**Предварительная обработка данных**

План:

[Что такое машинное обучение? 2](#_30j0zll)

[История машинного обучения 2](#_1fob9te)

[Почему машинное обучение важно для прогнозирования? 3](#_3znysh7)

[Обзор главных методов машинного обучения 4](#_2et92p0)

[Предобработка данных (Предварительная обработка данных) 6](#_tyjcwt)

[1) Генерация и сбор данных 7](#_3dy6vkm)

[1) Генерация датасеть 8](#_1t3h5sf)

[2) Загрузка данных 27](#_4d34og8)

[3) Загрузка датасет из Kaggle 38](#_2s8eyo1)

[2) Разделите набор данных на зависимую и независимую переменную 40](#_17dp8vu)

[2) Проверить пропущенные значения 43](#_3rdcrjn)

[3) Проверить категориальные значения 48](#_26in1rg)

[4) Разделите набор данных на обучающий набор и тестовый набор 52](#_lnxbz9)

[5) Масштабирование функций 53](#_35nkun2)

[Задачи для самостоятельной работы 55](#_1ksv4uv)

[Литература 57](#_44sinio)

**Что такое машинное обучение?**

Машинное обучение - это подход к искусственному интеллекту, который позволяет компьютерным системам изучать и анализировать данные, не явно программированные для этого. Вместо того, чтобы человек явно задавал инструкции компьютеру, как обрабатывать данные, машинное обучение позволяет компьютеру самостоятельно находить закономерности и обучаться на основе этих данных.

Существует несколько типов машинного обучения, включая обучение с учителем, обучение без учителя и обучение с подкреплением.

В обучении с учителем, компьютер обучается на основе предоставленных ему данных и соответствующих им меток. Например, если мы хотим обучить компьютер распознавать изображения кошек и собак, мы предоставляем ему множество изображений кошек и собак, а также метки, указывающие, какое изображение относится к кошке, а какое - к собаке. Компьютер использует эти данные для создания модели, которая может распознавать кошек и собак на новых изображениях.

В обучении без учителя, компьютер самостоятельно находит закономерности в данных без предоставления меток. Например, если у нас есть множество покупок клиентов в интернет-магазине, мы можем использовать машинное обучение без учителя для выявления групп товаров, которые часто покупают вместе.

В обучении с подкреплением, компьютер обучается на основе опыта взаимодействия с окружающей средой. Например, мы можем использовать машинное обучение с подкреплением для создания робота-манипулятора, который будет учиться выполнению задачи на основе опыта взаимодействия с окружающей средой.

Машинное обучение находит широкое применение в различных областях. Например, в медицине его можно использовать для анализа медицинских данных и диагностики заболеваний. В финансовой сфере - для прогнозирования курсов валют и цен на акции. В транспортной отрасли - для оптимизации маршрутов и повышения безопасности дорожного движения. В производстве - для оптимизации производственных процессов и предотвращения аварийных ситуаций.

**История машинного обучения**

Машинное обучение - это область искусственного интеллекта, которая позволяет компьютерам обучаться на основе опыта и данных, не явно заданных программистом. Эта технология имеет давнюю историю, начиная с конца XIX века.

**Ранние работы**

В 1950-х годах Алан Тьюринг предложил идею создания машины, которая могла бы обучаться на основе опыта. Однако на тот момент компьютеры были слишком медленными, чтобы реализовать эту идею.

В 1956 году Джон Маккарти создал термин "искусственный интеллект" и организовал конференцию по этой теме. На этой конференции было представлено несколько алгоритмов машинного обучения, включая персептрон Фрэнка Розенблатта.

**Период затишья**

В 1960-х годах искусственный интеллект пережил период затишья, связанный с недостатком вычислительных ресурсов и недостаточной эффективностью алгоритмов машинного обучения.

**Возрождение**

В 1980-х годах машинное обучение начало возрождаться благодаря развитию компьютерной техники и новым алгоритмам, таким как метод опорных векторов и нейронные сети.

**Современность**

Сегодня машинное обучение находится на пике своей популярности. Большие объемы данных и доступность мощных вычислительных ресурсов позволяют создавать более сложные модели машинного обучения и применять их в различных областях, таких как медицина, финансы, транспорт и другие.

Таким образом, история машинного обучения свидетельствует о том, что эта технология имеет давнюю историю и продолжает развиваться. Сегодня машинное обучение является одной из самых перспективных областей искусственного интеллекта, которая имеет широкие возможности для применения во многих сферах деятельности.

**Почему машинное обучение важно для прогнозирования?**

Машинное обучение - это подход к искусственному интеллекту, который позволяет компьютерным системам изучать и анализировать данные, не явно программированные для этого. Одним из важных применений машинного обучения является прогнозирование.

Прогнозирование - это процесс предсказания будущих событий на основе имеющихся данных. В различных областях, таких как финансы, маркетинг, производство и транспорт, прогнозирование играет важную роль в принятии решений. Например, финансовые аналитики используют прогнозирование для прогнозирования курсов валют и цен на акции, а производственные инженеры - для оптимизации производственных процессов и предотвращения аварийных ситуаций.

Машинное обучение позволяет создавать модели, которые могут использоваться для прогнозирования на основе имеющихся данных. Эти модели могут быть обучены на больших объемах данных, что позволяет им выявлять скрытые закономерности и предсказывать будущие события с высокой точностью.

Например, в финансовой сфере машинное обучение может использоваться для прогнозирования курсов валют и цен на акции. Модели машинного обучения могут быть обучены на исторических данных о курсах валют и ценах на акции, а затем использоваться для прогнозирования будущих значений. Это позволяет финансовым аналитикам принимать более точные решения на основе прогнозов, что может привести к повышению прибыльности инвестиций.

В производственной сфере машинное обучение может использоваться для прогнозирования аварийных ситуаций. Модели машинного обучения могут быть обучены на исторических данных о производственных процессах и аварийных ситуациях, а затем использоваться для прогнозирования вероятности возникновения аварийных ситуаций в будущем. Это позволяет инженерам принимать меры по предотвращению аварийных ситуаций и повышению безопасности производственных процессов.

В маркетинге машинное обучение может использоваться для прогнозирования поведения потребителей. Модели машинного обучения могут быть обучены на исторических данных о покупках потребителей и других факторах, влияющих на их поведение, а затем использоваться для прогнозирования будущих покупок и предпочтений потребителей. Это позволяет маркетологам принимать более эффективные решения по продвижению продуктов и услуг.

Таким образом, машинное обучение играет важную роль в прогнозировании в различных областях. Оно позволяет создавать модели, которые могут предсказывать будущие события на основе имеющихся данных с высокой точностью. Это помогает принимать более эффективные

**Обзор главных методов машинного обучения**

Как отметили ранее, машинное обучение - это процесс обучения компьютерных систем на основе данных, с целью автоматического улучшения их производительности в выполнении задач. Существует несколько методов машинного обучения, которые можно разделить на три категории: обучение с учителем, обучение без учителя и обучение с подкреплением.

**Обучение с учителем**

Обучение с учителем - это метод машинного обучения, при котором модель обучается на основе размеченных данных. Данные состоят из входных и выходных значений, и модель пытается выучить зависимость между ними. Некоторые из наиболее распространенных методов обучения с учителем включают в себя:

* Линейная регрессия
* Логистическая регрессия
* Решающие деревья
* Случайный лес
* Градиентный бустинг

**Обучение без учителя**

Обучение без учителя - это метод машинного обучения, при котором модель обучается на основе неразмеченных данных. Это означает, что модель не знает правильных ответов и пытается выявить закономерности в данных самостоятельно. Некоторые из наиболее распространенных методов обучения без учителя включают в себя:

* Кластеризация
* Понижение размерности
* Ассоциативные правила
* Аномальное обнаружение

**Обучение с подкреплением**

Обучение с подкреплением - это метод машинного обучения, при котором модель обучается на основе опыта взаимодействия с окружающей средой. Модель получает награду или штраф за каждое действие, которое она выполняет, и пытается максимизировать свою награду. Некоторые из наиболее распространенных методов обучения с подкреплением включают в себя:

* Q-обучение
* Динамическое программирование
* Методы Монте-Карло

Таким образом, машинное обучение - это быстро развивающаяся область, и каждый из методов обучения имеет свои преимущества и недостатки. Выбор подходящего метода зависит от конкретной задачи и доступных данных. Я надеюсь, что этот краткий обзор поможет вам лучше понять различные методы машинного обучения и выбрать подходящий для вашей задачи.

**Предобработка данных (Предварительная обработка данных)**

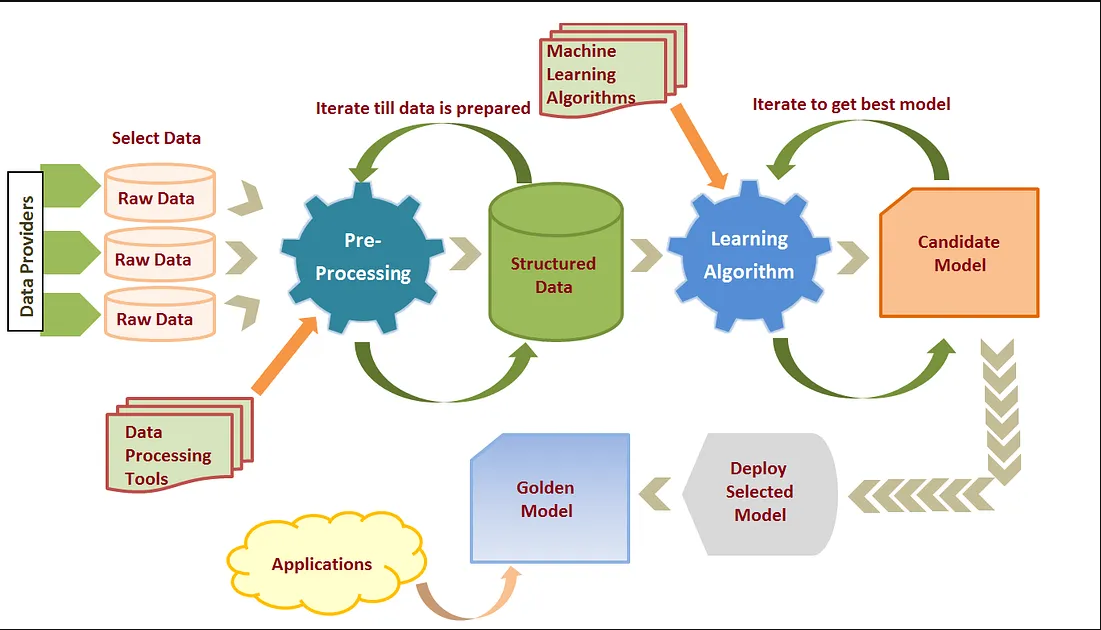
Машинное обучение сейчас в тренде. Каждый бизнес хочет развернуть модель машинного обучения для своего бизнес-кейса. Это дало нам интеллектуальных роботов, беспилотные автомобили, виртуальных личных помощников, таких как Siri и помощник Google и т. д. Помните, что вы купили что-то пару дней назад, и вы продолжаете получать предложения или рекомендации по покупкам на основе вашей недавней покупки? Или, может быть, вы когда-нибудь получали уведомление от Facebook, в котором говорилось: «Ваш друг добавил фотографию, на которой может быть вы!»

Все это возможно благодаря машинному обучению. Но в основе всего этого лежат Данные! Данные играют решающую роль в создании модели машинного обучения. Ваши сложные приложения для машинного обучения не могут быть построены на основе плохих данных.

Прежде чем необработанные данные можно будет отправить через модель машинного обучения, они должны пройти предварительную обработку. И это просто потому, что данные в реальном мире, как правило, неполные, зашумленные и непоследовательные . И если это ввести в модель машинного обучения, результаты могут прийти неожиданно! И это не совсем то, чего мы хотим. Предварительная обработка данных является проверенным методом решения таких проблем.

Предварительная обработка данных обычно выполняется в 7 простых шагов:

* Сбор данных
* Импорт набора данных и библиотек
* Разделите набор данных на зависимую и независимую переменную
* Проверить пропущенные значения
* Проверить категориальные значения
* Разделите набор данных на обучающий и тестовый наборы
* Масштабирование функций



Процесс машинного обучения (imarticus.org)

1. **Генерация и сбор данных**

Данные — это необработанная информация, представляющая как человеческое, так и машинное наблюдение за миром. Набор данных полностью зависит от того, какую проблему вы хотите решить. Каждая проблема в машинном обучении имеет свой уникальный нюанс и подход. И иногда может быть довольно сложно найти конкретный набор данных для множества задач машинного обучения. Но хорошая новость в том, что есть набор данных для каждой проблемы, о которой вы только можете подумать. Вот список веб-сайтов, с которых вы можете получить наборы данных:

* Kaggle: Kaggle — мой личный фаворит для получения набора данных.  
  <https://www.kaggle.com/datasets>
* [Репозиторий машинного обучения UCI](http://mlr.cs.umass.edu/ml/) : один из старейших источников наборов данных в Интернете и отличная первая остановка при поиске интересных наборов данных.  
  <http://mlr.cs.umass.edu/ml/>
* В этом потрясающем репозитории GitHub есть множество ссылок на высококачественные наборы данных.

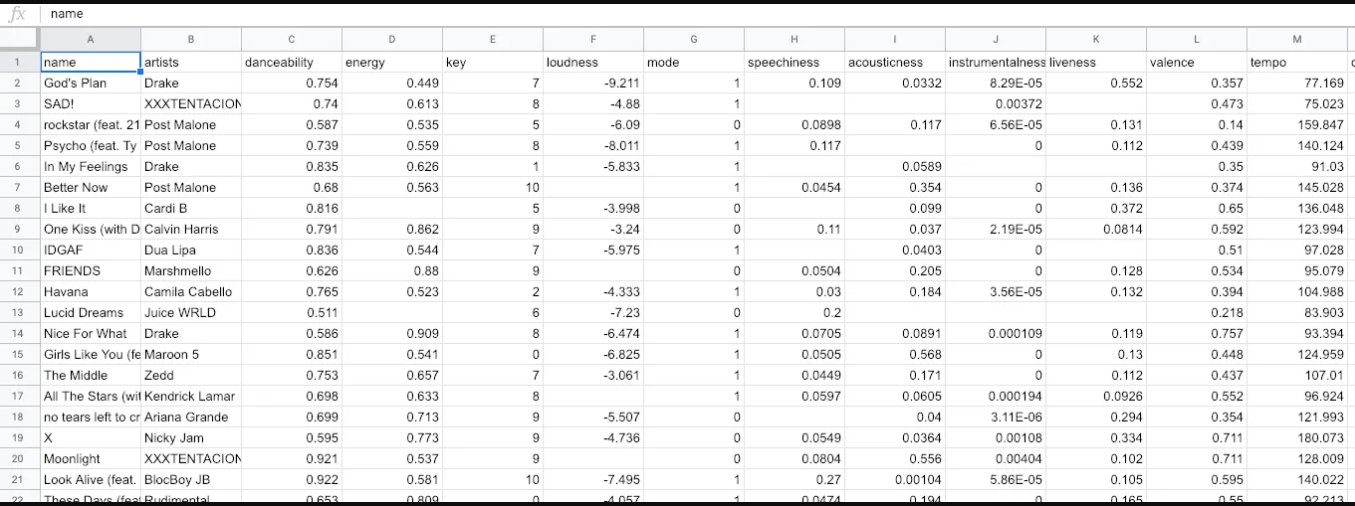
Из их GitHub readme : этот список тематически-ориентированных общедоступных источников данных в высоком качестве. Они собираются и упорядочиваются из блогов, ответов и ответов пользователей. Большинство наборов данных, перечисленных ниже, бесплатны, однако некоторые платные.

Ссылка: <https://github.com/awesomedata/awesome-public-datasets>

* Есть еще Zanran Numerical Data Search Engine! <http://www.zanran.com/q/>
* И если вы ищете открытые данные правительства, то вот некоторые из них:
* Правительство Индии: [http://data.gov.in](http://data.gov.in/)
* Правительство США: <https://www.data.gov/>
* Правительство Великобритании: [https://data. gov.uk/](https://data.gov.uk/)
* Правительство Франции: <https://www.data.gouv.fr/en/>
* И, конечно же, поиск в Google!

***Важно!*** *Пожалуйста, посетите этот вопрос Quora:*[*https://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public*](https://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public)

* Вот набор данных, который я скачал и буду использовать с Kaggle « Лучшие треки Spotify 2018 — Аудиофункции лучших песен Spotify ».  
  Вы можете скачать его с <https://www.kaggle.com/nadintamer/top-spotify-tracks-of-2018>  
  Так выглядит набор данных



Теперь я намеренно удалил некоторые значения, чтобы показать, как обрабатывать отсутствующие значения. Помните, я говорил вам, что реальные данные бывают неполными, зашумленными и непоследовательными? Вот как это может выглядеть в целом.

Если вы внимательно посмотрите, есть ли пустые ячейки, это преднамеренно. Вскоре вы увидите, как с этим играть.

После того, как вы закончите выбор данных, следующим шагом будет импорт необходимых библиотек и набора данных.

1. **Генерация датасеть**

Датасет (англ. dataset) - это структурированная коллекция данных, которая используется для анализа, исследования и обучения моделей машинного обучения. Датасет может содержать данные различных типов, таких как числа, тексты, изображения, аудио и другие форматы.

Данные в датасете обычно представлены в виде таблицы или матрицы, где каждая строка представляет собой один объект или пример данных, а каждый столбец - один признак или характеристика объекта. Например, датасет, содержащий информацию о пассажирах Титаника, может включать такие признаки, как возраст, пол, класс каюты и т.д.

Для использования датасета в задачах машинного обучения необходимо разделить его на две части: обучающую и тестовую выборки. Обучающая выборка используется для обучения модели, а тестовая выборка - для проверки ее качества и оценки ее способности обобщаться на новые данные.

Генерация датасетов - это процесс создания структурированных данных для использования в анализе и машинном обучении. Она может быть выполнена с помощью различных методов, включая генерацию случайных данных с использованием следующих библиотек, или сбор данных из различных источников, таких как базы данных и API.

* NumPy: библиотека для работы с массивами и матрицами, которая также может использоваться для генерации случайных данных.
* Pandas: библиотека для работы с данными, которая содержит функции для создания и манипулирования датасетами.
* Faker: библиотека для генерации случайных данных различных типов, таких как имена, адреса, номера телефонов и т.д.
* SciPy: библиотека для выполнения научных и инженерных расчетов, которая содержит множество функций для генерации случайных данных.
* Scikit-learn: библиотека для машинного обучения, которая содержит функции для генерации и обработки датасетов.
* TensorFlow: библиотека для машинного обучения, которая может использоваться для генерации и обработки датасетов.
* FakerSchema: библиотека, основанная на Faker, которая позволяет генерировать сложные датасеты с использованием схемы.
* mimesis: еще одна библиотека для генерации случайных данных различных типов.
* PyDBGen: библиотека для генерации баз данных различных типов, таких как PostgreSQL, MySQL и SQLite.
* PySynth: библиотека для генерации музыкальных датасетов.
* PyDataset: библиотека для генерации различных типов датасетов, включая текстовые, числовые и временные ряды.
* Quandl: библиотека для получения и анализа финансовых данных.
* GPT-2 Simple: библиотека для генерации текстовых датасетов с помощью нейросетей.
* PyDBLite: библиотека для генерации баз данных различных типов.

**Примеры**

1. Генерация случайной матрицы 5x5 с помощью библиотеки NumPy

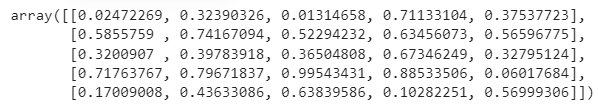
| import numpy as np  # Создаем массив  data = np.random.rand(5, 5)  # вывод датасета  print(data)  # Сохраняем массив в файл  np.savetxt('my\_numpy1\_dataset.txt', data) |
| --- |

Этот код создает двумерный массив размером 5x5, заполненный случайными числами от 0 до 1 с помощью функции np.random.rand().

Более подробно:

* np.random.rand(5, 5) создает двумерный массив размером 5x5, содержащий случайные числа от 0 до 1.
* print(data) – вывод датасет
* np.savetxt('my\_numpy\_dataset.txt', data) - сохраняет этот массив для дальнейшего использования в коде.

Результат.



1. Сгенерировать датасет из 100 строк и 5 столбцов со случайными числами и вывести первые 5 строк датасета.

| import numpy as np  import pandas as pd  # Создаем массив случайных чисел  data = np.random.rand(100, 5)  # Создаем датафрейм из массива  dataset = pd.DataFrame(data, columns=['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5'])  # Вывод первых 5 строк датафрейма  print(dataset.head())  # Сохраняем датафрейм в файл CSV  dataset.to\_csv('my\_numpy2\_dataset.txt', index=False) |
| --- |

Этот код создает массив data размером 100x5, заполненный случайными числами от 0 до 1 с помощью функции np.random.rand(). Затем он использует этот массив для создания датафрейма pandas dataset с помощью функции pd.DataFrame(). Каждый столбец датафрейма имеет название "Feature X", где X - номер столбца.

Затем код выводит первые 5 строк датафрейма с помощью метода head() и сохраняет датафрейм в файл "my\_dataset.csv" с помощью метода to\_csv(). Параметр index=False указывает на то, что индексы строк не должны быть сохранены в файл, а только значения признаков.

Обратите внимание, что сохранение происходит в формате CSV, поэтому файл будет сохранен в текстовом формате, разделенном запятыми. Если требуется сохранить данные в другом формате, например, в формате Excel, необходимо использовать соответствующие методы pandas.

Результат:

| Feature 1 Feature 2 Feature 3 Feature 4 Feature 5  0 0.388459 0.705375 0.940729 0.358782 0.927425  1 0.873583 0.907487 0.235753 0.907984 0.895728  2 0.589956 0.528866 0.007542 0.633298 0.462289  3 0.724588 0.214697 0.617951 0.527346 0.179034  4 0.871924 0.346176 0.013914 0.499481 0.253718 |
| --- |

1. Сгенерировать датасет для задачи прогнозирования продаж на основе исторических данных с помощью pandas.

Условия:

* Использовать библиотеку pandas для генерации датасета
* Датасет должен содержать два столбца: дата и продажи
* Даты должны быть сгенерированы за период с 1 января 2022 года по 31 декабря 2022 года с помощью функции pd.date\_range()
* Продажи должны быть сгенерированы случайными числами из нормального распределения с помощью функции np.random.normal()
* Датасет должен содержать 366 строк (включая високосный день)
* Первые 365 строк должны содержать исторические данные, а последняя строка - целевое значение, которое нужно предсказать
* Целевое значение должно быть продажами на 1 января 2023 года
* Вывести первые 5 строк датасета с помощью метода head().

| import pandas as pd  import numpy as np  # создание временного ряда с датами  dates = pd.date\_range('2022-01-01', '2022-12-31')  # создание временного ряда с продажами  sales = np.random.normal(1000, 100, len(dates))  # создание датасета из временных рядов  dataset = pd.DataFrame({'Sales': sales}, index=dates)  # создание столбца с целевой переменной  dataset['Target'] = dataset['Sales'].shift(-1)  # удаление последней строки, так как в ней отсутствует целевая переменная  dataset = dataset[:-1]  # вывод первых 5 строк датасета  print(dataset.head())  # Сохранение датафрейма в файл CSV  dataset.to\_csv('my\_pandas1\_dataset.csv', index=True) |
| --- |

В этом примере мы создаем временной ряд с датами с помощью функции pd.date\_range(). Затем мы создаем временный ряд с продажами с помощью функции np.random.normal(), которая генерирует случайные числа из нормального распределения. Далее мы объединяем два временных ряда в датасет с помощью функции pd.DataFrame(). Далее мы создаем столбец с целевой переменной, который представляет собой продажи на следующий день, с помощью метода shift(), который сдвигает значения столбца на одну строку вверх. Наконец, мы удаляем последнюю строку, так как в ней отсутствует целевая переменная, и выводим первые 5 строк датасета с помощью метода head().

Таким образом, мы получаем датасет размером 365x2 с историческими данными по продажам и целевым значением - продажами на 1 января 2023 года.

Результат:

| Sales Target  2022-01-01 1044.606563 942.694832  2022-01-02 942.694832 941.878711  2022-01-03 941.878711 1031.421276  2022-01-04 1031.421276 1005.715549  2022-01-05 1005.715549 1198.311365 |
| --- |

1. Сгенерируйте 30-дневный временной ряд с частотой один день:

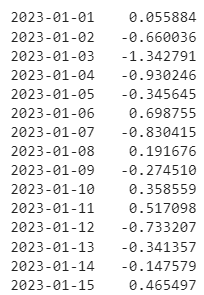
| import pandas as pd  import numpy as np  # Создание временного ряда  dates = pd.date\_range('2023-01-01', periods=30, freq='D')  data = pd.Series(np.random.randn(30), index=dates)  # Вывод объекта Series  print(data)  # Сохранение объекта Series в файл CSV  data.to\_csv('my\_data.csv', index=True) |
| --- |

Этот код создает объект data типа Series библиотеки pandas, который содержит 30 случайных чисел, соответствующих 30 дням, начиная с 1 января 2023 года.

Более подробно:

* pd.date\_range('2023-01-01', periods=30, freq='D') создает массив дат, начиная с 1 января 2023 года и продолжительностью в 30 дней с частотой 'D' (каждый день).
* np.random.randn(30) создает массив из 30 случайных чисел, распределенных по нормальному закону.
* pd.Series(np.random.randn(30), index=dates) создает объект Series, который содержит случайные числа и даты в качестве индексов.

Результат.



1. Создайте список из 100 случайных имен, используя библиотеку Faker.

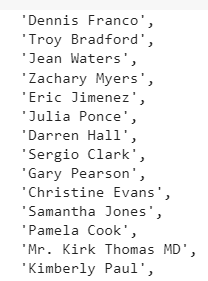
| from faker import Faker  fake = Faker()  # Создание списка имен  names = [fake.name() for i in range(100)]  # Вывод списка имен  print(names)  # Сохранение списка имен в файл  with open('my\_names.txt', 'w') as f:      for name in names:          f.write(name + '\n') |
| --- |

Этот код использует библиотеку faker, чтобы создать 100 случайных имен с помощью метода name(). Результат сохраняется в переменной names, которая является списком.

Более подробно:

* Faker() создает объект fake, который будет использоваться для генерации случайных данных.
* fake.name() генерирует случайное имя.
* [fake.name() for i in range(100)] создает список, содержащий 100 случайных имен, сгенерированных с помощью метода name().
* names сохраняет этот список для дальнейшего использования в коде.

Результат



1. Сгенерировать датасет для задачи прогнозирования продаж на основе исторических данных с помощью faker.

Условия:

* Нам нужен датасет, который содержит информацию о продажах за последние 2 года (с 2019 по 2021 год).
* Датасет должен содержать информацию о продуктах, проданных за этот период, включая наименование, категорию, цену и количество.
* Нам также нужно знать информацию о покупателях, включая их имена, адреса электронной почты и местоположение.
* Для каждой продажи нам нужно знать дату и время покупки.
* Мы будем использовать этот датасет для прогнозирования будущих продаж на основе исторических данных.

| !pip install faker  from faker import Faker  import random  import pandas as pd  fake = Faker()  # Создаем список продуктов  products = []  for i in range(100):      product = {          'name': fake.word(),          'category': fake.word(),          'price': random.randint(10, 100),      }      products.append(product)  # Создаем список покупателей  customers = []  for i in range(50):      customer = {          'name': fake.name(),          'email': fake.email(),          'location': fake.country(),      }      customers.append(customer)  # Создаем список продаж  sales = []  for i in range(1000):      sale = {          'product\_name': random.choice(products)['name'],          'product\_category': random.choice(products)['category'],          'product\_price': random.choice(products)['price'],          'customer\_name': random.choice(customers)['name'],          'customer\_email': random.choice(customers)['email'],          'customer\_location': random.choice(customers)['location'],          'date\_time': fake.date\_time\_between(start\_date='-2y', end\_date='now'),      }      sales.append(sale)  # Создаем датафрейм из списка продаж  df\_sales = pd.DataFrame(sales)  # вывод первых 5 строк датасета  df\_sales.head(5)  # Сохраняем датафрейм в CSV файл  df\_sales.to\_csv('sales\_dataset.csv', index=False) |
| --- |

Этот код создаст датасет, содержащий 1000 записей о продажах за последние 2 года, каждая запись содержит информацию о продукте, покупателе и дату/время покупки. Файл будет сохранен в формате CSV под названием "sales\_dataset.csv". Теперь объясняем каждую часть кода:

* from faker import Faker - мы импортируем библиотеку faker, которая позволяет создавать случайные данные.
* import random - мы импортируем модуль random, который позволяет генерировать случайные числа.
* import pandas as pd - мы импортируем библиотеку pandas, которая позволяет работать с таблицами данных.
* fake = Faker() - мы создаем объект faker, который будем использовать для генерации случайных данных.
* products = [] - мы создаем пустой список продуктов, который будем заполнять случайными товарами.
* for i in range(100): - мы создаем цикл, который будет повторяться 100 раз.
* product = {'name': fake.word(), 'category': fake.word(), 'price': random.randint(10, 100),} - мы создаем словарь продукта, который будет содержать случайное наименование, категорию и цену.
* products.append(product) - мы добавляем созданный продукт в список продуктов.
* Аналогично генерируются список покупателей и список продаж.
* df\_sales = pd.DataFrame(sales) - мы создаем датафрейм pandas из списка продаж.
* df\_sales.to\_csv('sales\_dataset.csv', index=False) - мы сохраняем датафрейм в файл CSV с именем "sales\_dataset.csv".
* print(df\_sales.head(5)) - мы выводим первые 5 строк датафрейма с помощью метода head().

Результат:



1. Необходимо сгенерировать временной ряд, используя функцию scipy.stats.norm() для создания нормального распределения с помощью библиотеки SciPy.

| import pandas as pd  from scipy.stats import norm  import numpy as np  # Задание параметров распределения  mu = 1000  sigma = 50  # Генерация случайных чисел из нормального распределения  data = norm.rvs(loc=mu, scale=sigma, size=100)  # Создание датафрейма из временного ряда  dates = pd.date\_range('2022-01-01', periods=100, freq='D')  dataset = pd.DataFrame({'Sales': data}, index=dates)  # Создание столбца с целевой переменной  dataset['Target'] = dataset['Sales'].shift(-1)  # Удаление последней строки, так как в ней отсутствует целевая переменная  dataset = dataset[:-1]  # Вывод первых 5 строк датафрейма  print(dataset.head())  # Сохранение датафрейма в файл CSV  dataset.to\_csv('my\_dataset.csv', index=True) |
| --- |

Данный код выполняет следующие действия:

* Импортирует необходимые библиотеки: pandas, scipy.stats и numpy.
* Задает параметры нормального распределения: среднее значение mu равно 1000, стандартное отклонение sigma равно 50.
* Генерирует случайную выборку размера 100 из нормального распределения с заданными параметрами с помощью функции norm.rvs().
* Создает датафрейм dataset из временного ряда, используя функцию pd.date\_range() для создания дат и метод pd.DataFrame() для объединения данных.
* Создает столбец Target, который содержит значения целевой переменной, сдвинутые на одну позицию вперед с помощью метода shift().
* Удаляет последнюю строку датафрейма, так как в ней отсутствует целевая переменная.
* Выводит первые 5 строк датафрейма с помощью метода head().
* Сохраняет датафрейм в файл CSV с помощью метода to\_csv().
* Таким образом, код генерирует случайный временной ряд и сохраняет его в файл для дальнейшего использования.

1. Необходимо сгенерировать временной ряд, используя функцию scipy.stats.expon() для экспоненциального распределения с помощью библиотеки SciPy.

| import pandas as pd  from scipy.stats import expon  import numpy as np  # Задание параметров распределения  scale = 1/100  # Генерация случайных чисел из экспоненциального распределения  data = expon.rvs(scale=scale, size=100)  # Создание датафрейма из временного ряда  dates = pd.date\_range('2023-08-22', periods=100, freq='D')  dataset = pd.DataFrame({'Values': data}, index=dates)  # Сохранение датафрейма в файл CSV  dataset.to\_csv('my\_dataset.csv', index=True)  # Вывод первых 5 строк датафрейма  print(dataset.head()) |
| --- |

Данный код использует библиотеку Pandas для генерации случайного временного ряда из экспоненциального распределения и сохранения его в файл CSV.

* Импортируются библиотеки Pandas, Scipy и NumPy. Объявляется переменная "scale", которая задает параметр масштаба для экспоненциального распределения. Затем генерируется случайный временной ряд из экспоненциального распределения с помощью функции "expon.rvs()".
* Создается датафрейм, используя функцию "pd.DataFrame()", который содержит сгенерированные случайные числа и даты, сгенерированные функцией "pd.date\_range()".
* Далее, датафрейм сохраняется в файл CSV с помощью метода "to\_csv()".
* Наконец, выводится первые 5 строк датафрейма с помощью метода "head()".

Результат:

| Values  2023-08-22 0.001865  2023-08-23 0.003029  2023-08-24 0.004645  2023-08-25 0.015694  2023-08-26 0.010819 |
| --- |

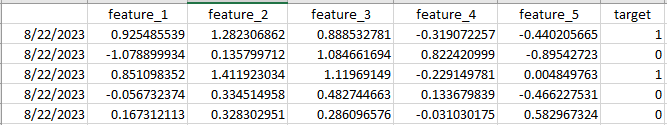
1. Сгенерируйте датасет с помощью Scikit-learn.

| from sklearn.datasets import make\_classification  import pandas as pd  from datetime import datetime  # Генерация датасета  X, y = make\_classification(n\_samples=1000, n\_features=5, n\_classes=2)  # Создание датафрейма  data = pd.DataFrame(X, columns=['feature\_{}'.format(i) for i in range(1, 6)])  data['target'] = y  # Добавление индекса с использованием текущей даты и времени  now = datetime.now()  data.index = pd.date\_range(start=now, periods=len(data), freq='S')  # Вывод первых 5 строк датафрейма  print(data.head())  # Сохранение датафрейма в файл  data.to\_csv('dataset.csv') |
| --- |

Данный код выполняет следующие действия:

* from sklearn.datasets import make\_classification - импортирует функцию make\_classification из библиотеки Scikit-learn, которая позволяет генерировать синтетические датасеты для задач классификации.
* import pandas as pd - импортирует библиотеку Pandas, которая позволяет работать с табличными данными.
* from datetime import datetime - импортирует функцию datetime из стандартной библиотеки Python, которая позволяет работать с датами и временем.
* X, y = make\_classification(n\_samples=1000, n\_features=5, n\_classes=2) - генерирует датасет, содержащий 1000 объектов с 5 признаками каждый и 2 классами.
* data = pd.DataFrame(X, columns=['feature\_{}'.format(i) for i in range(1, 6)]) - создает датафрейм Pandas на основе матрицы признаков X и задает имена столбцов.
* data['target'] = y - добавляет столбец с метками классов в датафрейм.
* now = datetime.now() - сохраняет текущую дату и время в переменной now.
* data.index = pd.date\_range(start=now, periods=len(data), freq='S') - добавляет индекс в датафрейм, используя текущую дату и время, создавая последовательность дат с интервалом в 1 секунду.
* print(data.head()) - выводит первые 5 строк датафрейма.
* data.to\_csv('dataset.csv') - сохраняет датафрейм в файл в формате CSV.

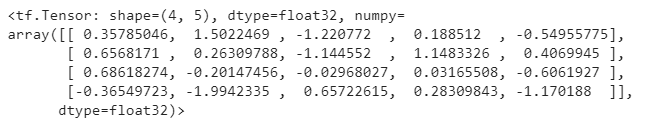
Результат:



1. Генерация данных машинного обучения — создание случайного набора данных 100x10 с использованием библиотеки TensorFlow.

| import tensorflow as tf  data = tf.random.normal(shape=(4, 5))  data |
| --- |

Результат.



1. Сгенерировать датасет с помощью tensorflow

| import pandas as pd  import tensorflow as tf  from sklearn.datasets import make\_classification  # Создание датасета с 2 признаками и 2 классами  X, y = make\_classification(n\_samples=100, n\_features=2, n\_informative=2, n\_redundant=0, n\_repeated=0, n\_classes=2)  dataset = tf.data.Dataset.from\_tensor\_slices((X, y))  # Преобразование датасета в Pandas DataFrame  data = pd.DataFrame(columns=['feature1', 'feature2', 'target'])  for x, y in dataset:      data = data.append({'feature1': x.numpy()[0], 'feature2': x.numpy()[1], 'target': y.numpy()}, ignore\_index=True)  #вывод датасет  print(data.head(5))  # Сохранение датасета в файл  data.to\_csv('dataset.csv', index=False) |
| --- |

Данный код создает датасет с двумя признаками и двумя классами, используя библиотеки pandas, tensorflow и sklearn.

* С помощью функции make\_classification() из библиотеки sklearn создается датасет с 100 наблюдениями, 2 признаками и 2 классами.
* Затем датасет преобразуется в формат tensorflow Dataset и далее конвертируется в pandas DataFrame.
* Выводится первые 5 строк датасета с помощью функции head().
* Наконец, данные сохраняются в файл dataset.csv с помощью функции to\_csv() из библиотеки pandas.

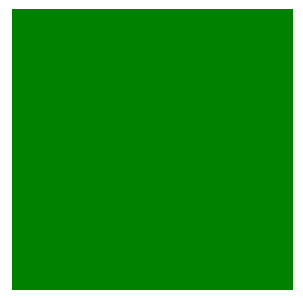
Результат:

| feature1 feature2 target  0 1.697149 -0.293692 1.0  1 1.197105 0.883810 1.0  2 1.252002 1.311769 1.0  3 -1.488783 1.650552 0.0  4 1.242233 0.199242 1.0 |
| --- |

1. Создайте изображение размером 256x256 пикселей, используя функцию Image.new библиотеки Pillow.

| from PIL import Image  image = Image.new('RGB', (256, 256), color='green')  image |
| --- |

Результат



1. Генерация текстовых данных (абзац текста) с помощью библиотеки lorem.

| import lorem  text = lorem.paragraph()  text |
| --- |

Результат.

| Voluptatem numquam non amet. Modi neque etincidunt eius quiquia quaerat. Neque neque amet sit magnam modi. Dolore quisquam velit sit. Modi porro neque tempora. Eius porro sit non modi voluptatem. |
| --- |

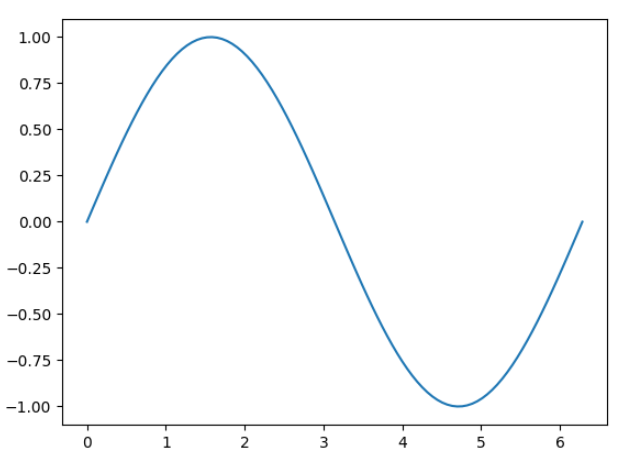
1. Генерация звуковых файлов — генерируем звуковой файл длиной 5 секунд с частотой дискретизации 44100 Гц с помощью библиотеки scipy

| from scipy.io.wavfile import write  import numpy as np  sample\_rate = 44100  duration = 5  frequency = 440  t = np.linspace(0, duration, duration \* sample\_rate, False)  audio\_data = np.sin(2 \* np.pi \* frequency \* t)  write('sound.wav', sample\_rate, audio\_data) |
| --- |

1. Сгенерируйте график функции y=sin(x) на интервале от 0 до 2π с помощью библиотеки matplotlib.

| import matplotlib.pyplot as plt  import numpy as np  x = np.linspace(0, 2\*np.pi, 100)  y = np.sin(x)  plt.plot(x, y)  plt.show() |
| --- |

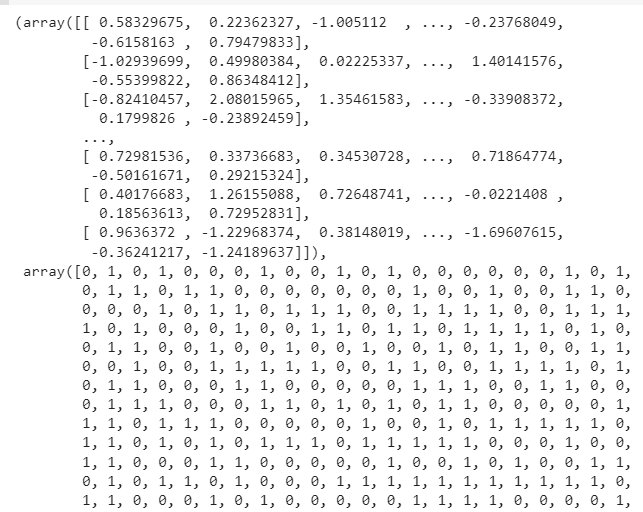
Результат.



1. Генерация данных для машинного обучения. Создайте набор данных из двух классов с помощью функции make\_classification с использованием библиотеки scikit-learn.

| from sklearn.datasets import make\_classification  X, y = make\_classification(n\_samples=1000, n\_features=10, n\_classes=2)  X, y |
| --- |

Результат.



1. Генерация данных для обработки естественного языка. Создайте список из 100 случайных слов на английском языке с помощью библиотеки NLTK.

| import nltk  import random  nltk.download('words')  words = nltk.corpus.words.words()  random\_words = [random.choice(words) for i in range(100)]  print(random\_words) |
| --- |

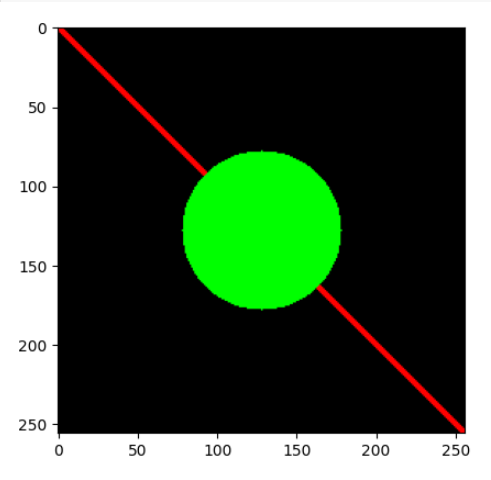
Результат.

| ['dismay', 'strubbly', 'iconomania', 'olfactor', 'stepgrandmother', 'embolismus', 'cysticercoid', 'leadless', 'Synedrium', 'indogen', 'adultness', 'resubjugate', 'gumlike', 'antipudic', 'shamalo', 'spireward', 'eulogistical', 'hackthorn', 'billposter', 'rhyparographist', 'overdeliciously', 'indigotindisulphonic', 'laevogyrate', 'punctographic', 'carpogone', 'nontransposition', 'disaccustomed', 'patriarchal', 'pyopericardium', 'ethenol', 'manneristic', 'isidiiferous', 'microsporangium', 'Antwerp', 'monopteroid', 'paltriness', 'restlessly', 'lai', 'roble', 'botchka', 'kudu', 'ricker', 'Geoteuthis', 'swaying', 'lovemate', 'technicalness', 'rosed', 'equicohesive', 'cotylar', 'earworm', 'uncanonize', 'monosyllabic', 'stirrupwise', 'Acanthopterygii', 'timesaver', 'rattle', 'maria', 'coruscate', 'noncritical', 'vallecular', 'unfeignedness', 'grilse', 'coalrake', 'scrooge', 'uxoriousness', 'technologue', 'unexhalable', 'movelessness', 'Gekkonidae', 'unduped', 'coir', 'keratinize', 'runnable', 'periotic', 'phylacteric', 'revelation', 'Pittism', 'drepanium', 'perissodactyle', 'parlay', 'iridesce', 'vinculate', 'concerningness', 'scour', 'knuckly', 'geodetician', 'undersetting', 'enraged', 'electrification', 'amidase', 'dermolysis', 'metapore', 'sanguinification', 'defense', 'radiosurgical', 'ophiomorph', 'unripening', 'devicefulness', 'saiga', 'cameleer'] |
| --- |

1. Генерация изображения. Создание изображения размером 256x256 пикселей со случайными линиями и кругами с использованием библиотеки OpenCV.

| import cv2  import numpy as np  import matplotlib.pyplot as plt  img = np.zeros((256, 256, 3), dtype=np.uint8)  cv2.line(img, (0, 0), (255, 255), (255, 0, 0), 2)  cv2.circle(img, (128, 128), 50, (0, 255, 0), -1)  plt.imshow(img)  plt.show() |
| --- |

Результат.



1. **Загрузка данных**

Первый шаг предобработки данных – это загрузка данных из различных источников, таких как базы данных, текстовые файлы, таблицы Excel и т.д. В Python для загрузки данных можно использовать библиотеки Pandas, NumPy, PyTorch, TensorFlow, Scikit-Learn, Keras, NLTK, OpenCV, Gensim, seaborn, yfinance и т.п.

Давайте приведем несколько примеров.

1. Сначала мы скачиваем датасет в Google Colab. Затем мы будем использовать библиотеку Pandas для чтения набора данных, который мы загрузили с Kaggle в формате CSV.

***Примечание***. *Ссылка на Google Colab:* <https://colab.research.google.com/drive/1qk7URor8jrWzrDCPECb0vTEOKHKoisnF?usp=drive_link>

Загрузка данных

| import pandas as pd  url = "top2018.csv"  dataset = pd.read\_csv(url) |
| --- |

Вывод на экран:

| Dataset |
| --- |

Результат:



Довольно просто, не так ли? Вы можете делать гораздо больше, например, знать форму, читать первые n строк, читать последние n строк и т. д.

Давайте попробуем это!

| dataset.shape |
| --- |

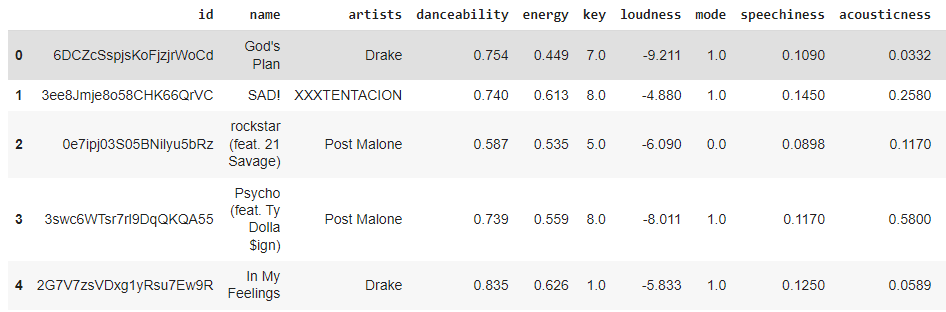
Результат

(100, 16)

Иногда лучше показывать несколько строк данных, чем заполнять экран целыми строкам.

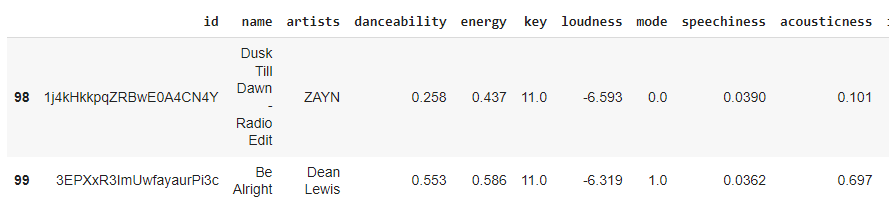
| dataset.head(5) |
| --- |

Результат:



Также, если вы хотите узнать последние несколько строк, используйте хвост

| dataset.tail(2) |
| --- |

Результат:

Или, если вы хотите знать только столбцы

| dataset.columns |
| --- |

Результат:

| Index(['id', 'name', 'artists', 'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration\_ms', 'time\_signature'], dtype='object') |
| --- |

1. **Numpy**:

| import numpy as np  data = np.load('data.npy')  X = data[:, :-1]  y = data[:, -1] |
| --- |

Данный код загружает данные из файла data.npy, который содержит массив numpy с признаками и метками целевой переменной.

* X = data[:, :-1] выбирает все столбцы, кроме последнего, что означает, что в X будут храниться только признаки.
* y = data[:, -1] выбирает только последний столбец, который содержит метки целевой переменной. Это означает, что в y будут храниться только метки.

Аналогично образом можно использовать Pandas.

1. **Pandas:**

| import pandas as pd  data = pd.read\_csv("data.csv")  X = data.iloc[:, :-1]  y = data.iloc[:, -1] |
| --- |

Обычно данные разделяют на признаки и метки, чтобы использовать их для обучения модели. В данном случае, X будет содержать признаки, которые будут использоваться для предсказания y.

Например, если у нас есть данные о размере дома, количестве комнат и цене на дома, то мы можем использовать размер дома и количество комнат в качестве признаков для предсказания цены на дом. Таким образом, размер дома и количество комнат будут храниться в X, а цена на дом - в y.

**Примечание**. Если у вас отсутствуют данные, то вы можете создать их, используя следующий код.

| import numpy as np  import pandas as pd  # Создаем случайные данные  size = np.random.randint(10, 300, size=(1000, 1))  rooms = np.random.randint(1, 6, size=(1000, 1))  price = np.random.randint(50000, 500000, size=(1000, 1))  # Объединяем данные в один массив  data = np.hstack((size, rooms, price))  # Создадим таблица данных  df = pd.DataFrame(  data=data,  columns=['size', 'rooms', 'price'])  # Сохраняем датасетm в файл  df.to\_csv("data.csv", encoding='utf-8', index=False) |
| --- |

Этот код генерирует случайные данные для создания датасета и сохраняет его в файл "data.csv". В датасете содержится информация о размере жилья, количестве комнат и цене. Количество строк в датасете равно 1000.

1. **PyTorch:**

| import torch  from torch.utils.data import Dataset, DataLoader  class MyDataset(Dataset):  def \_\_init\_\_(self, file\_path):  self.data = []  with open(file\_path, 'r') as f:  for line in f:  self.data.append(line.strip())  def \_\_getitem\_\_(self, index):  return self.data[index]  def \_\_len\_\_(self):  return len(self.data)  dataset = MyDataset('data.csv')  dataloader = DataLoader(dataset, batch\_size=2, shuffle=True)  dataset.data |
| --- |

В этом примере мы создаем класс MyDataset, который наследуется от torch.utils.data.Dataset. В конструкторе класса мы открываем файл с данными и считываем его содержимое в список self.data. Затем мы реализуем методы \_\_getitem\_\_ и \_\_len\_\_, которые позволяют получать элементы из датасета и узнавать его длину соответственно.

После того, как мы создали датасет, мы можем использовать torch.utils.data.DataLoader, чтобы создать загрузчик данных. В этом примере мы создаем загрузчик с размером пакета 2 и перемешиваем данные при каждой эпохе обучения.

1. **TensorFlow:**

| import tensorflow as tf  import pandas as pd  data = pd.read\_csv("data.csv")  dataset = tf.data.Dataset.from\_tensor\_slices(data)  dataset = dataset.batch(2).shuffle(buffer\_size=5)  for z in dataset:  print(z.numpy()) |
| --- |

Этот код загружает данные из файла "data.csv" в объект pandas DataFrame "data". Затем создается TensorFlow датасет "dataset" с помощью метода "from\_tensor\_slices", который принимает DataFrame в качестве аргумента. Датасет разбивается на батчи размером 2 и перемешивается с использованием буфера размера 5. В цикле происходит итерация по датасету и выводятся значения батчей в виде numpy массивов.

1. **Scikit-Learn:**

| from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  iris = load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2) |
| --- |

Этот код загружает датасет Iris из библиотеки scikit-learn с помощью метода "load\_iris". Затем данные делятся на обучающую и тестовую выборки с помощью метода "train\_test\_split". Размер тестовой выборки составляет 20% от всего набора данных. Переменные X\_train и y\_train содержат обучающие данные, а переменные X\_test и y\_test содержат тестовые данные.

**Загрузка датасета 'boston' (Boston Housing)**

| from sklearn.datasets import fetch\_openml  # Загрузка датасета  boston\_dataset = fetch\_openml(name='boston')  # Вывод описания датасета  print(boston\_dataset.DESCR)  # Получение массива признаков  X = boston\_dataset.data  # Получение массива целевой переменной  y = boston\_dataset.target |
| --- |

Этот код загрузит датасет Boston Housing и выведет его описание. Затем он получит массив признаков и массив целевой переменной.

В массиве признаков X будет содержаться 13 признаков для каждого наблюдения, такие как уровень преступности, среднее количество комнат в доме, наличие радиальных магистралей и т.д.

В массиве целевой переменной y будет содержаться медианная стоимость домов в районе в тысячах долларов.

1. **Keras:**

| from keras.datasets import mnist  from keras.utils import to\_categorical  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  x\_train = x\_train.reshape((60000, 28 \* 28))  x\_train = x\_train.astype('float32') / 255  y\_train = to\_categorical(y\_train) |
| --- |

Этот код загружает датасет MNIST из библиотеки Keras с помощью метода "mnist.load\_data()". Затем обучающие данные x\_train и y\_train извлекаются из загруженного датасета. Данные x\_train, которые представляют изображения цифр, преобразуются в одномерный массив размерности 784 (28 \* 28) и нормализуются путем деления на 255. Затем метки классов y\_train преобразуются в бинарные векторы с помощью метода "to\_categorical". Каждая метка класса заменяется на вектор размерности 10, где все элементы равны 0, кроме элемента с индексом, соответствующим классу, который должен быть равен 1.

1. **NLTK:**

| import nltk  nltk.download('punkt')  nltk.download('stopwords')  from nltk.corpus import stopwords  from nltk.tokenize import word\_tokenize  text = "This is a sample sentence, showing off the stop words filtration."  stop\_words = set(stopwords.words('english'))  words = word\_tokenize(text)  filtered\_sentence = [w for w in words if not w.lower() in stop\_words] |
| --- |

Этот код использует библиотеку NLTK для удаления стоп-слов из предложения "This is a sample sentence, showing off the stop words filtration.". Сначала загружаются пакеты "punkt" и "stopwords" с помощью метода "nltk.download()". Затем из корпуса NLTK "stopwords" загружается список стоп-слов для английского языка. Строка текста разбивается на слова с помощью метода "word\_tokenize". Затем создается новый список "filtered\_sentence", содержащий только те слова из исходного списка, которые не являются стоп-словами.

1. **OpenCV:**

| import cv2  img = cv2.imread('image.png')  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) |
| --- |

Этот код использует библиотеку OpenCV для чтения изображения из файла "image.jpg". Затем оно преобразуется в оттенки серого с помощью метода "cv2.cvtColor", который принимает два аргумента: исходное изображение и флаг преобразования цветового пространства. В данном случае флаг "cv2.COLOR\_BGR2GRAY" указывает на преобразование изображения из цветового пространства BGR в оттенки серого. Результат сохраняется в переменной "gray".

1. **Gensim**:

| import gensim.downloader as api  model = api.load("glove-wiki-gigaword-100")  result = model.most\_similar(positive=['woman', 'king'], negative=['man'], topn=1) |
| --- |

Этот код использует библиотеку Gensim для загрузки модели "glove-wiki-gigaword-100", предобученной на большом корпусе текстов. Затем вызывается метод "most\_similar" модели, который принимает три аргумента: список позитивных слов "positive", список негативных слов "negative" и количество наиболее похожих слов, которые необходимо вернуть в результате "topn". В данном случае мы ищем слово, наиболее близкое по значению к словосочетанию "woman" + "king" - "man". Результат сохраняется в переменной "result".

1. **Seaborn**:

| import seaborn as sns  tips = sns.load\_dataset("tips")  sns.relplot(x="total\_bill", y="tip", data=tips) |
| --- |

Этот код использует библиотеку Seaborn для загрузки датасета "tips", который содержит информацию о счетах и чаевых в ресторане. Затем вызывается метод "sns.relplot", который принимает три аргумента: значения по оси X, значения по оси Y и набор данных "data". В данном случае мы строим график рассеяния, отображающий зависимость между общей суммой счета и размером чаевых. Результат отображается на графике.

**Загрузка датасета titanic**

| import seaborn as sns  # Загрузка датасета titanic  titanic = sns.load\_dataset('titanic')  # Вывод первых 5 строк датасета  print(titanic.head()) |
| --- |

Этот код загружает библиотеку seaborn и использует ее функцию load\_dataset() для загрузки датасета "titanic", который содержит информацию о пассажирах корабля "Титаник", включая их возраст, пол, класс каюты, стоимость билета, а также информацию о том, выжил ли пассажир или нет.

После загрузки датасета, функция head() используется для вывода первых 5 строк датасета в консоль.

Это позволяет быстро ознакомиться с данными и убедиться, что датасет был загружен корректно.

1. yfinance

| import yfinance as yf  # Загрузка исторических данных о ценах на акцию Tesla  tesla = yf.download('TSLA', start='2010-01-01', end='2023-08-22')  # Вывод первых 5 строк датасета  print(tesla.head()) |
| --- |

Этот код использует библиотеку yfinance для загрузки исторических данных о ценах на акцию Tesla.

Функция download() загружает данные с Yahoo Finance для указанного тикера (в данном случае 'TSLA' - тикер акций Tesla) и для указанного диапазона дат (с 2010-01-01 по 2023-08-22).

Загруженные данные сохраняются в переменной tesla.

После загрузки данных, функция head() используется для вывода первых 5 строк датасета в консоль.

Это позволяет быстро ознакомиться с данными и убедиться, что данные были загружены корректно.

**Загрузка известных датасетов**

1. Загрузка датасета MNIST через библиотеку Keras:

| from keras.datasets import mnist  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data() |
| --- |

1. Загрузка датасета CIFAR-10 через библиотеку Keras:

| from keras.datasets import cifar10  (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data() |
| --- |

1. Загрузка датасета CIFAR-100 через библиотеку Keras:

| from keras.datasets import cifar100  (x\_train, y\_train), (x\_test, y\_test) = cifar100.load\_data() |
| --- |

1. Загрузка датасета Fashion-MNIST через библиотеку Keras:

| from keras.datasets import fashion\_mnist  (x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data() |
| --- |

1. Загрузка датасета Reuters новостей через библиотеку Keras:

| from keras.datasets import reuters  (x\_train, y\_train), (x\_test, y\_test) = reuters.load\_data() |
| --- |

1. Загрузка датасета California Housing через библиотеку scikit-learn:

| from sklearn.datasets import fetch\_california\_housing  # Load California Housing dataset  california\_dataset = fetch\_california\_housing() |
| --- |

1. Загрузка датасета Iris через библиотеку scikit-learn:

| from sklearn.datasets import load\_iris  # Load Iris dataset  iris\_dataset = load\_iris() |
| --- |

1. Загрузка датасета Wine через библиотеку scikit-learn:

| from sklearn.datasets import load\_wine  # Load Wine dataset  wine\_dataset = load\_wine() |
| --- |

1. Загрузка датасета Breast Cancer Wisconsin через библиотеку scikit-learn:

| from sklearn.datasets import load\_breast\_cancer  # Load Breast Cancer Wisconsin dataset  breast\_cancer\_dataset = load\_breast\_cancer() |
| --- |

1. Загрузка датасета Diabetes через библиотеку scikit-learn:

| from sklearn.datasets import load\_diabetes  # Load Diabetes dataset  diabetes\_dataset = load\_diabetes() |
| --- |

1. Загрузка датасета Olivetti Faces через библиотеку scikit-learn:

| from sklearn.datasets import fetch\_olivetti\_faces  # Load Olivetti Faces dataset  olivetti\_faces\_dataset = fetch\_olivetti\_faces() |
| --- |

1. Загрузка датасета Labeled Faces in the Wild через библиотеку scikit-learn:

| from sklearn.datasets import fetch\_lfw\_people  # Load Labeled Faces in the Wild dataset  lfw\_dataset = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4) |
| --- |

1. Загрузка датасета MNIST через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load MNIST dataset  dataset, info = tfds.load('mnist', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета CIFAR-10 через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load CIFAR-10 dataset  dataset, info = tfds.load('cifar10', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета CIFAR-100 через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load CIFAR-100 dataset  dataset, info = tfds.load('cifar100', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета Flowers через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load Flowers dataset  dataset, info = tfds.load('tf\_flowers', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета Rock Paper Scissors через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load Rock Paper Scissors dataset  dataset, info = tfds.load('rock\_paper\_scissors', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета Open Images через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Download and extract Open Images dataset  dataset, info = tfds.load('open\_images\_v4', with\_info=True, data\_dir='/content/open\_images', download=True) |
| --- |

1. Загрузка датасета Dogs vs Cats через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load Dogs vs Cats dataset  dataset, info = tfds.load('cats\_vs\_dogs', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета EuroSAT через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load EuroSAT dataset  dataset, info = tfds.load('eurosat/rgb', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета EuroSAT через библиотеку TensorFlow:

| import tensorflow\_datasets as tfds  # Load SVHN dataset  dataset, info = tfds.load('svhn\_cropped', with\_info=True, data\_dir='/content/data') |
| --- |

1. Загрузка датасета COCO через библиотеку pycocotools:

| !pip install pycocotools  import os  import urllib.request  # Download and extract COCO dataset  os.makedirs('/content/coco')  urllib.request.urlretrieve('http://images.cocodataset.org/annotations/annotations\_trainval2017.zip', '/content/coco/annotations.zip')  urllib.request.urlretrieve('http://images.cocodataset.org/zips/train2017.zip', '/content/coco/train.zip')  urllib.request.urlretrieve('http://images.cocodataset.org/zips/val2017.zip', '/content/coco/val.zip')  !unzip /content/coco/annotations.zip -d /content/coco  !unzip /content/coco/train.zip -d /content/coco  !unzip /content/coco/val.zip -d /content/coco |
| --- |

1. Загрузка репозитория целиком:

| !git clone https://github.com/Mcompetitions/M4-methods.git |
| --- |

1. Загрузка только одного файла:

| !wget  https://github.com/Mcompetitions/M4-methods/blob/master/Dataset/M4-info.csv |
| --- |

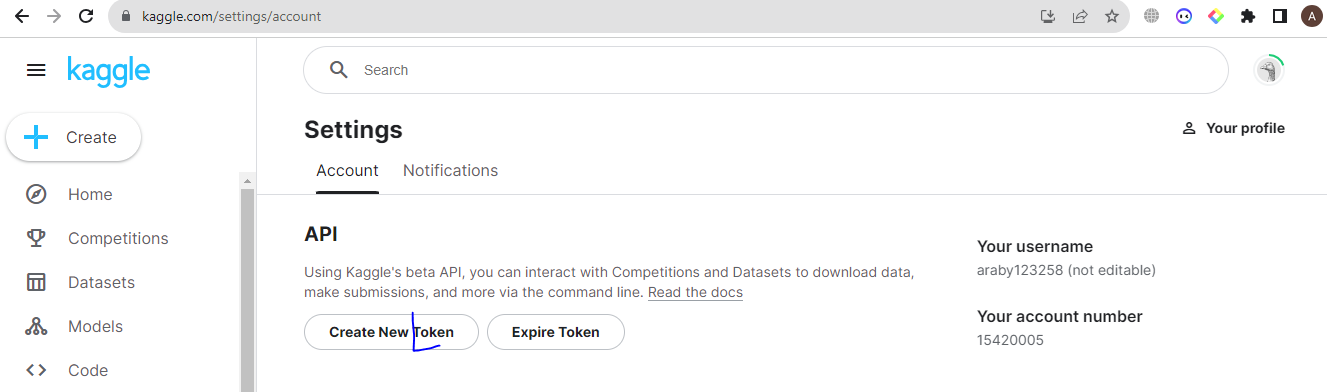
1. Загрузка файла из определенной ветки:

| !wget https://github.com/Mcompetitions/M4-methods/blob/CNN-TS/CNN-TS/predict.py |
| --- |

**3) Загрузка датасет из Kaggle**

Для загрузки датасетов с Kaggle в Google Colab необходимо использовать API-ключ Kaggle. Вы можете получить API-ключ, следуя этим шагам:

1. Зарегистрируйтесь на Kaggle, если вы еще этого не сделали.
2. Войдите в свой аккаунт Kaggle и перейдите на страницу "Account".
3. Прокрутите страницу вниз до раздела "API" и нажмите кнопку "Create New API Token".



Это загрузит файл kaggle.json на ваш компьютер.

1. Загрузите файл kaggle.json в Google Colab, используя команду files.upload().
2. После загрузки файла kaggle.json вам нужно выполнить седеющего кода

| # Установите библиотеку kaggle  !pip install -q kaggle  # Загрузите файл kaggle.json, содержащий ваш API-ключ Kaggle  from google.colab import files  files.upload()  # Создайте каталог для хранения файла конфигурации Kaggle  !mkdir ~/.kaggle  # Переместите файл kaggle.json в каталог .kaggle  !cp kaggle.json ~/.kaggle/  # Ограничьте права доступа к файлу конфигурации Kaggle  !chmod 600 ~/.kaggle/kaggle.json  # Загрузите датасет с помощью API Kaggle  !kaggle competitions download -c ashrae-energy-prediction  # Распакуйте архив с данными  !unzip ashrae-energy-prediction.zip |
| --- |

* Код начинается с установки библиотеки Kaggle с помощью команды !pip install -q kaggle.
* Далее, для загрузки файлов из Kaggle, необходимо иметь API-ключ. Для этого используется команда files.upload(), которая позволяет загрузить файл kaggle.json с локального компьютера пользователя на сервер.
* Затем создается каталог ~/.kaggle, который будет использоваться для хранения файла конфигурации Kaggle. Команда !cp kaggle.json ~/.kaggle/ перемещает загруженный файл kaggle.json в каталог .kaggle.
* Чтобы обеспечить безопасность, используется команда !chmod 600 ~/.kaggle/kaggle.json, которая ограничивает права доступа к файлу конфигурации Kaggle.
* Далее, используя API Kaggle, загружается датасет ashrae-energy-prediction с помощью команды !kaggle competitions download -c ashrae-energy-prediction.

После этого вы сможете загрузить датасет с помощью API Kaggle.

1. **Разделите набор данных на зависимую и независимую переменную**

После импорта набора данных следующим шагом будет определение независимой переменной (x) и зависимой переменной (y). Давайте вернемся к школьной математике и попробуем вспомнить, что это такое?

Независимая переменная (x) — это переменная, которую вы намеренно изменяете, чтобы увидеть эффект от зависимой переменной (y).

Например, в уравнении , x — независимая переменная, а y — зависимая переменная, потому что вы изменяете/настраиваете x, чтобы увидеть его изменение на y. Не так ли?

*В реальном мире представьте ученого, который хочет изучить влияние лекарства на рак. Независимыми переменными являются время приема препарата, дозировка и т. д. В то время как зависимой переменной является воздействие. Если он/она хочет увидеть изменение воздействия препарата, он/она может настроить/изменить время и дозировку.*

*В нашем случае, если мы хотим предсказать танцевальность на основе энергии, громкости, акустики и инструментальности, нашей зависимой переменной (y) будет танцевальность, а независимой переменной будет энергия, громкость, акустика и инструментальность.*

Итак, давайте сделаем это! Для этого нам понадобится iloc от pandas. iloc - это индексация на основе целочисленного местоположения для выбора по позиции. Опять запутались? Давайте посмотрим на синтаксис.

| iloc[<row-selection>, <column-selection>] |
| --- |

Пример

| X=dataset.iloc[:, [3,5,8,9]].values  y=dataset.iloc[:,2].values |
| --- |

Это код использует библиотеку pandas для работы с набором данных.

* dataset - это объект DataFrame, который содержит данные.
* iloc - это атрибут DataFrame, который используется для доступа к данным по индексу.
* [:, [3, 5, 8, 9]] - это синтаксис для извлечения всех строк в столбцах 3, 5, 8 и 9.
* .values - это атрибут DataFrame, который используется для извлечения значений из DataFrame в виде массива NumPy.

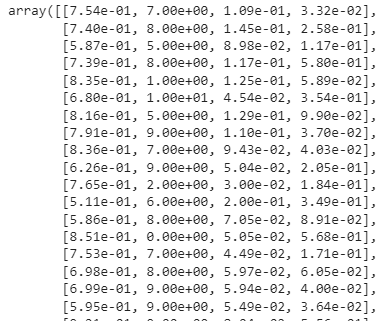
Таким образом, код X=dataset.iloc[:, [3,5,8,9]].values извлекает данные из столбцов 3, 5, 8 и 9 набора данных и сохраняет их в переменную X в виде массива NumPy.

Аналогично, код y=dataset.iloc[:,2].values извлекает данные из столбца 2 набора данных и сохраняет их в переменную y в виде массива NumPy.

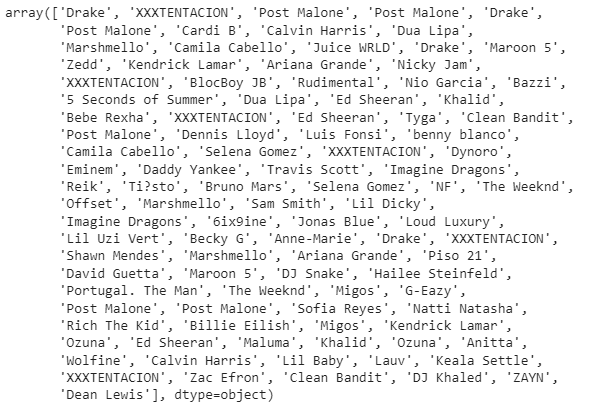
Вот как мы смогли выбрать зависимую переменную (y) и независимую переменную (x).

Вы, должно быть, уже чувствуете себя волшебником математики.

Результат X:



Результат y

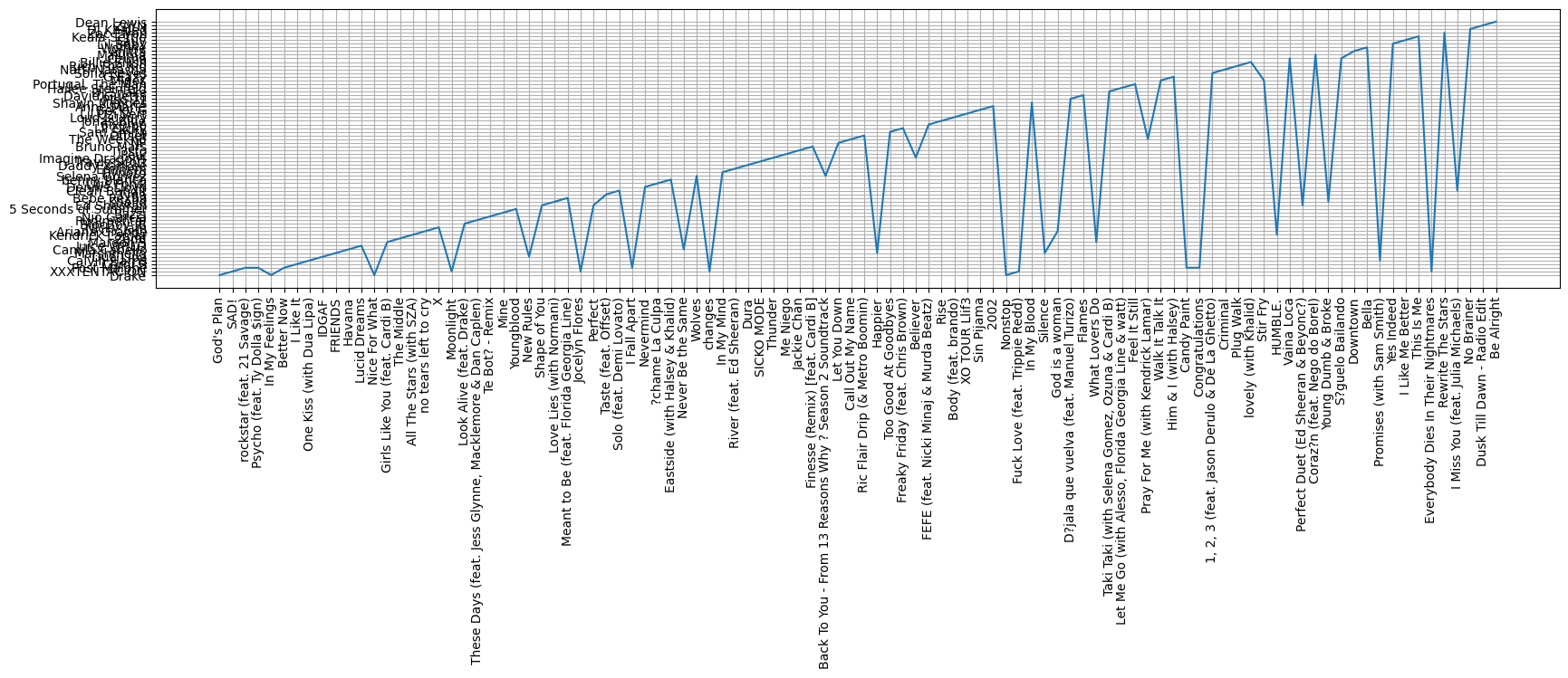


Подождите, прежде чем мы продолжим, вы помните библиотеку matplotlib, о которой мы говорили ранее? Почему бы нам не построить график на основе набора данных, чтобы увидеть, чья песня более танцевальная? Давайте сделаем это!

А вот код для этого. Не стесняйтесь настраивать!

| import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  url = "top2018.csv"  dataset = pd.read\_csv(url)  x = list(range(len(dataset)))  y = np.array(dataset.iloc[:, 2])  my\_xticks = dataset.iloc[:, 1]  plt.figure(figsize=(20, 4))  plt.xticks(x, my\_xticks, rotation='vertical')  plt.grid(True)  plt.plot(x, y)  plt.show() |
| --- |

Результат:



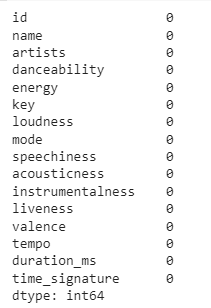
1. **Проверить пропущенные значения**

Понимание способов управления отсутствующими значениями очень важно. Как мы обсуждали ранее, в реальных данных часто могут быть пропущенные значения. Если пропущенные значения не обрабатываются должным образом, исследователь может в конечном итоге получить неправильную ссылку на данные. А обработка отсутствующих значений важна, потому что многие модели машинного обучения не поддерживают данные с отсутствующими значениями.

Мы всегда можем подсчитать количество столбцов с пропущенными значениями.

| dataset.isnull().sum() |
| --- |

Результат:



Вы можете увидеть количество столбцов с отсутствующими значениями. Теперь есть два подхода к этому:

* Удалить строку полностью с отсутствующими значениями
* Заменить отсутствующие значения

***Примечание.*** *Замена отсутствующих значений на замененные значения называется импьютацией.*

**Полностью удалить строку с отсутствующими значениями**

Этот подход, вероятно, самый простой, но имеет свои недостатки. Это может быть довольно опасно, что, если эта строка содержит важную информацию?

Дело не в том, что мы никогда не удаляем строку полностью, когда значение отсутствует, мы удаляем ее. Как и в нашем наборе данных, что, если имя исполнителя отсутствует или какой-либо ярлык класса, вы можете удалить в таком случае. Кроме того, если столбцы с отсутствующими значениями не являются ни независимой, ни зависимой переменной, их лучше игнорировать.

Панды предоставляют функцию dropna() , которую можно использовать для удаления строки или столбца с отсутствующими данными. Мы можем использовать dropna() для удаления всех строк с отсутствующими данными.

| dataset.shape # (49, 14)  dataset.dropna(inplace=True) dataset.shape # (38, 14) |
| --- |

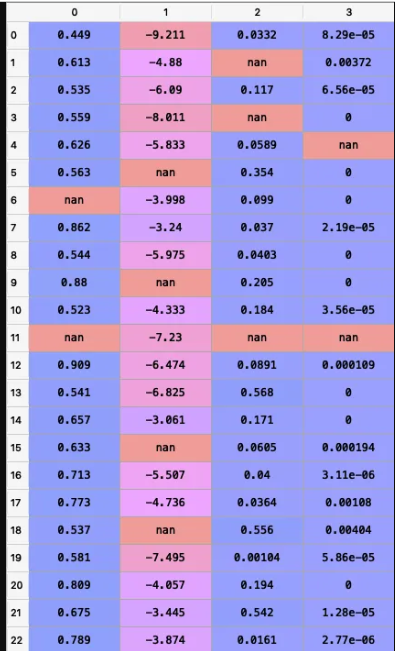
Вы можете видеть, что он удалил все строки с отсутствующими значениями. Но это не всегда хорошая идея.

Замена отсутствующих значений

Этот подход может быть применен к функции, которая имеет числовые данные, такие как зарплата, возраст и т.д., но не ограничивается ими. Мы можем вычислить среднее значение, моду или медиану функции и заменить ими отсутствующие значения. Это приближение, которое может добавить дисперсию в набор данных.

Теперь я буду использовать SimpleImputer для замены отсутствующих данных на среднее/медиану/наиболее часто встречающееся значение.

Давайте проверим нашу независимую переменную (x).



| import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  from sklearn.impute import SimpleImputer  #reading dataset  dataset = pd.read\_csv('top2018.csv')  #dividing into dependent and independent variables  X = dataset.iloc[:, [3,5,8,9]].values  y = dataset.iloc[:, 2].values  imputer = SimpleImputer(missing\_values=np.nan, strategy="median")  X[:, 0:4] = imputer.fit\_transform(X[:, 0:4]) |
| --- |

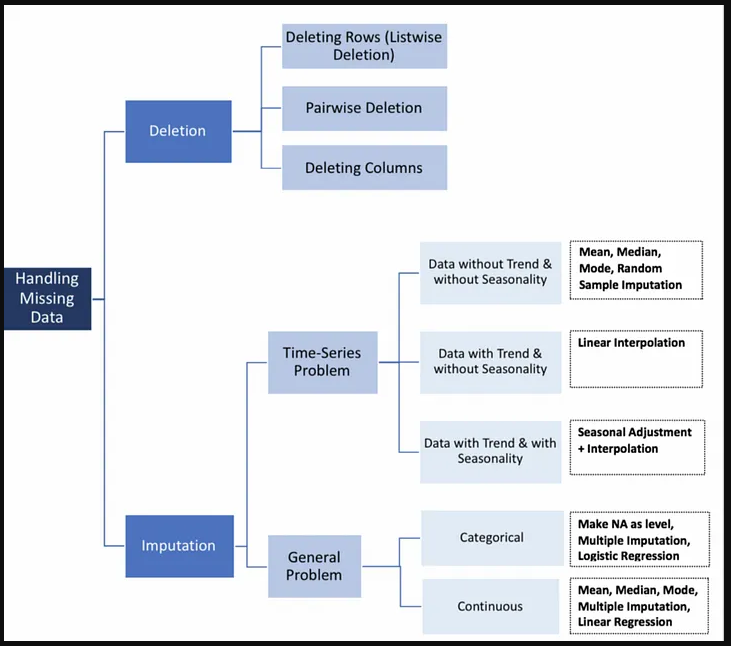
Мы использовали медианную стратегию для замены пропущенных значений



Вы можете видеть, что пропущенные значения заменены медианной стратегией. SimpleImputer также предоставляет другие стратегии:

1. Постоянная
2. Наиболее часто встречающаяся
3. Среднее
4. Медианное

Теперь следующий вопрос может заключаться в том, когда вы должны использовать эти разные подходы? Использование этих подходов полностью зависит от характера данных и причин их отсутствия.



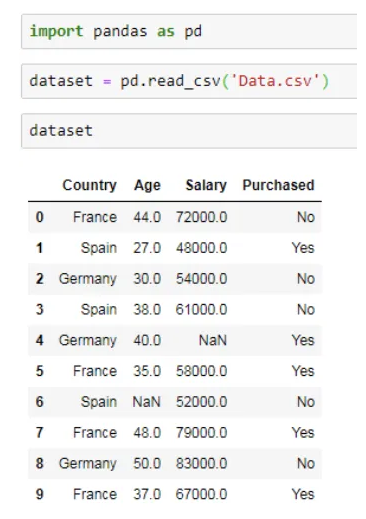
Эта шпаргалка может быть действительно полезной.

1. **Проверить категориальные значения**

Теперь давайте посмотрим, как работать с категориальными значениями.

Иногда наши данные могут быть в качественной форме, что означает, что в наших данных есть тексты. Теперь машинам становится сложнее понимать текст и обрабатывать его, а не числовые значения, потому что модели машинного обучения основаны на математическом уравнении! Поэтому необходимо кодировать категориальные значения.

Наш набор данных довольно большой, и его может быть трудно понять. Давайте посмотрим на это.



Как видите, первый столбец содержит данные в текстовом виде. Мы можем заметить, что есть 3 категории: Франция, Испания и Германия. Чтобы преобразовать категориальные значения в числовые данные, мы можем использовать класс **LabelEncoder()** из sklearn.preprocessing.

Давайте импортируем библиотеку

| from sklearn.preprocessing import LabelEncoder |
| --- |

Следующим шагом будет создание объекта этого класса, назовем его labelEncoderX.

| labelEncoderX = LabelEncoder() |
| --- |

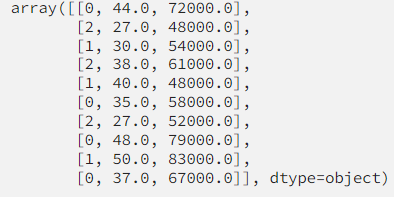
Теперь, чтобы преобразовать это в числовое, мы можем использовать

| X[:,0] = labelEncoderX.fit\_transform(X[:,0]) |
| --- |

Это выберет все строки и 0-й столбец, который является списком стран, и подгонит к нему кодировщик меток и преобразует значения. Это будет немедленно закодировано в 0,1,2,3… и т. д.

| print(X) |
| --- |

Результат:



Как видите, категориальные значения были закодированы. Но есть проблема!

Проблема все та же. Модели машинного обучения основаны на уравнениях, и хорошо, что мы заменили текст цифрами. Однако, поскольку 1 больше 0, а 2 больше 1, уравнения в модели будут считать, что Испания имеет более высокое значение, чем Германия, и Франция, а Германия имеет более высокое значение, чем Франция. И это, конечно, не так. На самом деле это три категории, и между ними нет порядка отношений. Мы не можем сравнивать Францию, Испанию и Германию, говоря, что Испания больше Германии или Германия больше Франции.

Таким образом, нам нужно найти способ помешать уравнениям машинного обучения думать, что Испания больше, чем Германия, а Германия больше, чем Франция. Чтобы предотвратить это, мы используем фиктивные переменные .

**Что такое фиктивные переменные?**

Фиктивные переменные — это дискретные переменные, принимающие значения «0» или «1». Их часто называют переменными «включено или выключено» или переменными-индикаторами. Они просто принимают 0 или 1, чтобы указать на наличие категорического эффекта.

Вместо того, чтобы иметь один столбец и назначать числовое значение, которое может вызвать смещение, мы создаем разные столбцы в зависимости от типа присутствующих категорий.

Number of Columns = Types of Categories

В нашем случае у нас есть 3 типа, поэтому у нас будет 3 столбца. Для этого мы импортируем еще одну библиотеку под названием OneHotEncoder.

| from sklearn.preprocessing import LabelEncoder, OneHotEncoder |
| --- |

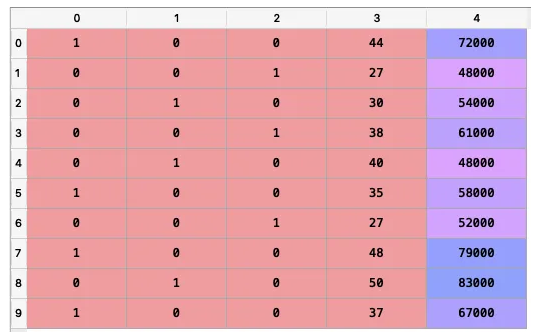
Следующим шагом является создание объекта этого класса с важным параметром, называемым categorical\_features, который принимает значение индекса столбца.

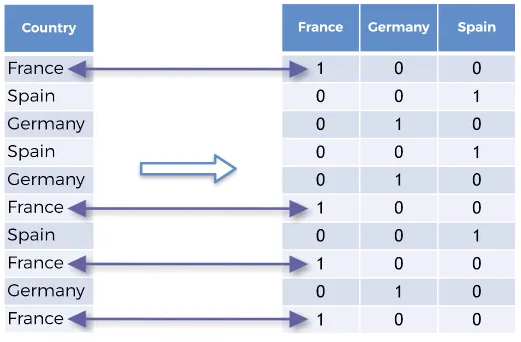
| onehotencoder = OneHotEncoder(categories=[0]) |
| --- |

Точно так же, как мы использовали fit\_transform() для LabelEncoder, мы будем использовать его и для OneHotEncoder.

| X = onehotencoder.fit\_transform(X).toarray() |
| --- |

Теперь, если вы видите нашу независимую переменную (X), вот как она выглядит.





1. **Разделите набор данных на обучающий набор и тестовый набор**

Когда мы делаем шаг назад и сосредотачиваемся на слове « Машинное обучение», становится понятным, что ваша модель будет учиться на ваших данных и делать прогнозы. И когда машины делают такие прогнозы, мы также должны их проверять.

Набор обучающих данных используется для обучения модели. Модель учится на этих данных. И тестовый набор используется для проверки того, насколько точно наша модель может предсказывать.

Мы снова импортируем библиотеку из scikit под названием test\_train\_split из класса model.selection.

| from sklearn.model\_selection import train\_test\_split |
| --- |

Теперь нам нужны 4 переменные: X\_train, X\_test, y\_train, y\_test.

Давайте разделим наш набор данных на обучающий и тестовый наборы.

| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2) |
| --- |

***Примечание.*** *test\_size + train\_size = 1*

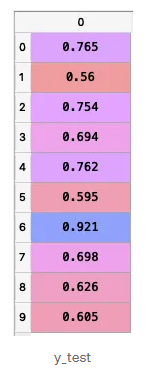
Здесь мы разделили 20% данных на тестовый набор и 80% данных на обучающие данные. В настоящее время не существует эмпирического правила как такового, согласно которому вы должны делить свой набор данных на соотношение 80–20, но это очень распространенная практика.

Я прошел онлайн-курс Эндрю Нг по машинному обучению. Его рекомендация была:

Обучение: 60% Перекрестная проверка(Cross-validation): 20% Тестирование: 20% .

Сейчас это просто рекомендация! В конце концов, все зависит от имеющихся данных.

Теперь, если вы видите наш набор данных лучших песен 2018 года, это то, что происходит. У нас было 50 строк данных, 20% от 50 будет 10, поэтому 40 подходов — это тренировка, а 10 — тест.



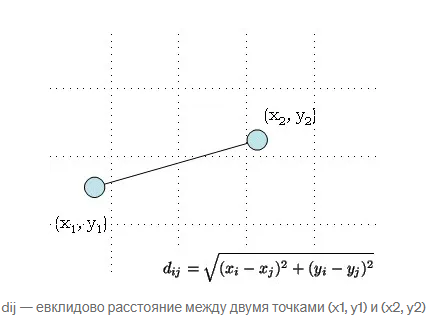
1. **Масштабирование функций**

Это последний шаг в предварительной обработке данных — масштабирование функций. Масштабирование признаков — проверенный метод ограничения диапазона переменных, чтобы их можно было сравнивать на общих основаниях. Например, если вы видите этот набор данных.



Зарплата колеблется от 48 000 до 83 000. Точно так же и возраст колеблется от 27 до 50. Если внимательно посмотреть, одна характеристика измеряется десятками, а другая тысячами. Большинство алгоритмов машинного обучения учитывают только величину значений, а не единицы этих значений.

Многие алгоритмы машинного обучения основаны на так называемом евклидовом расстоянии.

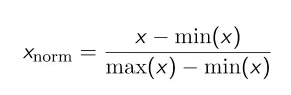


Скажем, если мы возьмем расстояние между (72000–48000)² = 576000000 и (44–27)² = 289. 576000000 явно преобладает над 289, и мы не делаем этого. Теперь мы должны ограничить диапазон переменных, чтобы их можно было сравнивать на общих основаниях!

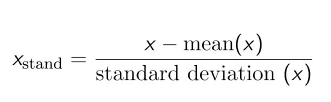
Существует несколько способов масштабирования данных. Есть два способа сделать это.

**Нормализация**

Нормализация масштабирует функцию между 0,0 и 1,0, сохраняя их пропорциональный диапазон друг к другу.



Стандартизация обычно означает, что диапазон значений «стандартизирован» для измерения того, сколько стандартных отклонений значения от его среднего значения. Стандартизация преобразует ваши данные таким образом, чтобы результирующее распределение имело среднее значение 0 и стандартное отклонение 1.



Для этого нам нужно импортировать StandardScaler из библиотеки предварительной обработки scikit.

| from sklearn.preprocessing import StandardScaler  sc\_X = StandardScaler() |
| --- |

Теперь все, что нам нужно сделать, это подогнать и преобразовать наш набор X\_Training.

Краткое примечание. Для тестового набора необходимо подобрать его, а затем преобразовать, но для обучающего набора требуется простое преобразование.

Теперь вы увидите, что данные будут в одном диапазоне.

Это были общие шаги для предварительной обработки данных. В зависимости от того, какой набор данных у вас есть, вам может понадобиться выполнить эти шаги, а может и не потребоваться.

**Задачи для самостоятельной работы**

1. Прочитайте (смотрит весь плейлиста) [1-3] и реализуйте на компьютере (например, в Google Colab), а затем подготовьте презентацию.
2. Генерируйте несколько наборов данных и применяйте к ним все сем шагов предварительной обработки данных.
3. Загрузите следующие датасеты и примените ко всем из них все шаги предварительной обработки данных.
4. Air Quality Data Set: https://archive.ics.uci.edu/ml/datasets/Air+Quality
5. Air Quality Index: https://www.epa.gov/outdoor-air-quality-data/download-daily-data
6. Australian Weather Dataset: <http://www.bom.gov.au/climate/data/>
7. Bitcoin Historical Data: https://www.kaggle.com/mczielinski/bitcoin-historical-data
8. Bitcoin Price: https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory
9. Climate Data Online (CDO): https://www.ncdc.noaa.gov/cdo-web/webservices/v2
10. Electricity Load Demand: https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014#
11. Energy Consumption: https://www.kaggle.com/robikscube/hourly-energy-consumption
12. Exchange Rates: https://www.kaggle.com/brunotly/foreign-exchange-rates-per-dollar-20002019
13. FRED-MD: A Monthly Database for Macroeconomic Research: https://research.stlouisfed.org/econ/mccracken/fred-databases/
14. Global Climate Change Data: https://data.giss.nasa.gov/gistemp/
15. Google Trends: https://trends.google.com/trends/
16. Household Electric Power Consumption Data Set: https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
17. Human Activity Recognition Using Smartphones Data Set: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones
18. Human Activity Recognition Using Smartphones: <http://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones>
19. M4 Time Series Forecasting Competition: <https://github.com/M4Competition/M4-methods>
20. M5 Forecasting Competition: https://www.kaggle.com/c/m5-forecasting-accuracy/data
21. NOAA Climate Data Online: https://www.ncdc.noaa.gov/cdo-web/webservices/v2
22. Numenta Anomaly Benchmark: https://github.com/numenta/NAB
23. S&P 500 Stock Data Set: https://www.kaggle.com/camnugent/sandp500
24. The Great Energy Predictor Kaggle Competition: https://www.kaggle.com/c/ashrae-energy-prediction/data
25. Time Series Datasets for Machine Learning: https://github.com/laiguokun/multivariate-time-series-data
26. Traffic Prediction Challenge: https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data
27. Twitter Sentiment Analysis Dataset: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/
28. Walmart Sales Forecasting Kaggle Competition: https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data
29. Wikipedia Web Traffic Time Series: https://www.kaggle.com/c/web-traffic-time-series-forecasting/data
30. Yahoo Finance: <https://finance.yahoo.com/>

**Литература**

1. <https://habr.com/ru/articles/511132/>
2. <https://www.youtube.com/watch?v=O1-mdHIQnP0>
3. <https://www.youtube.com/watch?v=ZddUwo4R5ug&list=PLBSCvBlTOLa8_IHxi8x2P9ySJgHWGhppn>
4. <https://academy.yandex.ru/handbook/ml/article/about> - Хороший курс по ML
5. <https://lena-voita.github.io/nlp_course.html> - Хороший курс по NLP
6. <https://www.analyticsvidhya.com/blog/2021/08/data-preprocessing-in-data-mining-a-hands-on-guide/>
7. <https://medium.com/@yogeshojha/data-preprocessing-75485c7188c4>
8. <https://medium.com/@ben.putney/list/predictive-modeling-w-python-e3668ea008e1>
9. <https://destingong.medium.com/list/practical-guides-to-machine-learning-a877c2a39884>