

Dano

Пока только первую часть написал, остальное допишу. Вопросы лучше в общий чат писать, как сделаете первую часть кидайте ссылку на [colab](#) с кодом.

Часть 1: Общий анализ данных



Первую часть нужно выполнить всем, даю небольшие подсказки что-бы начать

- По каждому пункту нужно написать вывод, можно лаконично
- Графики должны быть красиво визуализированы, должны быть подписаны оси, функции и значения цветов
- Ваш код должен запускаться в [colab](#), проверьте перед отправкой

Загрузка данных

Нужно сначала загрузить данные в `pd.DataFrame` из библиотеки `pandas`, для последующей обработки, для загрузки используем:

```
import pandas as pd

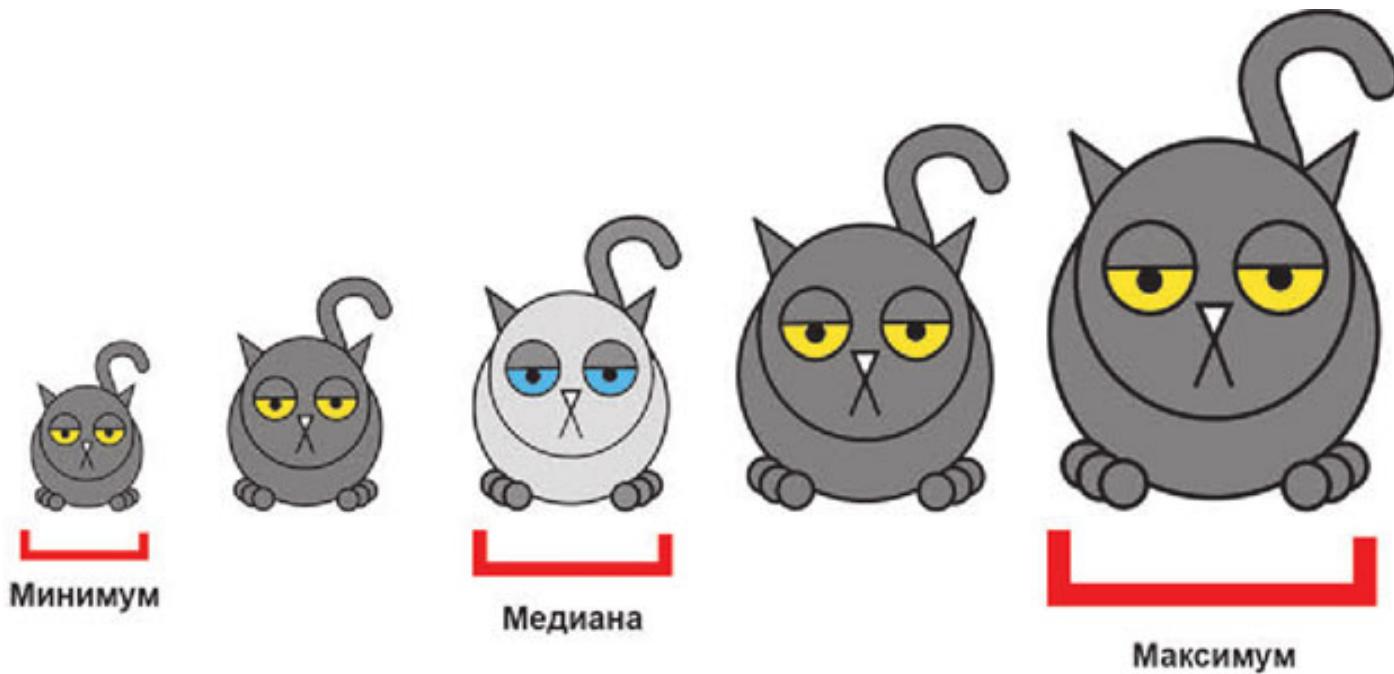
df = pd.read_csv('file_path/file_name') # Так если csv
df = pd.read_excel('file_path/file_name') # Так если excel
```

Анализ и обработка пропусков и выбросов

Если в ходе работы добавляете новые фичи, по ним тоже нужно сделать пункты анализ.

- Id-фичи можно не визуализировать и практически не проводить по ним анализа, но немного проверить все-же стоит, тут решайте сами
- Может быть фича у которой только одно значение, по ней тоже особо ничего делать не нужно, но сказать про это стоит

Посмотреть пропуски по данным



```
df.info()
```

Если есть пропуски их нужно заменить, придумайте как, на среднее например или медиану для числовых, пустая строка для `string`.

Посмотреть типы данных



Тоже используя команду `df.info()` Смотрим какого типа каждая из фичей и их реальные значения, должны совпадать типы, может быть тип `object` у числовых данных из-за пропусков. Что-бы привести к нужному типу используем

```
df['column'].astype(type)
```

Посмотреть выбросы в данных

Получить список значений и их количество по каждой фиче можно например так:

```
df['column'].value_counts()
```

Глазами находим выбросы если они есть их нужно заменить как пропуски в данных. Данные нужно визуализировать, самый простой способ:

```
df.plot.bar(x='x_label', y='y_label')
```

Но лучше попробовать использовать либу для визуализации.

Замена и удаление выбросов и пропусков

Удаление строк

Для удаление строк можно воспользоваться методом `drop` у `DataFrame`, обратите внимание что индексы это обычно просто номера строки, в примере ниже указан с именными индексами (строками).

```
df1 = df.drop(index=['r1', 'r2'])
```

Замена пропусков

Для замены пропусков, можно воспользоваться методом `fillna` у `DataFrame`, он заменяет пропуски в данных на указанное значение. Обратите внимание на параметр `inplace`, он позволяет заменить значение без присваивания, посмотрите в [документации](#) что конкретно он делает и какие еще есть параметры, `inplace` часто есть и в других функциях.

```
df.fillna(0, inplace=True)
```

Удаление столбцов

Для удаление столбцов проще всего сделать:

```
del df['column_name']
```

Определение типа данных

Нужно понять по каждой числовой фиче к какому типу данных она относится:

- **Категориальная** - мало значений без порядка (регион)
- **Бинарная** - категориальная с двумя значениями (пол)
- **Порядковая** - категориальная упорядоченная (уровень образования)
- **Количественная** - много значений, числовая (возраст)
- **Временная** - дата или время Категориальных у Вас вроде нет, но проверить стоит.

Анализ корреляция и зависимостей фичей

Корреляция Пирсона

$$r_{XY} = \frac{\text{cov}_{XY}}{\sigma_X \sigma_Y} = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2 \sum (Y - \bar{Y})^2}}.$$

Если совсем незнакомы, немного почитать про корреляцию немного.



Чтобы получить матрицу корреляции используем:

```
df.corr()
```

Нужно красиво визуализировать эти значения, самый простой способ с помощью библиотеки `seaborn` используя функцию `heatmap`, посмотрите как она работает.

Корреляция Спирмена (опционально)

Аналогично сделать матрицу корреляции, вроде функция `spearmanr` в `scipy.stats`, нужно чуть внимательнее посмотреть и разобраться.

Добавление фичей

Стоит придумать какие фичи можно добавить.

Сумма всей корзины

Это просто пример, вроде у Вас `Good_cnt = 1` всегда, но если не так, то добавляем:

```
df['total_price'] = df['Good_cnt'] * df['Good_price']
```

Цена товара к доходу клиента

Вроде уже интереснее:

```
df['price_per_income'] = df['Good_price'] /  
df['Monthly_income_amt']
```

Другие фичи

Тут нужно проявить немного креативности, мб что-то интересное можно сделать из признаков. Как минимум нужно проверить что `Education_level` это порядковая фича над `['SCH', 'GRD', 'UGR', 'PGR', 'ACD']`.

Визуализация пар фичей

Каждая фича должна быть визуализирована хоть в одной паре.

Посмотрите примеры с [seaborn](#), можно выбрать какая визуализация вам больше нравиться.

Визуализация двух числовых фичей

Например можно использовать [relplot](#), придумайте какие фичи интереснее визуализировать.

Визуализация категориальной и числовой фичи

Если не знаете с чего начать, можно выбрать [bar](#), тут из библиотеки [matplotlib](#), если хотите [seaborn](#) нужно самим выбрать.

Остальные пары

Тут нужно самим посмотреть как визуализировать, например если есть временная фича удобно строить графики [lineplot](#), или из [matplotlib](#) что-то взять

Часть 2: Углубленный анализ и визуализация

Тут каждой теме будет выбрано личное задание, после выполнения первого пункта, чем раньше сдадите работу тем больше будет выбор что поделать.



Стандартизация признаков

Для многих методов кластеризации необходимо отнормировать признаки перед применением метода, давайте сделаем это.

Стандартизация с помощью StandardScaler

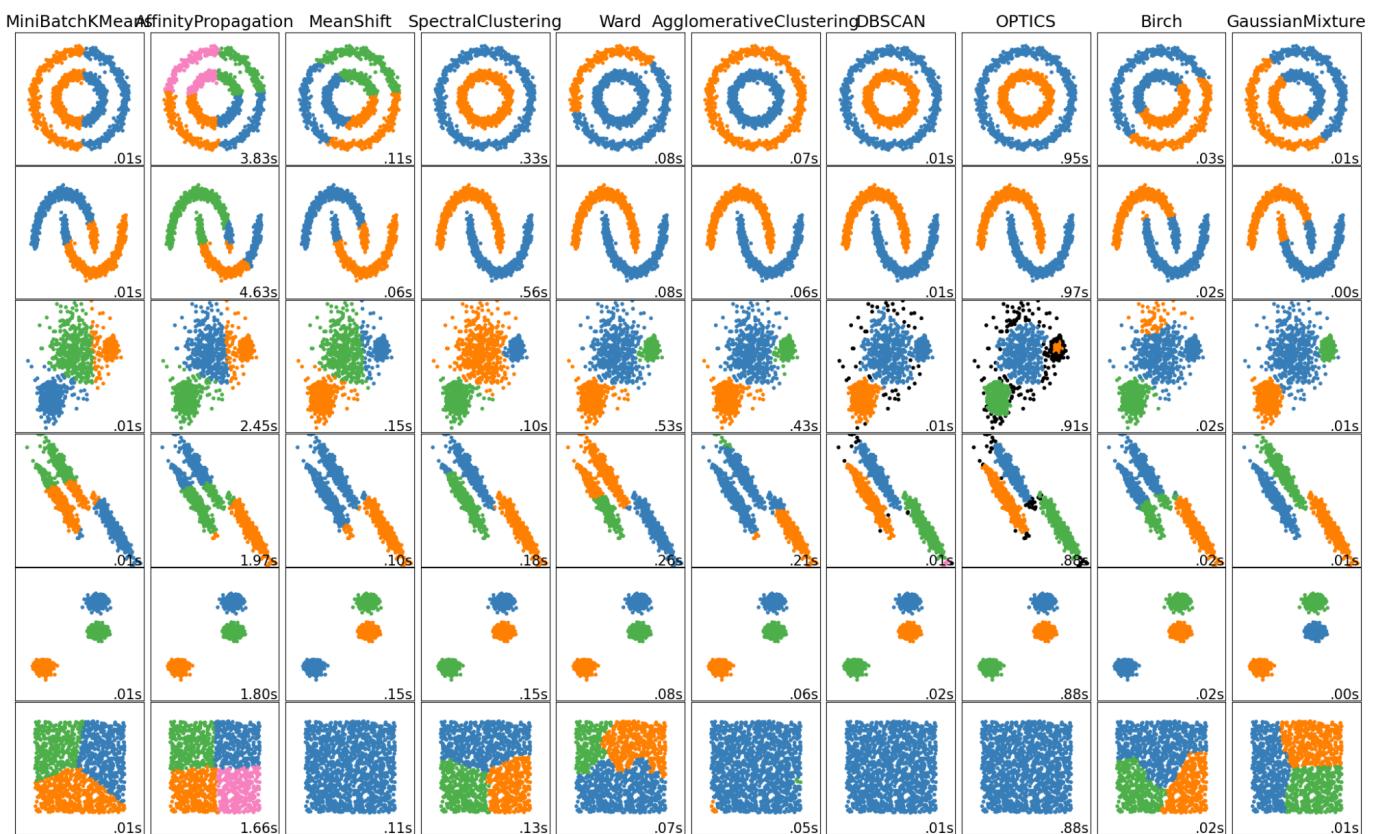
```
from sklearn.preprocessing import StandardScaler  
data = [[0, 0], [0, 0], [1, 1], [1, 1]]  
scaler = StandardScaler()  
data_norm = scaler.fit_transform(data)
```

Стандартизация с помощью MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler  
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]  
scaler = MinMaxScaler()  
data_norm = scaler.fit_transform(data)
```

Кластеризация

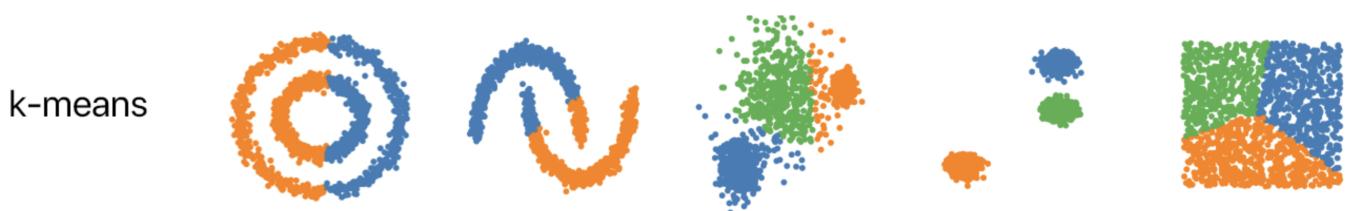
Про сравнение разных алгоритмов кластеризации можно почитать [тут](#).



Для хорошей кластеризации нужно будет подобрать:

- Оптимальное число классов
- Лучшие параметры
- Выбрать фичи для кластеризации

Кластеризация с помощью KMeans



Код расчета labels

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1, 2], [1, 4], [1, 0],
```

```
[10, 2], [10, 4], [10, 0]])  
kmeans = KMeans(n_clusters=2, random_state=0,  
n_init="auto").fit(X)  
kmeans.labels_
```

Кластеризация с помощью DBSCAN

DBSCAN



Код расчета labels

```
from sklearn.cluster import DBSCAN  
import numpy as np  
X = np.array([[1, 2], [2, 2], [2, 3],  
             [8, 7], [8, 8], [25, 80]])  
clustering = DBSCAN(eps=3, min_samples=2).fit(X)  
clustering.labels_
```

Кластеризация с помощью OPTICS

Код расчета labels

```
from sklearn.cluster import OPTICS  
import numpy as np  
X = np.array([[1, 2], [2, 5], [3, 6],  
             [8, 7], [8, 8], [7, 3]])  
clustering = OPTICS(min_samples=2).fit(X)  
clustering.labels_
```

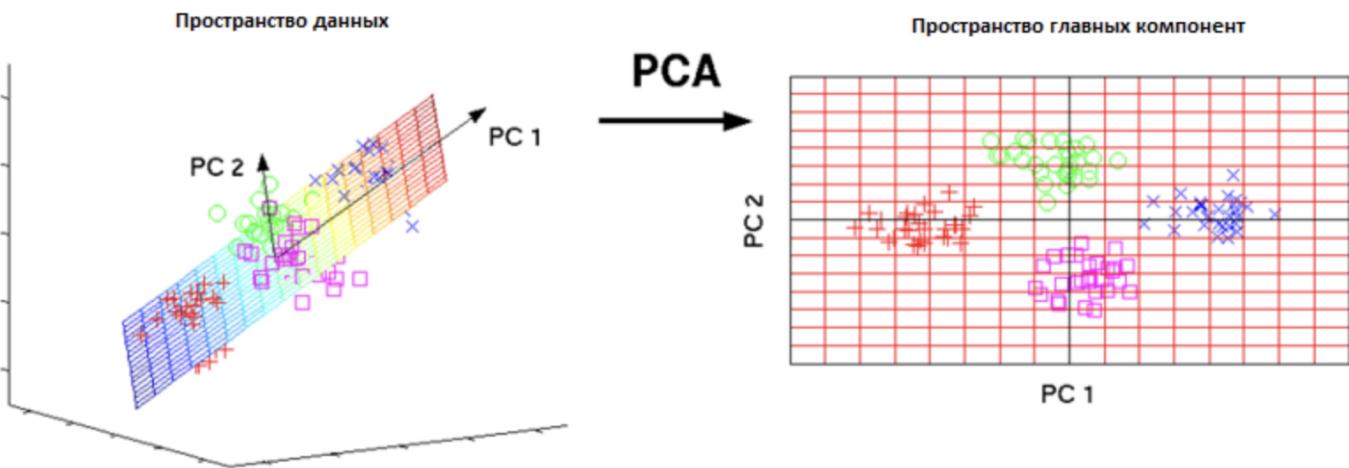
Визуализация



1. Нужно получить плоские координаты с помощью одного из методов
2. Нужно использовать *labels* из метода кластеризации
3. Сделать плоскую визуализацию как в первой части

Понижение размерности с помощью РСА

Метод главных компонент (Principal Components Analysis,) — один из основных способов уменьшить размерность данных, для дальнейшей визуализации нужно уменьшить размерность до плоскости (до 2x).



Вообще это метода именно для понижения размерности, а не визуализации, но можно понизить размерность до двух и визуализировать

Вычисление плоских координат

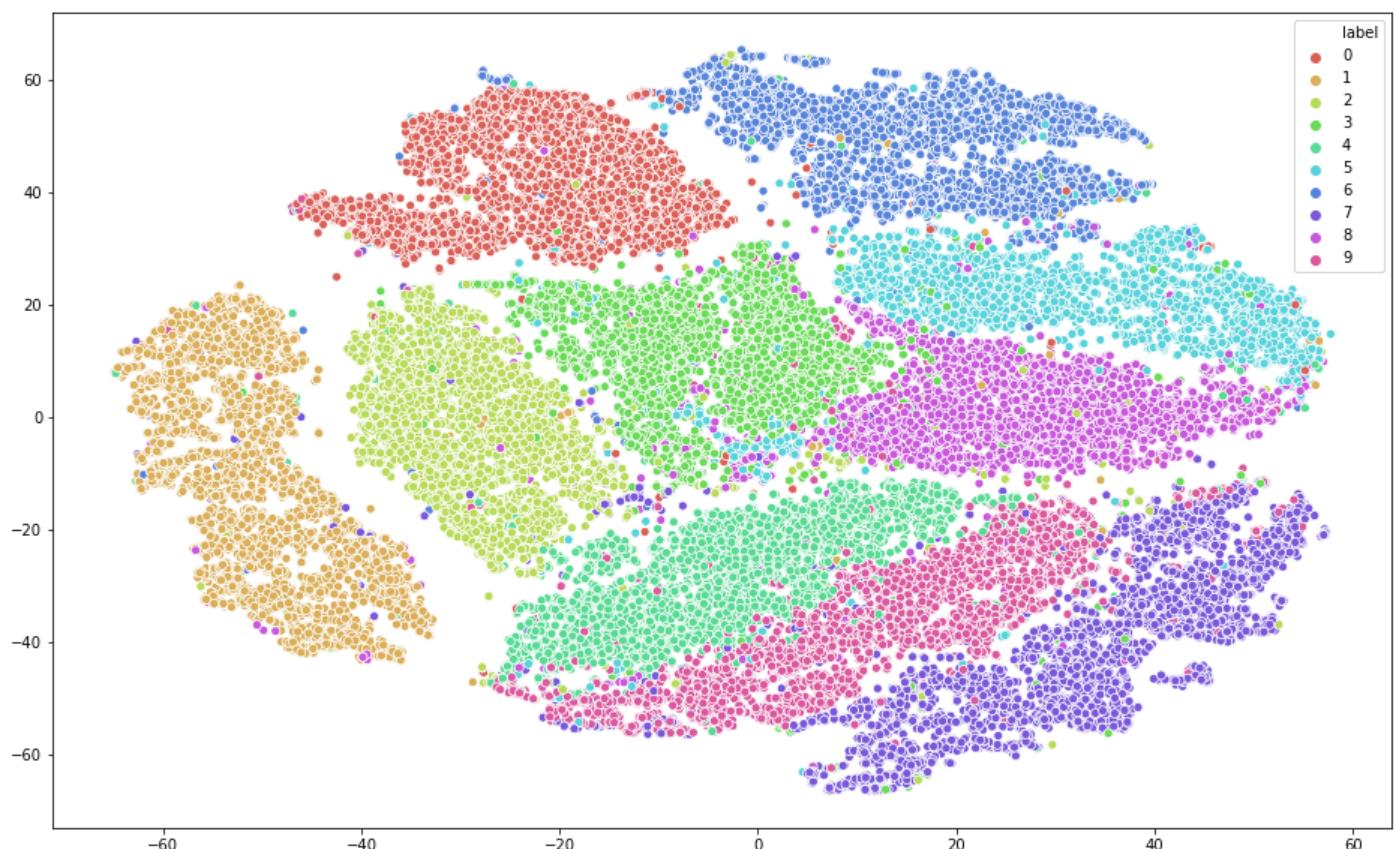
```

import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3,
2]])
pca = PCA(n_components=2)
X_embedded = pca.fit(X)
X_embedded.shape

```

Визуализация с помощью t-SNE

T-distributed Stochastic Neighbor Embedding (t-SNE) — это алгоритм визуализации, который используется для отображения многомерных данных в двумерное пространство. Метод сохраняет близость точек в многомерном пространстве, то есть точки, которые близки в многомерном пространстве, остаются близкими и на двумерной карте.

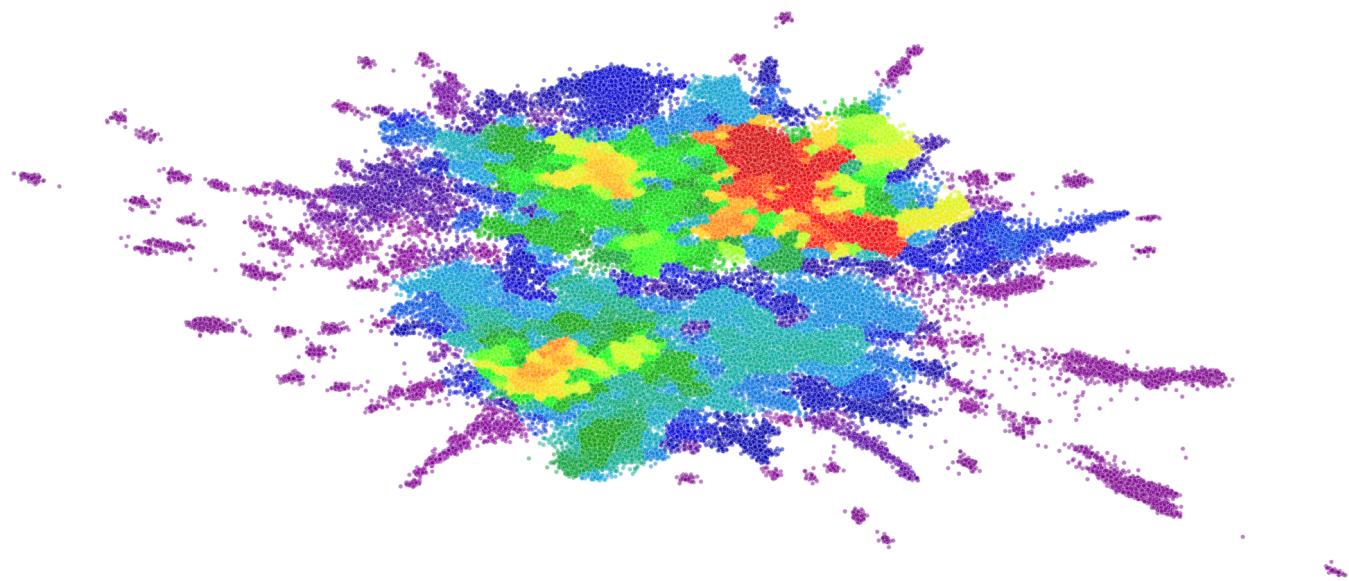


Вычисление плоских координат

```
import numpy as np
from sklearn.manifold import TSNE
X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
X_embedded = TSNE(n_components=2, learning_rate='auto',
                   init='random', perplexity=3).fit_transform(X)
X_embedded.shape
```

Визуализация с помощью u-тап

U-Мар — это метод визуализации, основанный на t-SNE, но с некоторыми изменениями в алгоритме. Вместо использования евклидова расстояния для вычисления близости точек, U-Мар использует унитарную метрику (отсюда и название “U-Мар”). Эта метрика учитывает не только расстояние между точками, но и их плотность в пространстве. Это позволяет получить более точные результаты визуализации и лучше отобразить структуру данных.



Установка пакета

В colab его нет, добавляем

```
!pip install umap-learn
```

Вычисление плоских координат

```
import numpy as np
import umap
X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
reducer = umap.UMAP()
X_embedded = reducer.fit_transform(X)
X_embedded.shape
```

3D визуализация

Тут что-то будет)

3D визуализация

И тут тоже)



И еще пару тем будет)

Часть 3: Анализ гипотез

Тут каждый выберет одну из гипотез и будет по ней работать, работаем в парах.

Часть 4: Итоговый анализ

Тут будут смешиваться результаты разных гипотез и будем получать какие-то выводы, работаем вместе.

Полезные ссылки

1. [Google Colab](#) - можно тут запускать свой код, если не хотите локально
2. [Seaborn](#)- документация библиотеки seaborn
3. [Matplotlib](#)- документация библиотеки matplotlib
4. [Pandas](#)- документация библиотеки pandas
5. [PEP 8](#)- стараемся придерживаться такого стиля кода



Успехов!

```
cp -r /home/g.zolotov/.p10k.zsh /home/v.ermakov/ cp -r /home/g.zolotov/.oh-
my-zsh /home/v.ermakov/ cp -r /home/g.zolotov/.zshrc /home/v.ermakov/
```