



hypo69 / 1001-python-ru



<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security 2



master



[1001-python-ru](#) / [articles](#) / [google-colab](#) / [colab_cheat_sheet.ipynb](#)



hypo69 Создано в Colab

e252d8e · 6 minutes ago



1478 lines (1478 loc) · 82.1 KB



master

[1001-python-ru](#) / [articles](#) / [google-colab](#) / [colab_cheat_sheet.ipynb](#)

↑ Top

Preview

Code

Blame



Raw



Jupyter Notebook Colab: Шпаргалка по использованию интерактивной среды для анализа данных, машинного обучения и не только

Что такое Jupyter Notebook?

Jupyter Notebook — это **веб-приложение с открытым исходным кодом**, которое позволяет создавать интерактивные документы, объединяющие код (например, Python, R, Julia), визуализации, текст и мультимедиа. Изначально проект назывался *IPython Notebook* и был создан в 2014 году как часть экосистемы Python. Сегодня Jupyter поддерживает **более 40 языков программирования**, а его название образовано от комбинации **Julia, Python** и **R**.

Ключевые особенности:

- **Интерактивное выполнение кода** по ячейкам.
- Поддержка **Markdown** и *L^AT_EX* для документации:

Тождество Эйлера: $e^{i\pi} + 1 = 0$

Формула Стирлинга: $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

Основная теорема арифметики:

Любое целое число $n > 1$ может быть представлено единственным образом в виде произведения простых чисел, с точностью до порядка сомножителей:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k},$$

- Визуализация данных прямо в документе (графики, таблицы, анимации).
- Экспорт в форматы HTML, PDF, LaTeX, презентации.
- **Виджеты** для создания интерактивных интерфейсов.
- Интеграция с облачными сервисами и инструментами Big Data.

Установка и настройка

1. **Установка через pip** (требуется Python 3.3+):

```
pip install jupyter
```

2. **Запуск сервера:**

```
jupyter notebook
```

После этого в браузере откроется интерфейс Jupyter на

`http://localhost:8888 .`

3. **Jupyter Lab** (расширенная версия):

```
pip install jupyterlab
jupyter lab
```

Интерфейс и базовое использование

1. Новый пункт
2. Новый пункт

Основные элементы:

1. **Панель управления** (Dashboard): список файлов и директорий.
2. **Блокнот (Notebook)**: состоит из ячеек двух типов:
 - **Code** — для написания кода.
 - **Markdown** — для текста, формул, изображений.

Горячие клавиши:

- **Shift + Enter** — выполнить ячейку.
- **Esc + A/B** — добавить ячейку выше/ниже.
- **Esc + M/Y** — сменить тип ячейки на Markdown/Code.

Горячие клавиши (в Colab):

- **Ctrl+Enter** (или **Shift+Enter**): Выполнить ячейку и перейти к следующей.
- **Ctrl+M A**: Вставить ячейку выше текущей.
- **Ctrl+M B**: Вставить ячейку ниже текущей.
- **Ctrl+M M**: Преобразовать ячейку в Markdown.
- **Ctrl+M Y**: Преобразовать ячейку в Code.
- **Ctrl+M D**: Удалить ячейку.

Примеры использования

1. Анализ данных о пассажирах Титаника с Pandas и Seaborn

В этом примере я загружаю данные из github. Источник:

<https://github.com/datasciencedojo/datasets/blob/master/titanic.csv>

Поля таблицы данных о пассажирах Титаника

- **PassengerId**: Уникальный идентификатор для каждого пассажира. Это просто номер пассажира в базе данных.

- **Survived:** указывает, выжил пассажир (1) или нет (0).
- 0: Пассажир погиб.
- 1: Пассажир выжил.
- **Pclass:** Класс билета пассажира. Обычно это соответствует уровню комфорта и расположению каюты.
- 1: Первый класс (самый дорогой и комфортный).
- 2: Второй класс.
- 3: Третий класс (самый дешевый).
- **Name:** Имя пассажира (включая звание, например, Mr., Mrs., Miss.).
- **Sex:** Пол пассажира (male - мужской, female - женский).
- **Age:** Возраст пассажира в годах. Обратите внимание, что в реальных данных часто бывают пропущенные значения возраста (NaN).
- **SibSp:** Количество братьев/сестер (siblings) и супругов (spouses) на борту судна.
- **Parch:** Количество родителей (parents) и детей (children) на борту судна.
- **Ticket:** Номер билета пассажира.
- **Fare:** Стоимость билета пассажира (в британских фунтах того времени).
- **Cabin:** Номер каюты пассажира (если известен). Многие значения пропущены.
- **Embarked:** Порт, в котором пассажир сел на корабль.
- C: Cherbourg (Шербур, Франция).
- Q: Queenstown (Квинстаун, Ирландия).
- S: Southampton (Саутгемптон, Англия).

Загрузка данных

```
In [ ]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Загрузка данных с GitHub
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
titanic_df = pd.read_csv(url)

# Вывод первых строк таблицы
print("Первые 5 строк данных:")
print(titanic_df.head())
```

Основная статистика

- **count** : Количество не-пустых (не-NaN) значений в столбце. Это показывает, для скольких пассажиров доступна информация по данному признаку. Например, если **count** для столбца **Age** меньше, чем для других столбцов, это означает, что есть пропущенные значения возраста.
- **mean** : Среднее значение столбца. Это сумма всех значений в столбце, деленная на количество значений (**count**). Полезно для понимания "типичного" значения признака.
 - Для **Survived** : Показывает долю выживших пассажиров (0.383838 означает, что примерно 38% пассажиров выжили).
 - Для **Pclass** : Показывает средний класс билета (2.308642 говорит о том, что большинство пассажиров были в 2-м или 3-м классе).
 - Для **Age** : Показывает средний возраст пассажиров (29.699118 лет).
 - Для **SibSp** : Показывает среднее количество братьев/сестер и супругов на борту (0.523008).
- **std** : Стандартное отклонение. Это мера разброса данных вокруг среднего значения. Большое стандартное отклонение указывает на то, что значения в столбце сильно разбросаны, а маленькое - на то, что они сконцентрированы вокруг среднего.
 - Например, большое стандартное отклонение для возраста (14.526497) говорит о том, что возраст пассажиров сильно варьировался.
- **min** : Минимальное значение в столбце. Показывает наименьшее значение признака.
 - Для **Survived** : 0 (кто-то не выжил).
 - Для **Pclass** : 1 (кто-то был в первом классе).
 - Для **Age** : 0.42 (самый маленький возраст - меньше года).
 - Для **SibSp** : 0 (у кого-то не было братьев/сестер или супругов на борту).
- **25%** : 25-й процентиль (или первый квартиль). Это значение, ниже которого находится 25% данных. Полезно для понимания распределения данных.
 - Например, 25% пассажиров младше 20.125 лет.
- **50%** : 50-й процентиль (или второй квартиль, или медиана). Это значение, ниже которого находится 50% данных. Медиана - это "середина" данных. Она менее чувствительна к выбросам, чем среднее значение.
 - Например, половина пассажиров младше 28 лет.
- **75%** : 75-й процентиль (или третий квартиль). Это значение, ниже которого находится 75% данных.

- например, 15% пассажиров младше 38 лет.
- **max** : Максимальное значение в столбце. Показывает наибольшее значение признака.
 - Для **Survived** : 1 (кто-то выжил).
 - Для **Pclass** : 3 (кто-то был в третьем классе).
 - Для **Age** : 80 (самый старший возраст).
 - Для **SibSp** : 8 (у кого-то было 8 братьев/сестер или супругов на борту).

Вместе, эти статистики дают общее представление о распределении данных в каждом столбце, что полезно для:

- Обнаружения пропущенных значений (сравнение **count** с общим количеством строк в **DataFrame**).
- Выявления выбросов (сравнение **min** и **max** со средним и квантилями).
- Понимания типичных значений и разброса данных.
- Принятия решений о предварительной обработке данных (например, нужно ли заполнять пропущенные значения возраста и как это лучше сделать).

```
In [ ]: # Описательная статистика
print("\nОписательная статистика:")
print(titanic_df.describe())
```

Визуализация

Визуализация распределения выживших

```
In [ ]: # Визуализация распределения выживших
sns.countplot(x='Survived', data=titanic_df)
plt.title('Распределение выживших')
plt.show()
```

Зависимость выживания от класса билета

```
In [ ]: # Зависимость выживания от класса билета

sns.countplot(x='Pclass', hue='Survived', data=titanic_df)
"""
Args:
- `x`: str` указывает, что столбец 'Pclass' будет использоваться в каче
- `hue`: параметр, который добавляет еще одно измерение к графику. Он
  указанного столбца и отображает их разными цветами.
- `data`: DataFrame` указывает, что данные будут использоваться из обы
"""

plt.title('Выживаемость по классам билетов')
plt.show()
```

Гистограмма возраста

```
In [ ]: # Гистограмма возраста
sns.histplot(titanic_df['Age'].dropna(), kde=True) # Dropna чтобы избави
"""kde=True: аргумент функции sns.histplot указывает, что на гистограмме
плотности ядра (Kernel Density Estimate - KDE). KDE - это сглаженная кри
которая аппроксимирует распределение данных. Она помогает визуальнo оцен
особенно если данные не идеально соответствуют какой-либо известной форм
"""

plt.title('Распределение возраста пассажиров')
plt.show()
```

```
In [ ]: # Зеркальная диаграмма: слева – мужчины, справа – женщины
# 1. Предобработка
df = titanic_df[['Sex', 'Age']].dropna() # Оставим только пол и возраст.

# 2. Формируем возрастные группы по 10 лет
bins = list(range(0, 90, 10)) # 0-9, 10-19, ..., 80-89
labels = [f'{i}-{i+9}' for i in bins[:-1]]
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)

# 3. Группируем по полу и возрасту
grouped = df.groupby(['AgeGroup', 'Sex']).size().unstack(fill_value=0)

# 4. Данные для графика
age_groups = grouped.index.astype(str)
male_counts = -grouped['male'] # Отрицательные значения для зеркального
female_counts = grouped['female']

# 5. Построение пирамиды
fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(age_groups, male_counts, color='skyblue', label='Male')
ax.barh(age_groups, female_counts, color='lightcoral', label='Female')

# Настройка внешнего вида
ax.set_xlabel('Number of passengers')
ax.set_title('Titanic: Population Pyramid by Age and Sex')
ax.legend()
ax.set_xlim(-max(abs(male_counts.max()), female_counts.max()) - 5,
             max(abs(male_counts.max()), female_counts.max()) + 5)

# Добавим подписи
for i, age in enumerate(age_groups):
    ax.text(male_counts[i] - 1, i, str(abs(male_counts[i])), ha='right',
    ax.text(female_counts[i] + 1, i, str(female_counts[i]), ha='left', v

plt.tight_layout()
plt.show()
```

2. Анализ набора данных Diamonds .

Диаграмма набор данных "Diamonds" который содержит информацию о

л использую набор данных `Diamonds`, который содержит информацию о бриллиантах (размер, цвет, чистота, цена и т.д.). Этот набор данных доступен в библиотеке `Seaborn`.

Поля таблицы `Diamonds` :

- **carat**: Вес бриллианта в каратах.
- **cut**: Качество огранки (Fair, Good, Very Good, Premium, Ideal).
- **color**: Цвет бриллианта (от J - наименее желательный, до D - бесцветный и самый желательный).
- **clarity**: Чистота бриллианта.
(мера отсутствия включений и пятен, от I1 - худшая, до IF - безупречная).
- **depth**: Общая высота бриллианта (измеряется от верха до калетты) деленная на средний диаметр.
- **table**: Ширина верхней плоской грани бриллианта относительно его ширины.
- **price**: Цена бриллианта в долларах США.
- **x**: Длина в мм.
- **y**: Ширина в мм.
- **y**: Ширина в мм.
- **z**: Глубина в мм.

Информация о таблице

```
In [ ]: import seaborn as sns
import pandas as pd

# Загрузка набора данных Diamonds
diamonds = sns.load_dataset('diamonds')

# Вывод первых 5 строк
print("Первые 5 строк данных:")
print(diamonds.head())

# Информация о типах данных и пропущенных значениях
print("\nИнформация о типах данных и пропущенных значениях:")
print(diamonds.info())

# Описательная статистика
print("\nОписательная статистика:")
print(diamonds.describe())
```

1. Фильтрация

```
In [ ]: # 1. Фильтрация: Выбор бриллиантов с ценой больше 5000 долларов
expensive_diamonds = diamonds[diamonds['price'] > 5000]
print("\nБриллианты с ценой больше 5000 долларов:")
print(expensive_diamonds.head())
```

2. Группировка


```
In [ ]: # 2. Группировка: Средняя цена бриллиантов по качеству огранки
average_price_by_cut = diamonds.groupby('cut')['price'].mean()
print("\nСредняя цена бриллиантов по качеству огранки:")
print(average_price_by_cut)
```

2.1. Столбчатая диаграмма

```
In [ ]: # 2.1. Столбчатая диаграмма (Bar plot)
plt.figure(figsize=(8, 6))
average_price_by_cut.sort_values().plot(kind='bar', color='skyblue') # 
plt.title('Средняя цена бриллиантов по качеству огранки', fontsize=14)
plt.xlabel('Качество огранки', fontsize=12)
plt.ylabel('Средняя цена', fontsize=12)
plt.xticks(rotation=45, ha='right') # Поворот меток оси X
plt.tight_layout()
"""
Функция plt.tight_layout() автоматически корректирует интервалы между под-
тобы они лучше помещались в отведенное пространство. Она предназначена для
элементами графика, такими как заголовки, метки осей, и сами графики.
Более подробно:
- Автоматическая регулировка интервалов: Функция анализирует размеры и по-
заголовков, меток осей, легенд, цветовых шкал и т.д.) и пытается под-
чтобы избежать наложения.
- Обработка сложных макетов: Особенно полезна для графиков с множеством
или графиками с длинными подписями и заголовками.
- Оптимизация использования пространства: plt.tight_layout() старается ма-
для отрисовки графиков, что часто приводит к более читабельному и ви-

Применение до или после отрисовки: Обычно plt.tight_layout() вызывается
но до сохранения фигуры в файл или отображения на экране (plt.show()).
Вы можете вызывать ее повторно после изменения размеров элементов график:
"""
plt.show()
```

2.2. Горизонтальная столбчатая диаграмма

```
In [ ]: # 2.2. Горизонтальная столбчатая диаграмма (Horizontal bar plot)
plt.figure(figsize=(8, 6))
average_price_by_cut.sort_values().plot(kind='barh', color='lightcoral')
plt.title('Средняя цена бриллиантов по качеству огранки', fontsize=14)
plt.xlabel('Средняя цена', fontsize=12)
plt.ylabel('Качество огранки', fontsize=12)
plt.tight_layout()
plt.show()
```

2.3. Точечная диаграмма (Point plot)

```
In [ ]: # 2.3. Точечная диаграмма (Point plot)
plt.figure(figsize=(8, 6))
sns.pointplot(x=average_price_by_cut.index, y=average_price_by_cut.value)
plt.title('Средняя цена бриллиантов по качеству огранки', fontsize=14)
plt.xlabel('Качество огранки', fontsize=12)
plt.ylabel('Средняя цена', fontsize=12)
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()
plt.show()
```

3. Создание нового столбца

```
In [ ]: # 3. Создание нового столбца: Плотность бриллианта (масса/объем, приближ
diamonds['density'] = diamonds['carat'] / (diamonds['x'] * diamonds['y'])
print("\nПервые 5 строк с добавленной колонкой 'density':")
print(diamonds.head())
```

4. Визуализация

4.1. Распределение цены бриллиантов

```
In [ ]: # 4.1. Распределение цены бриллиантов
sns.histplot(diamonds['price'])
plt.title('Распределение цены бриллиантов')
plt.show()
```

4.2. Гистограммы для числовых признаков

```
In [ ]: # 4.2. Гистограммы для числовых признаков
numerical_features = ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i+1)
    sns.histplot(diamonds[feature], kde=True)
    plt.title(f'Распределение {feature}', fontsize=14)
plt.tight_layout()
plt.show()
```

```
In [ ]: # 6. Ящики с усами (Boxplots) для числовых признаков по качеству огранки
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i+1)
    sns.boxplot(x='cut', y=feature, data=diamonds)
    plt.title(f'{feature} по качеству огранки', fontsize=14)
plt.tight_layout()
plt.show()
```

```
In [ ]: # 7. Тепловая карта корреляции
correlation_matrix = diamonds[numerical_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Тепловая карта корреляции', fontsize=16)
plt.show()
```

```
In [ ]: # 8. Диаграммы рассеяния (Scatter plots) между ценой и другими числовыми
plt.figure(figsize=(15, 5))
```

```
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.scatterplot(x='carat', y='price', data=diamonds, alpha=0.5)
plt.title('Цена vs. Вес (carat)', fontsize=14)

plt.subplot(1, 3, 2)
sns.scatterplot(x='x', y='price', data=diamonds, alpha=0.5)
plt.title('Цена vs. Длина (x)', fontsize=14)

plt.subplot(1, 3, 3)
sns.scatterplot(x='y', y='price', data=diamonds, alpha=0.5)
plt.title('Цена vs. Ширина (y)', fontsize=14)

plt.tight_layout()
plt.show()
```

```
In [ ]: # 9. Столбчатые диаграммы (Countplots) для категориальных признаков
categorical_features = ['cut', 'color', 'clarity']
plt.figure(figsize=(15, 5))
for i, feature in enumerate(categorical_features):
    plt.subplot(1, 3, i+1)
    sns.countplot(x=feature, data=diamonds, palette='viridis')
    plt.title(f'Распределение {feature}', fontsize=14)
plt.tight_layout()
plt.show()
```

3. Классификация изображений с помощью MNIST и Keras

Этот пример показывает, как загрузить набор данных MNIST (рукописные цифры) из Keras, построить и обучить простую нейронную сеть для классификации изображений.

Функция `keras.datasets.mnist.load_data()` из Keras загружает набор данных MNIST. MNIST содержит 60 000 изображений рукописных цифр (от 0 до 9) для обучения и 10 000 изображений для тестирования.

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers

# Загрузка данных MNIST
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
"""
Функция `load_data()` возвращает два кортежа:

- `(x_train, y_train)` Обучающий набор данных:
  - `x_train` Массив NumPy, содержащий изображения для обучения (60 000 и:
  - `y_train` Массив NumPy, содержащий метки (цифры от 0 до 9), соответст

- `(x_test, y_test)` Тестовый набор данных:
  - `x_test` Массив NumPy, содержащий изображения для тестирования (10 000 и:
  - `y_test` Массив NumPy, содержащий метки для тестовых изображений.
"""
```

```
# Нормализация данных (масштабирование к диапазону [0, 1])
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
"""
```

Код нормализует значения пикселей в обучающем и тестовом наборах данных,

- `x_train.astype("float32")` преобразует тип данных массива `x_train` в `float32`. Это важно, потому что нейронные сети обычно работают лучше с числами с плавающей запятой.
- `/ 255` делит все значения пикселей в массиве `x_train` на 255. Значения в диапазоне от 0 до 255. Деление на 255 масштабирует значения пикселей в диапазон [0, 1].

Это называется нормализацией данных, и она помогает улучшить сходимость обучения.

```
# Преобразование формы данных для нейронной сети (выпрямление изображений)
x_train = x_train.reshape((-1, 28 * 28))
x_test = x_test.reshape((-1, 28 * 28))
"""
```

Код преобразует каждое изображение из матрицы 28x28 в вектор длиной 784. Это необходимо, потому что полносвязные слои (Dense layers) в нейронной сети требуют векторного ввода.

- `x_train.reshape((-1, 28 * 28))`: Изменяет форму массива `x_train`. Исходно `x_train` имеет форму (60000, 28, 28) - 60 000 изображений размером 28x28. `reshape((-1, 28 * 28))` преобразует его в массив с формой (60000, 784).

Args:

- `-1` означает, что NumPy должен автоматически вычислить размер первого измерения (в данном случае, количество изображений - 60 000).
- `28 * 28 = 784` указывает, что второе измерение должно иметь размер 784. Это означает, что каждое изображение 28x28 будет "выпрямлено" в вектор длиной 784.

```
# Определение модели
```

```
model = keras.Sequential([
    layers.Dense(512, activation="relu", input_shape=(28 * 28,)),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"), # 10 классов (цифры 0-9)
])
"""
```

- `keras.Sequential([...])` Создает последовательную модель Keras. После этого слои нейронной сети будут добавлены последовательно, один за другим.

Args:

- `layers.Dense(512, activation="relu", input_shape=(28 * 28,))` Добавляет полносвязный слой с 512 нейронами. 512 Количество нейронов (узлов) в слое.
- `activation="relu"` Функция активации ReLU (Rectified Linear Unit), которая возвращает 0 для отрицательных входов и сам вход для положительных.
- `input_shape=(28 * 28,)` Указывает форму входных данных для первого выпрямленного изображения. `input_shape` нужно указывать только для первого слоя.
- `layers.Dropout(0.5)` добавляет слой Dropout. `0.5` указывает долю нейронов, которые будут случайно "выключены" (игнорированы) во время обучения. Dropout - это метод регуляризации, который помогает предотвратить переобучение (когда модель хорошо работает на обучающих данных, но плохо на новых).
- `layers.Dense(10, activation="softmax")` добавляет полносвязный выходной слой с 10 нейронами. Так как MNIST имеет 10 классов (цифры 0-9), нам нужно 10 нейронов, по одному для каждого класса.
- `activation="softmax"` Функция активации Softmax. Softmax преобразует

```

- activation= Softmax - функция активации Softmax. Softmax преобразует
суммируются в 1. Например, если выход Softmax для данного изобра
[0.1, 0.05, 0.02, 0.08, 0.01, 0.7, 0.01, 0.01, 0.01, 0.01],
это означает, что модель предсказывает, что это изображение с вероят
"""

# Компиляция модели
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=
"""
- `model.compile(...)` конфигурирует модель для обучения.

- Args:
- `loss="sparse_categorical_crossentropy"`: функция потерь (loss funct
Функция потерь измеряет, насколько хорошо модель предсказывает прави
`sparse_categorical_crossentropy` - это подходящая функция потерь для
где метки являются целыми числами (в данном случае, цифры от 0 до 9)
- `optimizer="adam"`: оптимизатор. Оптимизатор определяет, как будут о
чтобы минимизировать функцию потерь. adam - это популярный и эффекти
- `metrics=["accuracy"]` метрики. Метрики используются для оценки прои
во время обучения и тестирования.
`accuracy` - это доля правильно классифицированных изображений.
"""

# Обучение модели
history = model.fit(x_train, y_train, epochs=2, batch_size=32, validation
"""
- `model.fit(...)` Обучает модель на обучающих данных.
Args:
- `x_train`, `y_train` Обучающие изображения и соответствующие метки.
- `epochs=2` Количество эпох (полных проходов по обучающему набору дан
тем больше времени модель будет учиться, но тем выше риск переобучен
- `batch_size=32` Размер пакета (batch size). Обучающие данные разделя
после обработки каждого пакета. Меньший размер пакета требует больше
- `validation_split=0.2` Доля обучающих данных, которая будет использо
(оценки производительности модели во время обучения). В данном случае
как валидационный набор. Это позволяет отслеживать, как хорошо мод
которые она не видела во время обучения.
Returns:
- `history: model.fit()` возвращает объект history, который содержит и
такую как значения функции потерь и метрик на каждой эпохе.
"""

# Оценка модели
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
"""
- `model.evaluate(...)` Оценивает производительность модели на тестовых
Args:
- `x_test`, `y_test` Тестовые изображения и соответствующие метки.
- `verbose=` Управляет выводом информации во время оценки. verbose=0 о:
Returns
- `loss`, `accuracy`: model.evaluate() возвращает функцию потерь и мет
"""
print(f"Точность на тестовых данных: {accuracy:.4f}")

# Вывод графика обучения
plt.plot(history.history['accuracy'], label='Точность на обучающих данных')
plt.plot(history.history['val_accuracy'], label='Точность на проверочных')
plt.xlabel('Эпоха')
plt.ylabel('Точность')
plt.legend()
plt.show()

```

4. Регрессия с использованием Boston Housing Dataset и Scikit-learn

В этом примере я использую Boston Housing Dataset для построения модели линейной регрессии. Этот набор данных доступен непосредственно из Scikit-learn.

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Загрузка данных о жилье в Бостоне из оригинального источника
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
"""
- `pd.read_csv(...)` функция из библиотеки Pandas, которая используется ,
(или файла с разделителями).

Args:
- `data_url` переменная, которая содержит URL-адрес, откуда загружаются:

- `sep="\s+"` аргумент, который задает разделитель между значениями в (
- `"\s+"` - это регулярное выражение, которое означает "один или неско
Это необходимо, потому что в файле данных значения разделены пробелами

- `skiprows=22` аргумент, который указывает, сколько строк нужно пропу
В данном случае, пропускаются первые 22 строки, так как они содержат

- `header=None` аргумент, который указывает, что в файле нет строки за

Returns:
- `raw_df:DataFrame`, полученный в результате чтения CSV-файла. raw_df
"""

data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
"""
Эта строка выполняет операцию по извлечению признаков (features) из сы
Она разделяет сырые данные на две части (строки с четными и нечетными инд
выбирает нужные столбцы из каждой части и объединяет их горизонтально, ч
Этот способ вызван особенностью хранения данных в файле.

Args:
- `raw_df.values` свойство DataFrame raw_df, которое возвращает данные
- `raw_df.values[::2, :]` операция среза (slicing) массива NumPy, кото
Эта часть кода выбирает строки, содержащие часть информации о приз
- `raw_df.values[1::2, :2]` операция среза, которая выбирает строки с
Эта часть кода выбирает другую часть информации о признаках (featu
- `np.hstack([...])` функция из NumPy, которая выполняет горизонтально
Она объединяет массивы по столбцам.

Returns:
- `data:NumPy` массив NumPy, представляющий собой матрицу признаков (fe
"""
```

```

target = raw_df.values[1::2, 2]

# Присваивание данных и целевой переменной X и y
X, y = data, target

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
"""
Эта строка разделяет данные на обучающий и тестовый наборы, используя 20%
и фиксированное случайное состояние для воспроизводимости.

- `train_test_split(...)` функция из библиотеки scikit-learn, которая ис
Args:
  - X: Матрица признаков (features).
  - y: Целевая переменная.
  - test_size=0.2 аргумент, который задает размер тестового набора. В да
а остальные 80% - для обучения.
  - random_state=42 аргумент, который задает случайное состояние (random
перед разделением. Установка `random_state` гарантирует, что при каж
разделение данных будет происходить одинаково, что позволяет получать

Returns:
  Функция train_test_split() возвращает четыре массива:

  - X_train: Матрица признаков для обучающего набора.
  - X_test: Матрица признаков для тестового набора.
  - y_train: Целевая переменная для обучающего набора.
  - y_test: Целевая переменная для тестового набора.

"""

# Создание и обучение модели линейной регрессии
model = LinearRegression()
"""

`LinearRegression()` класс из библиотеки scikit-learn, который представл
Создает экземпляр модели линейной регрессии.
"""

model.fit(X_train, y_train)
"""

Метод `fit()` находит оптимальные коэффициенты (веса) для линейной регре
которые минимизируют функцию потерь (например, среднеквадратичную ошибку

Args:
  - X_train: Матрица признаков для обучающего набора.
  - y_train: Целевая переменная для обучающего набора.
"""

# Получение предсказаний на тестовых данных
y_pred = model.predict(X_test)
"""

Метод `predict()` использует обученную модель для предсказания значений
тестовом наборе.
Он использует обученную модель для предсказания цен на жилье на основе т

Args:
  - `X_test` матрица признаков для тестового набора.
Returns:
  - `y_pred` переменная, в которую сохраняются предсказанные значения.
"""

```

```

# Оценка модели с использованием среднеквадратичной ошибки
mse = mean_squared_error(y_test, y_pred)
"""
`mean_squared_error(y_test, y_pred)` функция из библиотеки scikit-learn,
MSE - это метрика, которая измеряет среднюю квадратичную разницу между y
MSE позволяет оценить, насколько хорошо модель предсказывает цены на жил
Args:
    - y_test: Фактические значения целевой переменной для тестового набора
    - y_pred: Предсказанные значения целевой переменной для тестового набо
Returns:
    - значение MSE
"""
print(f"Среднеквадратичная ошибка: {mse:.2f}")

# Визуализация результатов
plt.scatter(y_test, y_pred)
plt.xlabel("Фактические значения")
plt.ylabel("Предсказанные значения")
plt.title("Фактические vs. Предсказанные цены на жилье")
plt.show()

```

4. Кластеризация с использованием Iris Dataset и K-means

Этот пример демонстрирует кластеризацию данных Iris Dataset с помощью алгоритма K-means.

```

In [ ]: from sklearn.datasets import load_iris
"""
Импорт функции `load_iris` из модуля `sklearn.datasets`.
`sklearn.datasets` содержит различные наборы данных, которые часто испол
`load_iris` - это функция, которая загружает набор данных Iris (Ирисы).
"""

from sklearn.cluster import KMeans
"""
KMeans - это класс, который реализует алгоритм K-means.
`sklearn.cluster` содержит различные алгоритмы кластеризации, включая K-
"""

import matplotlib.pyplot as plt

# Загрузка данных
iris = load_iris()
X = iris.data

# Создание и обучение модели K-means (3 кластера, так как в Iris Dataset
kmeans = KMeans(n_clusters=3, random_state=42)
"""
`KMeans(...)` Это конструктор класса KMeans, который создает экземпляр а
Args:
    - `n_clusters=3` аргумент, который указывает количество кластеров, на
      В данном случае, мы указываем n_clusters=3, так как набор данных Iri
      Каждый вид ирисов будет представлен своим кластером.
    - `random_state=42` аргумент, который задает случайное состояние (randi
      Установка random_state гарантирует, что при каждом запуске кода клас
      что позволяет получать воспроизводимые результаты.
"""

```



```

kmeans.fit(X)
"""
`kmeans.fit(...)` метод класса KMeans, который используется для обучения
Метод `fit()` находит оптимальное положение центроидов кластеров,
которые минимизируют сумму квадратов расстояний от каждой точки до ближа
Args:
    - X данные (значения признаков), на которых будет обучаться модель.
"""

# Получение меток кластеров для каждой точки данных
labels = kmeans.labels_
"""
`kmeans.labels_` атрибут обученной модели kmeans, который содержит метки
присвоенные каждой точке данных. Метка кластера - это целое число, котор
"""

# Визуализация кластеров (я использую первые два признака для простоты ви
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Кластеризация Iris Dataset с использованием K-means')
plt.show()

```

5. Использование TensorFlow Datasets

TensorFlow Datasets предоставляет множество готовых к использованию наборов данных, упрощая процесс загрузки и подготовки данных для TensorFlow.

```

In [ ]: import tensorflow_datasets as tfds
"""
Импортирую библиотеку `tensorflow_datasets`.
TensorFlow Datasets (TFDS) предоставляет готовые к использованию наборы ,
которые можно легко загрузить и использовать в TensorFlow.
"""

import tensorflow as tf
"""
TensorFlow - это библиотека для машинного обучения, разработанная Google
"""

# Загрузка набора данных cats_vs_dogs
(ds_train, ds_validation), ds_info = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:]'],
    as_supervised=True,
    with_info=True
)
"""
`tfds.load(...)` функция из TensorFlow Datasets, которая загружает набор
разделяет его на обучающий и проверочный наборы (80% / 20%), и сохраняет
и информацию о наборе данных в соответствующих переменных.

Args:
    - `cats_vs_dogs` Это имя набора данных, который нужно загрузить. `cats_
    - `split=['train[:80%]', 'train[80%:]']` определяет, как разделить обу

```

- ``train[:80%]`` использует первые 80% обучающего набора данных в качестве обучающего набора данных.
- ``train[80%:]`` использует оставшиеся 20% обучающего набора данных в качестве проверочного набора данных.
- ``as_supervised=True`` этот аргумент указывает, что набор данных должен представлять собой кортеж (`image, label`) - это метка класса (0 для кошки, 1 для собаки).
- ``with_info=True`` указывает, что вместе с набором данных должна быть информация о наборе данных (например, количество образцов, названия классов).

Returns:

Функция `tfds.load()` возвращает два значения:

- ``(ds_train, ds_validation)`` кортеж, содержащий обучающий набор данных и проверочный набор данных (`ds_validation`).
- ``ds_train`` и ``ds_validation`` являются объектами ``tf.data.Dataset``, которые позволяют эффективно загружать и обрабатывать большие наборы данных.
- ``ds_info`` объект, содержащий информацию о наборе данных.

"""

Функция для изменения размера изображений

def `resize_image(image, label):`

""" функция изменяет размер изображения до 150x150 пикселей, сохраняя метку класса.

Args:

- ``image`` - изображение (объект типа `tf.Tensor`)
- ``label`` - метка класса (объект типа `tf.Tensor`)

Returns:

измененное изображение (объект типа `tf.Tensor`) и его соответствующая метка класса.

"""

`image = tf.image.resize(image, (150, 150))`

return `image, label`

Применение функции изменения размера к данным

`ds_train = ds_train.map(resize_image)`

`ds_validation = ds_validation.map(resize_image)`

"""

Этот блок кода изменяет размер всех изображений в обучающем и проверочном наборах данных.

- ``ds_train.map(resize_image)`` Применяет функцию ``resize_image`` к каждому элементу обучающего набора данных ``ds_train``. Метод ``map()`` преобразует каждый элемент набора данных.
- ``ds_validation.map(resize_image)`` Применяет функцию ``resize_image`` к каждому элементу проверочного набора данных.

"""

Настройка параметров пакетной обработки и кэширования

`batch_size = 32`

`ds_train = ds_train.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)`

`ds_validation = ds_validation.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)`

"""

Этот блок кода устанавливает размер пакета равным 32 и настраивает пакетную обработку и предварительную выборку для повышения производительности обучения.

- ``batch_size = 32`` задает размер пакета (batch size). Размер пакета определяет количество элементов, которые будут обработаны одновременно.
- ``ds_train.batch(batch_size)`` группирует элементы обучающего набора данных в пакеты.
- ``.prefetch(buffer_size=tf.data.AUTOTUNE)`` использует предварительную загрузку пакетов для предотвращения простоев.

Аналогичные действия выполняются для проверочного набора данных.

"""

Пример отображения нескольких изображений из набора данных

def `show_batch(image_batch, label_batch):`

""" Функция принимает два аргумента: `image_batch` (пакет изображений) и `label_batch` (пакет меток классов).

Args:

- ``image_batch`` - пакет изображений (объект типа `tf.Tensor`)
- ``label_batch`` - пакет меток классов (объект типа `tf.Tensor`)

"""

`plt.figure(figsize=(10,10))`

"""Создает новую фигуру Matplotlib размером 10x10 дюймов."""

```

for n in range(25):
    """Цикл, который повторяется 25 раз (для отображения 25 изображений)
    ax = plt.subplot(5,5,n+1)
    """Создает подграфик (subplot) на фигуре.
    Фигура разделена на сетку 5x5, и текущий подграфик будет размещен в :

    plt.imshow(image_batch[n].numpy().astype("uint8"))
    """ Отображает изображение на текущем подграфике.
    - `image_batch[n]` Выбирает n-е изображение из пакета изображений.
    - `.numpy()` преобразует тензор TensorFlow в массив NumPy.
    - `.astype("uint8")`: Преобразует тип данных массива NumPy в uint8 (бе:
    что необходимо для правильного отображения изображений."""

    plt.title(ds_info.features['label'].int2str(label_batch[n].numpy()))
    plt.axis("off")
plt.show()

for image_batch, label_batch in ds_train.take(1):
    show_batch(image_batch, label_batch)

```

Расширенные возможности

1. Магические команды (работают и в Colab)

- `%timeit` : Измерение времени выполнения кода.
- `!ls` : Выполнение команд shell.
- `%%writefile my_script.py` : Сохранение содержимого ячейки в файл.

```

In [ ]: %timeit
import numpy as np
np.random.randn(1000, 1000)

```

2. Интерактивные виджеты

```

In [ ]: from ipywidgets import interact
import matplotlib.pyplot as plt
import numpy as np

def plot_sine(frequency=1, theta=0):

```