраст клиента [AGE]*;
- категориальный признак *Социальный статус клиента относительно работы [SOCSTATUS_WORK_FL]*;
- категориальный признак *Социальный статус клиента относительно пенсии [SOCSTATUS_PENS_FL]*;
- категориальный признак *Пол клиента [GENDER]*;
- количественный признак *Количество детей клиента [CHILD_TOTAL]*;
- количественный признак *Количество иждивенцев клиента [DEPENDANTS]*;
- категориальный признак *Образование [EDUCATION]*;
- категориальный признак *Семейное положение [MARITAL_STATUS]*;
- категориальный признак *Отрасль работы клиента [GEN_INDUSTRY]*;
- категориальный признак *Должность [GEN_TITLE]*;
- категориальный признак *Форма собственности компании [ORG_TP_STATE]*;
- категориальный признак *Отношение к иностранному капиталу [ORG_TP_FCAPITAL]*;
- категориальный признак *Направление деятельности внутри компании [JOB_DIR]*;
- категориальный признак *Семейный доход [FAMILY_INCOME]*;

- количественный признак *Личный доход клиента в рублях [PERSONAL_INCOME]*;
- категориальный признак *Область регистрации клиента [REG_ADDRESS_PROVINCE]*;
- категориальный признак *Область фактического пребывания клиента [FACT_ADDRESS_PROVINCE]*;
- категориальный признак *Почтовый адрес область [POSTAL_ADDRESS_PROVINCE]*;
- категориальный признак *Область торговой точки, где клиент брал последний кредит [TP_PROVINCE]*;
- категориальный признак *Регион РФ [REGION_NM]*;
- количественный признак *Сумма последнего кредита клиента в рублях [CREDIT]*;
- количественный признак *Первоначальный взнос в рублях [FST_PAYMENT]*;
- количественный признак *Количество месяцев проживания по месту фактического пребывания [FACT_LIVING_TERM]*;
- количественный признак *Время работы на текущем месте в месяцах [WORK_TIME]*.

```
# увеличиваем количество выводимых столбцов
pd.set_option('display.max_columns', 60)
# загружаем набор данных
data = pd.read_csv('Data/Credit_OTP_short.csv', sep=';')
# выводим первые пять наблюдений
data.head()
```

	AGREEMENT_RK	TARGET	AGE	SOCSTATUS_WORK_FL	SOCSTATUS_PENS_FL	GENDER	CHILD_TOTAL	DEPENDANTS	EDUCATION	MARITAL_STATUS
0	59910150	0	49	1	0	1	2	1	Среднее специальное	Состою в браке
1	59910230	0	32	1	0	1	3	3	Среднее	Состою в браке
2	59910525	0	52	1	0	1	4	0	Неполное среднее	Состою в браке
3	59910803	0	39	1	0	1	1	1	Высшее	Состою в браке
4	59911781	0	30	1	0	0	0	0	Среднее	Состою в браке

Выясним, есть ли у нас пропуски.

```
# выводим информацию о пропусках
data.isnull().sum()
```

```
AGREEMENT_RK                 0
TARGET                       0
AGE                          0
SOCSTATUS_WORK_FL            0
SOCSTATUS_PENS_FL            0
GENDER                       0
CHILD_TOTAL                  0
DEPENDANTS                   0
EDUCATION                    0
MARITAL_STATUS               0
GEN_INDUSTRY              1367
GEN_TITLE                 1367
ORG_TP_STATE              1367
ORG_TP_FCAPITAL           1365
JOB_DIR                   1367
FAMILY_INCOME                0
PERSONAL_INCOME              0
REG_ADDRESS_PROVINCE         0
FACT_ADDRESS_PROVINCE        0
POSTAL_ADDRESS_PROVINCE      0
TP_PROVINCE                295
REGION_NM                    1
CREDIT                       0
FST_PAYMENT                  0
FACT_LIVING_TERM             0
WORK_TIME                 1368
dtype: int64
```

Мы видим, что у нас наблюдается пропуск практически одинакового количества значений по категориальным переменным *GEN_ INDUSTRY*, *GEN_TITLE*, *ORG_TP_STATE*, *ORG_TP_FCAPITAL*, *JOB_DIR* и количественной переменной *WORK_ TIME*.

Теперь воспользуемся удобной библиотекой missingno для визуализации пропусков (в виде белых горизонтальных линий).

```
# визуализируем пропуски с помощью missingno
msno.matrix(data, sparkline=False, figsize=(11, 11));
```