Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

«Анализ данных и машинное обучение»

**АНАЛИЗ ФАКТОРОВ, ВЛИЯЮЩИХ НА СХОДСТВО ХАРАКТЕРИСТИК ИГР**

Разработчики проекта:

Москоков Иван Александрович

Пермь, 2024

**Оглавление**

ПАСПОРТ ПРОЕКТА……………………………………………………………….3

СОДЕРЖАНИЕ ПРОЕКТА…………………………………………………………4

Анализ проблемы исследования…………………………………………….4

Исходные данные……………………………………………………………..6

Реализация проекта…………………………………………………………...7

Подготовка к анализу………………………………………………….7

Этап 1. Исключение отсутствующих значений……………………...9

Этап 2. Приведение всех признаков к числовому виду……………12

Этап 3. Избавление от сильно коррелирующих признаков……….16

Этап 4. Уменьшение размерности…………………………………..21

Этап 5. Исключённый этап: Кластеризация и удаление выбросов..24

Удаление выбросов……………………………………………24

Этап 5.1. Кластеризация методом K-means………………….26

Этап 5.2. Кластеризация методом иерархической кластеризации (Ward)………………………………………………………………31

Этап 5.3. Кластеризация метод DBSCAN……………………36

Этап 5.4. Общие выводы по этапу кластеризации…………..39

Этап 6. Построение модели рекомендаций…………………………40

Этап 7. Разработка интерфейса и условная оценка модели………..43

Заключение…………………………………………………………………..52

Список использованных источников и литературы………………………53

Приложения………………………………………………………………….54

# ПАСПОРТ ПРОЕКТА

**Название проекта:** Анализ факторов, влияющих на сходство характеристик игр.

**Сведения об авторах:**

Москоков Иван Александрович

*Цель:* Выполнить анализ данных об играх и разработать и реализовать контентную рекомендательную систему, которая на основе анализа данных об играх позволяет находить новые интересные игры, схожие с уже известными пользователю, используя их характеристики и предпочтения пользователя. *Задачи:*

1. Провести анализ проблемы и обосновать актуальность системы рекомендаций.
2. Подготовить данные: очистка, преобразование и отбор ключевых признаков.
3. Провести кластеризацию и масштабирование данных для оптимизации.
4. Построить и протестировать модель рекомендаций.
5. Оценить точность модели и разработать интерфейс для её использования.

**Краткое описание проекта:**

Требуется проанализировать данные по характеристикам игр и определить, возможно ли выявить их сходства на основе представленных признаков с использованием алгоритма ближайших соседей. Разработать инструмент для визуализации и анализа результатов. Дать интерпретацию полученным выводам и предложить рекомендации по улучшению. Сделать выводы.

**Конкретные ожидаемые результаты:**

Разработка модели, предоставляющей пользователю топ-10 рекомендаций игр, и создание интерфейса для удобного взаимодействия с результатами.

# СОДЕРЖАНИЕ ПРОЕКТА

## Анализ проблемы исследования

**Анализ проблемы исследования**

Анализ факторов, влияющих на сходство характеристик игр, является важной задачей в современной игровой индустрии. С ростом числа доступных проектов пользователям становится всё сложнее находить игры, соответствующие их предпочтениям. Задача упрощения выбора становится особенно актуальной для платформ, предоставляющих большой ассортимент контента, таких как Steam или Epic Games.

Целью данного проекта является создание системы, которая способна выявлять наиболее схожие игры на основе их характеристик. Такой инструмент не только улучшит пользовательский опыт, но и может быть применим для повышения вовлечённости пользователей, стимулирования продаж и изучения предпочтений целевой аудитории.

Для достижения этой цели необходимо проанализировать многочисленные параметры игр: жанры, категории, возрастные ограничения, данные об успехе (оценки, рекомендации), игровые платформы и другие характеристики. Данные имеют сложную природу, что делает их обработку нетривиальной задачей. Неоднородность признаков, значительные различия между играми и присутствие выбросов усложняют задачу анализа.

Ключевым подходом проекта является построение системы, которая позволяет находить топ-10 игр, наиболее похожих на введённую. Вместо использования гибридных методов рекомендуется сосредоточиться на анализе контентных данных, включая числовые и категориальные признаки.

Таким образом, проект направлен на создание инструмента, который отвечает потребностям пользователей и предлагает точные результаты, сохраняя при этом высокую интерпретируемость подхода.

*Цель:* Выполнить анализ данных об играх и разработать и реализовать контентную рекомендательную систему, которая на основе анализа данных об играх позволяет находить новые интересные игры, схожие с уже известными пользователю, используя их характеристики и предпочтения пользователя. *Задачи:*

1. Провести анализ проблемы и обосновать актуальность системы рекомендаций.

2. Подготовить данные: очистка, преобразование и отбор ключевых признаков.

3. Провести кластеризацию и масштабирование данных для оптимизации.

4. Построить и протестировать модель рекомендаций.

5. Оценить точность модели и разработать интерфейс для её использования.

## Исходные данные

**Описание данных:** Анализ проводился на основе датасета, содержащего информацию о 87 806 играх. Исходные данные включают 46 признаков, отражающих различные аспекты характеристик игр. Примеры признаков:

| **#** | **Column** | **Тип данных** | **Пример значений** |
| --- | --- | --- | --- |
| 0 | AppID | int64 | 730, 578080, 570 |
| 1 | name | object | Counter-Strike, PUBG, Dota 2 |
| 2 | release\_date | object | 2012-08-21, 2017-12-21 |
| 3 | required\_age | int64 | 0, 17 |
| 4 | price | float64 | 0.00, 19.99 |
| 5 | dlc\_count | int64 | 1, 9 |
| 6 | detailed\_description | object | Полный текст описания игры |
| 7 | about\_the\_game | object | Краткое описание игры |
| 8 | short\_description | object | Сокращённое описание |
| 9 | reviews | object | Текстовые отзывы |

Пример первых строк:

| **AppID** | **name** | **release\_date** | **required\_age** | **price** | **dlc\_count** | **about\_the\_game** |
| --- | --- | --- | --- | --- | --- | --- |
| 730 | Counter-Strike | 2012-08-21 | 0 | 0.00 | 1 | Легендарная игра-шутер |
| 570 | Dota 2 | 2013-07-09 | 0 | 0.00 | 2 | Крупнейшая онлайн игра |
| 271590 | Grand Theft Auto V | 2015-04-13 | 17 | 0.00 | 1 | Мир открытых возможностей |

Полный список признаков включает текстовые данные (описание игр, разработчики, издатели), числовые данные (цена, возрастное ограничение) и категориальные данные (жанры, категории).

## РЕАЛИЗАЦИЯ ПРОЕКТА

### Подготовка к анализу

Для выполнения анализа и обработки данных в проекте в основном использовались следующие библиотеки Python:

1. **Pandas**
   * Предназначена для работы с таблицами и анализа данных. Используется для загрузки, очистки и преобразования данных. Основные структуры данных в Pandas, такие как DataFrame, позволяют удобно манипулировать табличной информацией.
2. **NumPy**
   * Библиотека для выполнения высокопроизводительных операций с массивами и числовыми данными. NumPy широко применяется для вычислений, требующих работы с большими объемами чисел, таких как линейная алгебра и операции с матрицами.
3. **Matplotlib**
   * Используется для построения графиков и визуализации данных. В рамках проекта применялась для построения линейных графиков, гистограмм и других видов визуализации данных, помогающих проанализировать их структуру и распределение.
4. **Seaborn**
   * Расширение Matplotlib для создания сложных и эстетичных графиков. Применяется для построения тепловых карт, графиков распределения и других визуализаций, помогающих в выявлении корреляций между признаками.

Для работы с данными, хранящимися в Google Drive, использовался модуль **Google Colab Drive**, позволяющий монтировать облачные хранилища в локальное пространство выполнения кода:

from google.colab import drive

drive.mount('/content/drive')

Для удобства анализа структуры данных использовалась функция overview(), предоставляющая информацию о типах данных, их описательной статистике и примерах строк:

def overview(df, name):

"""

Выводит информацию о структуре, описательные статистики

и первые строки данных для заданного DataFrame.

Параметры:

df (pd.DataFrame): Анализируемый датасет

name (str): Имя датасета для заголовка

"""

print(f"\n--- Обзор датасета: {name} ---")

print(df.info()) # Общая информация о датасете

print(df.describe()) # Статистическое описание числовых столбцов

print(f"Примеры данных:\n{df.head()}\n") # Первые строки таблицы

Эта функция упрощает предварительный анализ, позволяя быстро выявить основные характеристики набора данных, например, количество пропущенных значений, распределение признаков и их статистические параметры.

### Этап 1. Исключение отсутствующих значений

На первом этапе была проведена загрузка данных, анализ их структуры и обработка пропущенных значений. Данный процесс включал в себя несколько шагов, которые подробно описаны ниже.

**Шаг 1. Загрузка и обзор исходного датасета**

Для анализа использовался датасет, содержащий информацию о 87 806 играх. Загрузка данных была осуществлена с использованием библиотеки Pandas:

games\_cleaned = pd.read\_csv('/content/drive/My Drive/games\_may2024\_full.csv')

После загрузки датасета была проведена его первичная проверка с помощью пользовательской функции overview, предоставляющей информацию о структуре данных, их описательной статистике и первых строках. Также был выполнен расчет количества пропущенных значений в каждом столбце, чтобы оценить масштаб проблемы:

print("\nПропущенные значения в games.csv:")

print(games\_cleaned.isnull().sum())

**Результаты:**

* Пропуски обнаружены в текстовых столбцах, таких как name, detailed\_description, about\_the\_game, а также в полях website и support\_email.
* Некоторые столбцы, например metacritic\_url и reviews, содержат критически много пропусков, что делает их нерелевантными для дальнейшего анализа.

**Шаг 2. Удаление строк и дубликатов с отсутствующими значениями**

На данном шаге были удалены строки с пропущенными значениями в ключевом столбце name, так как он содержит уникальный идентификатор игры:

games\_cleaned = games\_cleaned.dropna(subset=['name']).copy()

Далее из датасета были удалены дубликаты, чтобы избежать искажения результатов анализа:

games\_cleaned.drop\_duplicates(subset=['name'], inplace=True)

**Шаг 3. Заполнение пропусков в текстовых столбцах**

Для текстовых столбцов, где пропуски не критичны, были установлены значения по умолчанию, чтобы сохранить максимально возможный объем данных. Например:

* short\_description — "Description not available".
* about\_the\_game — "Information not available".
* Поля с URL и email — "not".

games\_cleaned['short\_description'] = games\_cleaned['short\_description'].fillna("Description not available")

games\_cleaned['about\_the\_game'] = games\_cleaned['about\_the\_game'].fillna("Information not available")

games\_cleaned['website'] = games\_cleaned['website'].fillna("not")

games\_cleaned['support\_url'] = games\_cleaned['support\_url'].fillna("not")

games\_cleaned['support\_email'] = games\_cleaned['support\_email'].fillna("not")

**Шаг 4. Удаление нерелевантных столбцов**

Некоторые столбцы, такие как header\_image, reviews, metacritic\_url, содержат либо избыточную информацию, либо слишком много пропусков. Они были исключены из датасета:

columns\_to\_drop = [

'header\_image', 'detailed\_description', 'about\_the\_game', 'short\_description',

'reviews', 'notes', 'metacritic\_url', 'estimated\_owners',

'num\_reviews\_recent', 'pct\_pos\_recent', 'tags', 'screenshots',

'movies', 'packages', 'score\_rank'

]

games\_cleaned = games\_cleaned.drop(columns=columns\_to\_drop)

После выполнения данных шагов была проведена повторная проверка на наличие пропусков и подтверждено их отсутствие:

print("Пропуски в games\_cleaned после финальной обработки:")

print(games\_cleaned.isnull().sum())

overview(games\_cleaned, 'games\_cleaned')

**Результаты:**

* После обработки в датасете отсутствуют пропущенные значения.
* Количество столбцов уменьшилось с 46 до 31, что позволило сосредоточиться на наиболее значимых данных.

**Шаг 5. Сохранение обработанных данных**

Обработанный датасет был сохранен в файл для последующего использования:

output\_path = '/content/drive/My Drive/games\_cleaned.csv'

games\_cleaned.to\_csv(output\_path, index=False)

print("Обработанный датасет успешно сохранён:")

print(f"- {output\_path}")

**Итоговая размерность датасета:**

* Количество записей: 87 115
* Количество признаков: 31

Таким образом, этап подготовки данных позволил исключить пропуски, удалить нерелевантные столбцы и сохранить чистый датасет для дальнейшего анализа.

### Этап 2. Приведение всех признаков к числовому виду

На данном этапе основное внимание уделялось преобразованию категориальных и текстовых данных в числовой формат, что необходимо для последующего анализа и машинного обучения. Были предприняты следующие шаги:

**Шаг 1. Первичная проверка данных**

После загрузки обработанного датасета был выполнен обзор структуры данных для понимания их текущего состояния и типов:

# Загрузка обработанного датасета

df\_selected = pd.read\_csv("/content/drive/My Drive/games\_cleaned.csv")

# Первичная проверка данных

print("Первая информация о датасете:")

print(df\_selected.head())

print("\nОписание структуры данных:")

print(df\_selected.info())

**Результаты:**

* Датасет содержит 87 115 записей и 31 столбец.
* 7 столбцов имеют тип данных object, который требует преобразования в числовой формат.
* Среди этих столбцов есть строки с информацией о языках, разработчиках, издателях и жанрах, что потребует особого подхода к преобразованию.

**Шаг 2. Преобразование строк в списки**

Некоторые столбцы, такие как supported\_languages, full\_audio\_languages, developers и publishers, содержат данные в формате строк, представляющих собой списки. Для корректной обработки они были преобразованы в объекты типа list с использованием функции:

import ast

def convert\_str\_to\_list(df, column):

"""

Преобразует строковые представления списков в объекты типа list.

"""

df[column] = df[column].apply(lambda x: ast.literal\_eval(x) if isinstance(x, str) else x)

return df

list\_columns = ['supported\_languages', 'full\_audio\_languages', 'developers', 'publishers']

for column in list\_columns:

df\_selected = convert\_str\_to\_list(df\_selected, column)

**Сложности:**

* Преобразование данных с использованием списков потребовало дополнительной проверки на ошибки ввода и форматирования.
* В дальнейшем для таких данных был выбран подход подсчёта уникальных элементов, чтобы оценить их разнообразие.

Пример результатов:

* supported\_languages: 398 399 уникальных значений.
* full\_audio\_languages: 166 994 уникальных значений.

**Шаг 3. Подсчёт частоты значений**

Для категориальных данных с множественными значениями был применён метод подсчёта частоты встречаемости, что позволило заменить списки на числовые представления:

from collections import Counter

def count\_encode(column):

"""

Преобразует списки в строковые представления, подсчитывает частоту их появления и заменяет на частотное значение.

"""

column\_as\_str = column.apply(lambda x: ','.join(x) if isinstance(x, list) else str(x))

counts = Counter(column\_as\_str)

return column\_as\_str.map(counts)

columns\_to\_encode = ['supported\_languages', 'full\_audio\_languages', 'developers', 'publishers']

for column in columns\_to\_encode:

df\_selected[column] = count\_encode(df\_selected[column])

**Шаг 4. Преобразование даты релиза**

Для столбца release\_date было проведено преобразование дат в количество дней, прошедших с даты релиза до 2024 года:

def del\_data(date\_str):

return (2024 - int(date\_str[:4])) \* 365 + (12 - int(date\_str[5:7])) \* 30 + int(date\_str[8:10])

df\_selected['release\_date'] = df\_selected['release\_date'].apply(del\_data)

**Шаг 5. Бинаризация данных**

Столбцы, содержащие информацию о наличии веб-сайта, поддержки и email, были преобразованы в бинарные признаки (1 — есть, 0 — нет):

import numpy as np

for column in ['website', 'support\_url', 'support\_email']:

df\_selected[column] = np.where(df\_selected[column] == 'not', 0, 1)

**Шаг 6. Кодирование жанров и категорий**

Для обработки столбцов genres и categories использовалось бинарное кодирование:

* Жанры:

df\_selected['genres'] = df\_selected['genres'].str.replace(r"[\[\]']", '', regex=True)

genres\_dummies = df\_selected['genres'].str.get\_dummies(sep=', ')

df\_selected = pd.concat([df\_selected, genres\_dummies], axis=1)

df\_selected.drop(columns=['genres'], inplace=True)

* Категории:

from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer()

categories\_encoded = mlb.fit\_transform(df\_selected['categories'])

categories\_df = pd.DataFrame(categories\_encoded, columns=mlb.classes\_)

df\_selected = pd.concat([df\_selected, categories\_df], axis=1)

df\_selected.drop(columns=['categories'], inplace=True)

**Шаг 7. Сохранение данных**

Обработанный датасет был сохранён в файл:

output\_path = '/content/drive/My Drive/df\_processed.csv'

df\_selected.to\_csv(output\_path, index=False)

print("Обработанные датасеты успешно сохранены:")

print(f"- {output\_path}")

**Итоги этапа:**

* Количество признаков увеличилось с 31 до 103.
* Преобразование данных позволило получить числовой формат, подходящий для последующего анализа.

### Этап 3. Избавление от сильно коррелирующих признаков

На данном этапе основной задачей было устранение признаков с высокой корреляцией, чтобы избежать избыточности данных и улучшить качество последующего анализа. Так как исходный датасет содержал большое количество признаков (103 после предыдущего этапа), анализ корреляций вручную оказался нерациональным, поэтому был настроен автоматизированный процесс устранения коррелирующих признаков.

**Шаг 1. Загрузка данных и первичный анализ**

Для начала были загружены данные из обработанного файла, созданного на предыдущем этапе, и выполнен первичный анализ структуры данных:

input\_path = "/content/drive/My Drive/df\_processed.csv"

df\_processed = pd.read\_csv(input\_path)

print("Первая информация о датасете:")

print(df\_processed.head())

print("\nОписание данных:")

print(df\_processed.info())

**Результаты:**

* Датасет содержит 87 115 записей и 103 признака.
* Все данные находятся в числовом формате (int64, float64, bool), что позволяет приступить к корреляционному анализу без дополнительных преобразований.

**Шаг 2. Расчёт корреляционной матрицы**

Для анализа была рассчитана корреляционная матрица с использованием только числовых признаков. Визуализация матрицы выполнена с помощью тепловой карты для наглядного представления:

numerical\_columns = df\_processed.select\_dtypes(include=['float64', 'int64', 'bool']).columns

numerical\_columns = numerical\_columns[1:] # Исключаем первый столбец (AppID)

correlation\_matrix = df\_processed[numerical\_columns].corr()

plt.figure(figsize=(15, 10))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', linewidths=0.5)

plt.title("Correlation Matrix of Features")

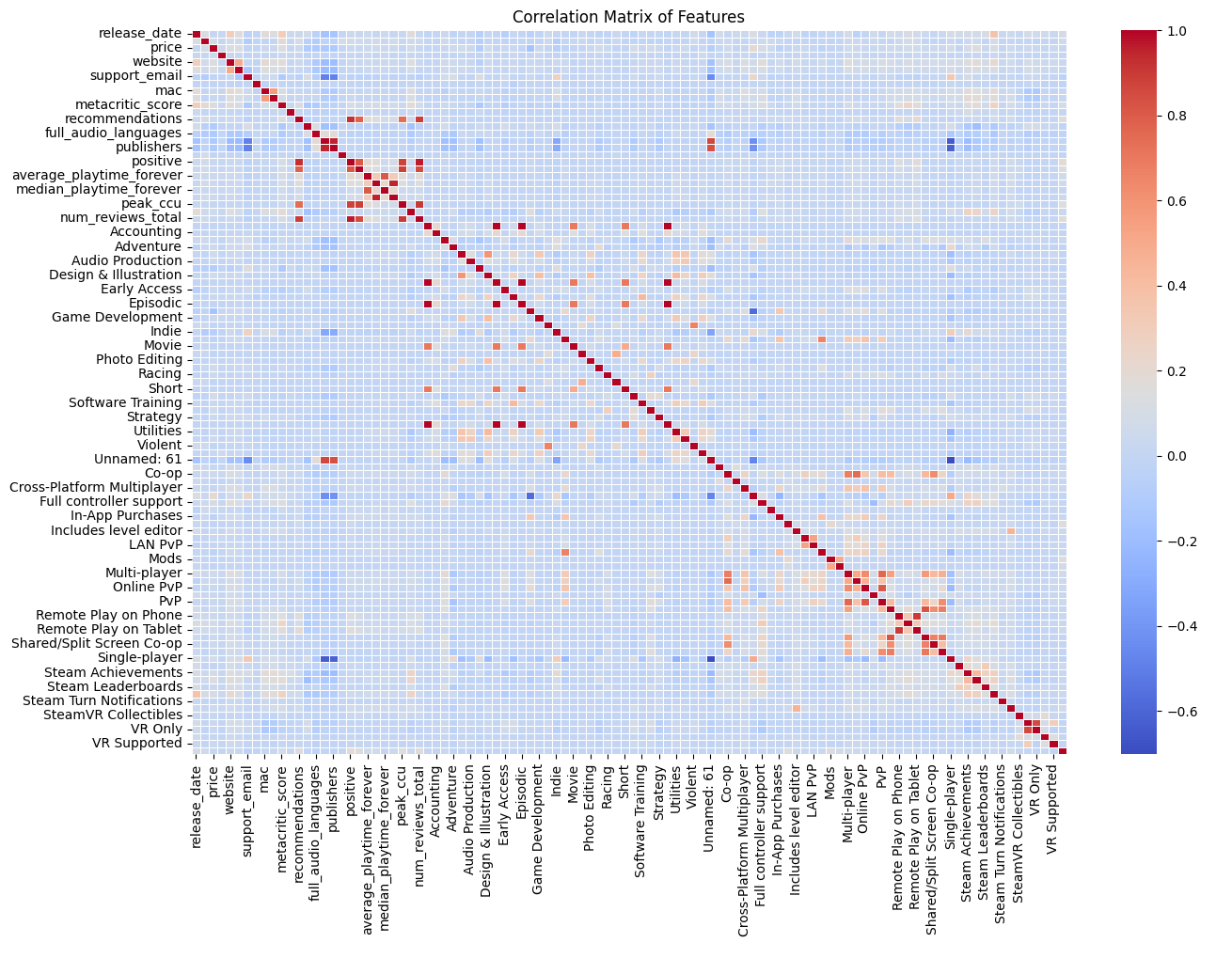
plt.show() ****

Рисунок 1. Тепловая карта до удаления слабо и сильно коррелирующих признаков

**Результаты:**

* На тепловой карте видно множество пар признаков с высокой корреляцией, что указывает на необходимость устранения избыточности.
* Анализировать и удалять признаки вручную оказалось слишком трудоёмким, поэтому был разработан автоматический процесс отбора признаков.

**Шаг 3. Автоматическое удаление коррелирующих признаков**

Для устранения коррелирующих признаков использовался следующий алгоритм:

1. Установлен порог корреляции — 0.5.
2. Удалены признаки с общей слабой корреляцией (сумма абсолютных корреляций по столбцу ниже порога).
3. Итеративно удалялись признаки из пар с высокой корреляцией (> 0.5), оставляя только один из них в датасете.

Алгоритм был реализован следующим образом:

threshold = 0.5

df\_reduced = df\_processed.copy()

df\_reduced.drop(columns=['AppID'], inplace=True)

uncorrelated\_features = correlation\_matrix.columns[(correlation\_matrix.abs().sum(axis=0) - 1) <= 0.5]

if not uncorrelated\_features.empty:

print(f"Удаляем фичи без корреляции: {list(uncorrelated\_features)}")

df\_reduced.drop(columns=uncorrelated\_features, inplace=True)

correlation\_matrix = df\_reduced.corr()

while True:

high\_corr = np.where(np.abs(correlation\_matrix) > threshold)

high\_corr\_pairs = [(correlation\_matrix.index[i], correlation\_matrix.columns[j])

for i, j in zip(\*high\_corr) if i != j]

if not high\_corr\_pairs:

break

feature1, feature2 = high\_corr\_pairs[0]

if feature2 in df\_reduced.columns:

df\_reduced.drop(columns=[feature2], inplace=True)

correlation\_matrix = df\_reduced.corr()

# Восстанавливаем столбец AppID

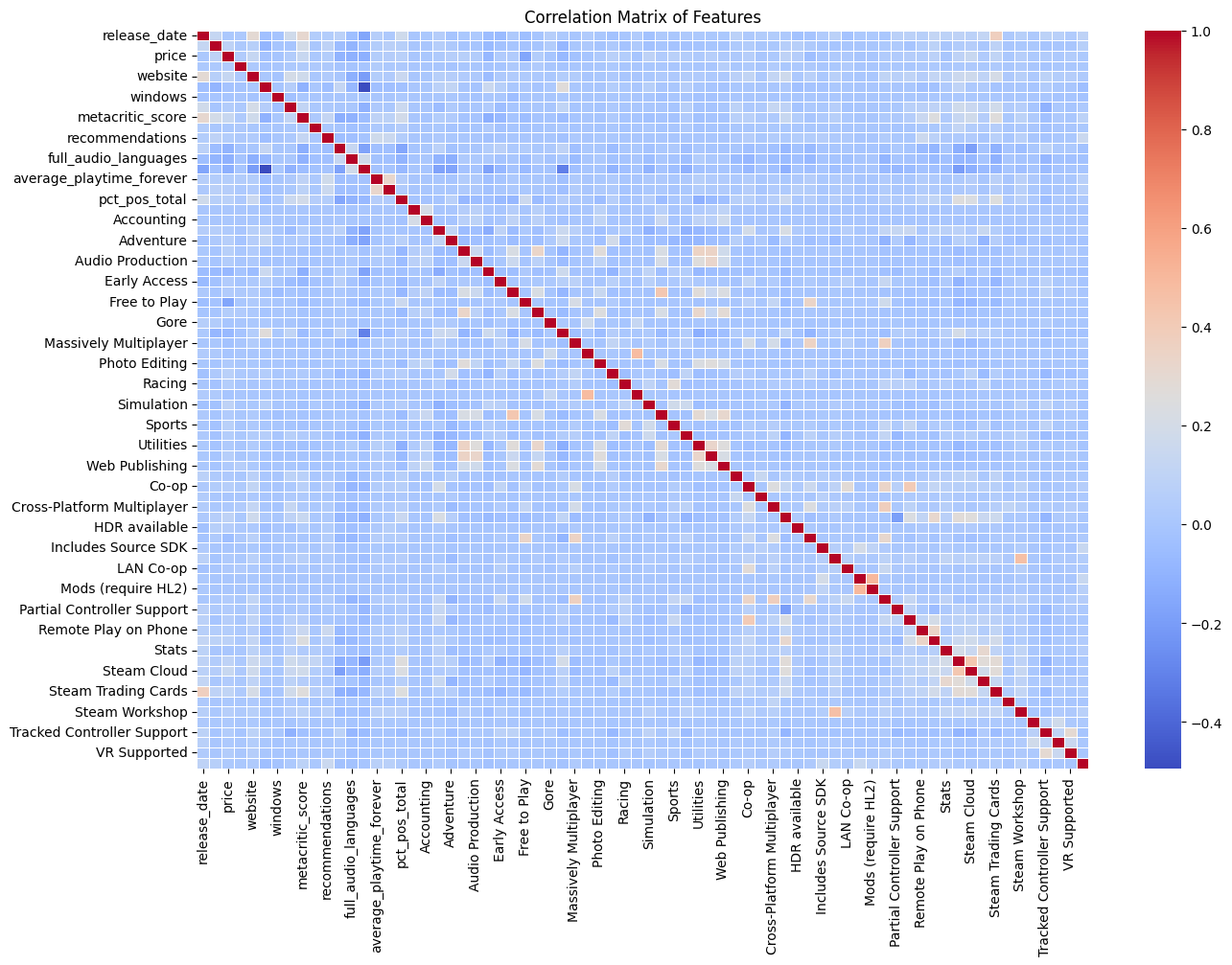


Рисунок 2. Тепловая карта после удаления слабо и сильно коррелирующих признаков

**Результаты:**

* Был удалён один слабо коррелирующий признак: user\_score.
* Общее количество признаков сократилось с 103 до 73, сохранив наиболее информативные данные.
* Итоговая корреляционная матрица имеет здоровую структуру, без избыточных корреляций.

Для визуализации новой корреляционной матрицы был выполнен повторный расчёт и построена тепловая карта:

plt.figure(figsize=(15, 10))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', linewidths=0.5)

plt.title("Correlation Matrix of Features (After Removing Highly Correlated Features)")

plt.show()

**Шаг 4. Сохранение результатов**

Обработанный датасет был сохранён в файл для последующего использования:

df\_reduced.to\_csv('/content/drive/My Drive/games\_cleaned\_filtered.csv', index=False)

print("Обновленный датасет сохранен в 'games\_cleaned\_filtered.csv'")

**Итоги этапа:**

* Успешно удалены сильно коррелирующие признаки, что позволило уменьшить размерность данных с 103 до 73 признаков.
* Настроенный автоматический алгоритм позволил сохранить релевантные данные без необходимости ручного анализа.
* Итоговый датасет подготовлен для следующего этапа анализа.

### Этап 4. Уменьшение размерности

На данном этапе была выполнена задача по уменьшению размерности данных с использованием метода анализа главных компонент (PCA). Этот метод был выбран благодаря его способности сохранять наиболее значимые признаки, минимизируя потерю информации. Основной целью этапа было сохранить индивидуальность данных при значительном уменьшении их размерности.

**Шаг 1. Подготовка данных и масштабирование**

Для начала были загружены данные из предыдущего этапа и выполнено их масштабирование, необходимое для корректной работы PCA. Масштабирование привело все признаки к единому диапазону значений:

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

file\_path = '/content/drive/My Drive/games\_cleaned\_filtered.csv'

df = pd.read\_csv(file\_path)

data\_for\_reduction = df\_reduced.drop(columns=['AppID']).values

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(data\_for\_reduction)

**Шаг 2. Применение PCA**

Метод PCA был использован для анализа дисперсии данных и определения минимального количества компонент, объясняющих 95% вариации. Это позволило значительно уменьшить количество признаков, сохранив их информационную значимость:

pca = PCA()

pca\_result = pca.fit\_transform(scaled\_data)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

cumulative\_variance = explained\_variance\_ratio.cumsum()

threshold = 0.95

num\_components = next(i for i, total in enumerate(cumulative\_variance) if total >= threshold) + 1

print(f"Общая объяснённая дисперсия (суммарно): {cumulative\_variance[-1]:.4f}")

print(f"Количество компонент для объяснения {threshold\*100}% дисперсии: {num\_components}")

pca = PCA(n\_components=num\_components)

reduced\_data = pca.fit\_transform(scaled\_data)

**Результаты:**

* Объяснённая дисперсия составила 100%.
* Для объяснения 95% дисперсии достаточно 65 компонент (изначально было 73 признака).
* Применение PCA позволило сохранить максимум информации при сокращении размерности.

**Шаг 3. Формирование итогового датасета**

На основе результатов PCA был создан новый датасет, содержащий 65 компонент, а также идентификатор AppID для связи с исходными данными:

reduced\_df = pd.DataFrame(reduced\_data, columns=[f'PC{i+1}' for i in range(num\_components)])

reduced\_df['AppID'] = df\_reduced['AppID'].values

print("Этап уменьшения размерности завершён.")

**Шаг 4. Проверка данных**

Для проверки корректности обработки был выполнен анализ структуры нового датасета:

print("Первая информация о датасете:")

print(reduced\_df.head())

print("\nОписание данных:")

print(reduced\_df.info())

**Результаты:**

* Новый датасет содержит 87 115 записей и 66 признаков (65 компонент + AppID).
* Все данные представлены в числовом формате.

**Шаг 5. Сохранение результатов**

Итоговый датасет был сохранён для дальнейшего использования:

reduced\_df.to\_csv('/content/drive/My Drive/games\_ym\_raz.csv', index=False)

print("Обновленный датасет сохранен в 'games\_ym\_raz.csv'")

**Итоги этапа:**

* Уменьшение размерности позволило оптимизировать структуру данных, сократив количество признаков с 73 до 65, при этом сохранив их информационную ценность.
* Выбранный метод PCA оказался наиболее подходящим для данной задачи благодаря своей способности сохранять основную вариацию данных.
* Дальнейшее уменьшение размерности было признано нецелесообразным, так как это могло бы привести к потере индивидуальности и качества данных.

### Этап 5. Исключённый этап: Кластеризация и удаление выбросов

На данном этапе были предприняты попытки провести кластеризацию данных с целью сегментации игр по схожим характеристикам. Однако результаты кластеризации оказались неудовлетворительными, несмотря на множество пробованных подходов. Одной из причин неудач стало наличие значительного числа выбросов в данных. Для снижения их влияния был реализован этап удаления выбросов, который применялся перед каждой попыткой кластеризации.

#### Удаление выбросов

Удаление выбросов выполнялось на основе интерквартильного размаха (IQR). Этот метод позволяет определить точки данных, находящиеся за пределами разумных границ, и удалить их. Для этого использовалась следующая функция:

# Функция для удаления выбросов с мягким порогом

def remove\_outliers\_soft(df, factor=3):

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

# Используем мягкий порог для удаления выбросов

return df[~((df < (Q1 - factor \* IQR)) | (df > (Q3 + factor \* IQR))).any(axis=1)]

Данный метод был применён к датасету, предварительно очищенному от столбца AppID, так как он не используется в анализе. Результаты удаления выбросов зависят от выбранного параметра factor, который определяет степень мягкости порога.

**Результаты удаления выбросов:**

1. **При параметре factor=3 (мягкий порог):**
   * Изначальное количество записей: 87 115
   * Итоговое количество записей: 66 817
   * Процент удалённых выбросов: 23.30%

Этот параметр позволил сохранить большую часть данных, удалив около четверти записей, что является компромиссным решением для минимизации влияния выбросов.

1. **При параметре factor=1.5 (жёсткий порог):**
   * Изначальное количество записей: 87 115
   * Итоговое количество записей: 44 100
   * Процент удалённых выбросов: 49.38%

Использование более жёсткого порога привело к удалению почти половины записей, что неприемлемо для задачи, так как значительное сокращение данных негативно влияет на качество кластеризации и снижает индивидуальность оставшихся данных.

**Вывод:** Был выбран параметр factor=3, так как он позволил сохранить как можно больше записей (66 817 из 87 115), удалив при этом значительное количество выбросов. Несмотря на это, процент удалённых выбросов остаётся высоким, что делает задачу кластеризации сложной даже после их устранения. Этот этап использовался в дальнейшем при каждой попытке кластеризации, как с выбросами, так и без них, чтобы оценить влияние выбросов на результаты кластеризации.

**Заключение:** Удаление выбросов с мягким порогом (factor=3) было признано оптимальным компромиссом между качеством данных и их объёмом. Однако даже при таком подходе кластеризация не дала удовлетворительных результатов, что будет подробно рассмотрено в следующих подэтапах.

#### Этап 5.1. Кластеризация методом K-means

Метод K-means был выбран для кластеризации данных из-за его популярности и эффективности при работе с большими наборами данных. Основная задача заключалась в определении оптимального количества кластеров и оценке качества кластеризации. Эксперименты проводились как с исходными данными, так и с предварительно обработанными (после удаления выбросов).

**Кластеризация без удаления выбросов**

На первом этапе кластеризация выполнялась на исходном датасете без удаления выбросов. Для выбора оптимального числа кластеров использовался метод локтя. Инерция — сумма расстояний от точек до центров кластеров — рассчитывалась для количества кластеров от 1 до 200:

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

inertias = []

for k in range(1, 201):

kmeans = KMeans(n\_clusters=k, random\_state=0).fit(df)

inertias.append(kmeans.inertia\_)

plt.plot(range(1, 201), inertias, 'bx-')

plt.xlabel('Количество кластеров')

plt.ylabel('Инерция')

plt.title('График локтя')

plt.show()

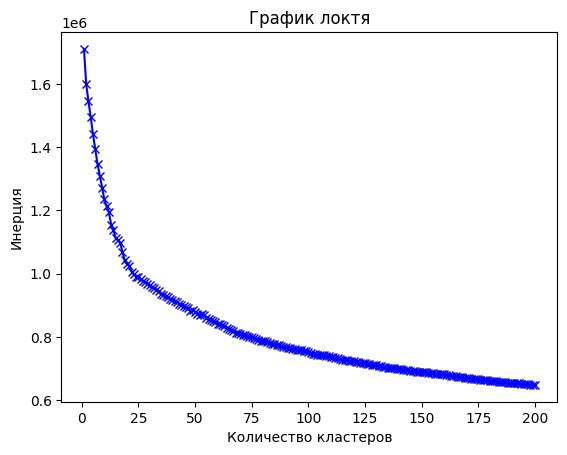


Рисунок 3. График локтя для метода K-means без удаления выбросов

**Результаты:**

* График локтя показал чётко выраженный излом на 25 кластерах, что было выбрано в качестве оптимального числа кластеров для следующего этапа анализа.

После выбора оптимального числа кластеров была выполнена кластеризация с использованием 25 кластеров, а также рассчитаны основные метрики для оценки качества:

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

kmeans = KMeans(n\_clusters=25, random\_state=0).fit(df)

print('Силуэтный коэффициент:', silhouette\_score(df, kmeans.labels\_))

print('Индекс Калинского-Харабаса:', calinski\_harabasz\_score(df, kmeans.labels\_))

print('Индекс Дэвиса-Болдина:', davies\_bouldin\_score(df, kmeans.labels\_))

**Результаты:**

* Силуэтный коэффициент: 0.0598 (почти равен нулю, что указывает на слабую выраженность кластеров).
* Индекс Калинского-Харабаса: 2033.36.
* Индекс Дэвиса-Болдина: 2.31.
* Распределение значений по кластерам показало сильную несбалансированность, где большая часть данных попадает в один или несколько кластеров.

**Кластеризация с удалёнными выбросами**

Далее кластеризация была выполнена на данных после удаления выбросов. Выбросы были удалены с использованием мягкого порога (factor=3). Как и ранее, для выбора оптимального числа кластеров использовался метод локтя:

inertias = []

for k in range(1, 201):

kmeans = KMeans(n\_clusters=k, random\_state=0).fit(df\_cleaned)

inertias.append(kmeans.inertia\_)

plt.plot(range(1, 201), inertias, 'bx-')

plt.xlabel('Количество кластеров')

plt.ylabel('Инерция')

plt.title('График локтя')

plt.show()

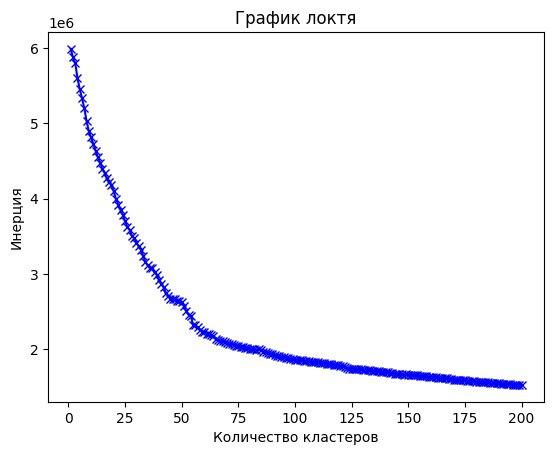


Рисунок 4. График локтя для метода K-means с удалением выбросов

**Результаты:**

* График локтя показал излом в диапазоне 44–54 кластеров. Для дальнейшего анализа были протестированы разные значения количества кластеров в этом диапазоне с шагом 2.

Для каждого количества кластеров были рассчитаны основные метрики:

for i in range(44, 55, 2):

print('При', i, 'кластерах')

kmeans = KMeans(n\_clusters=i, random\_state=0).fit(df\_cleaned)

print('Силуэтный коэффициент:', silhouette\_score(df\_cleaned, kmeans.labels\_))

print('Индекс Калинского-Харабаса:', calinski\_harabasz\_score(df\_cleaned, kmeans.labels\_))

print('Индекс Дэвиса-Болдина:', davies\_bouldin\_score(df\_cleaned, kmeans.labels\_))

**Результаты:**

* Лучшие результаты:
  + При 44 кластерах: силуэтный коэффициент — 0.0969.
  + Индекс Калинского-Харабаса: 2442.14.
  + Индекс Дэвиса-Болдина: 1.52.
* Для остальных значений (46–54 кластера) метрики оставались на аналогичном уровне, но силуэтный коэффициент всё равно был близок к нулю.

**Выводы:**

* Силуэтный коэффициент, являющийся основным индикатором качества кластеризации, оставался близким к нулю даже после удаления выбросов.
* Распределение данных по кластерам оставалось несбалансированным.
* Метод K-means не смог выделить качественные кластеры, что делает его непригодным для данной задачи.

#### Этап 5.2. Кластеризация методом иерархической кластеризации (Ward)

Иерархическая кластеризация, особенно метод Ward, была выбрана для сегментации данных, так как она позволяет учитывать внутрикластерные дисперсии при объединении кластеров. Данный метод позволяет визуально оценить количество кластеров с помощью дендрограммы.

**Кластеризация без удаления выбросов**

На первом этапе кластеризация выполнялась на исходных данных. Для визуального определения оптимального числа кластеров была построена дендрограмма:

from scipy.cluster.hierarchy import linkage, dendrogram

linkage\_matrix = linkage(df\_cleaned, method='ward')

plt.figure(figsize=(12, 8))

dendrogram(linkage\_matrix, truncate\_mode='lastp', p=30, leaf\_rotation=90., leaf\_font\_size=12.)

plt.title("Дендрограмма")

plt.xlabel("Индекс наблюдения или кластера")

plt.ylabel("Евклидово расстояние")

plt.show()

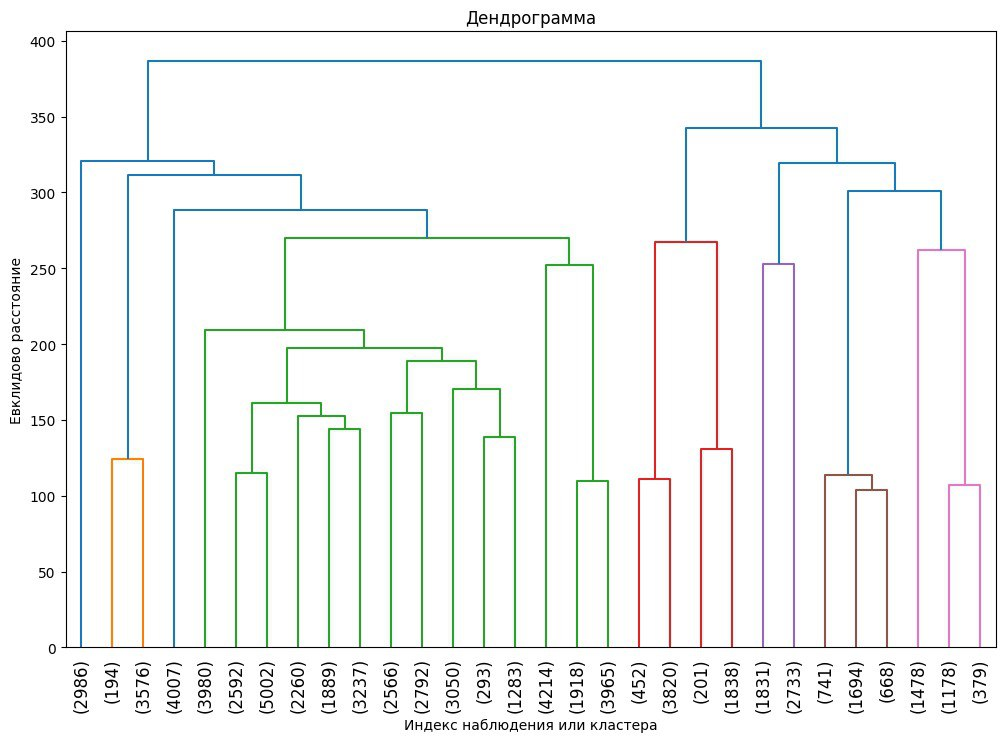


Рисунок 5. Дендограмма для метода иерархической кластеризации (Ward) без удаления выбросов

**Результаты:**

* Анализ дендрограммы показал, что оптимальное количество кластеров равно 6.

Далее была выполнена кластеризация с выбранным количеством кластеров, а также рассчитаны основные метрики:

from scipy.cluster.hierarchy import fcluster

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

n\_clusters = 6

labels = fcluster(linkage\_matrix, n\_clusters, criterion='maxclust')

silhouette\_avg = silhouette\_score(df\_cleaned, labels)

calinski\_harabasz = calinski\_harabasz\_score(df\_cleaned, labels)

davies\_bouldin = davies\_bouldin\_score(df\_cleaned, labels)

print(f"Silhouette Score: {silhouette\_avg:.3f}")

print(f"Calinski-Harabasz Index: {calinski\_harabasz:.3f}")

print(f"Davies-Bouldin Index: {davies\_bouldin:.3f}")

**Результаты:**

* Silhouette Score: 0.118 (низкий, что указывает на слабую структуру кластеров).
* Calinski-Harabasz Index: 2669.256.
* Davies-Bouldin Index: 2.619.
* Распределение по кластерам оставалось несбалансированным, что негативно сказалось на интерпретации результатов.

**Кластеризация с удалёнными выбросами**

Далее кластеризация была выполнена на данных после удаления выбросов. Как и ранее, для определения оптимального числа кластеров была построена дендрограмма:

linkage\_matrix = linkage(df\_cleaned, method='ward')

plt.figure(figsize=(12, 8))

dendrogram(linkage\_matrix, truncate\_mode='lastp', p=30, leaf\_rotation=90., leaf\_font\_size=12.)

plt.title("Дендрограмма")

plt.xlabel("Индекс наблюдения или кластера")

plt.ylabel("Евклидово расстояние")

plt.show()

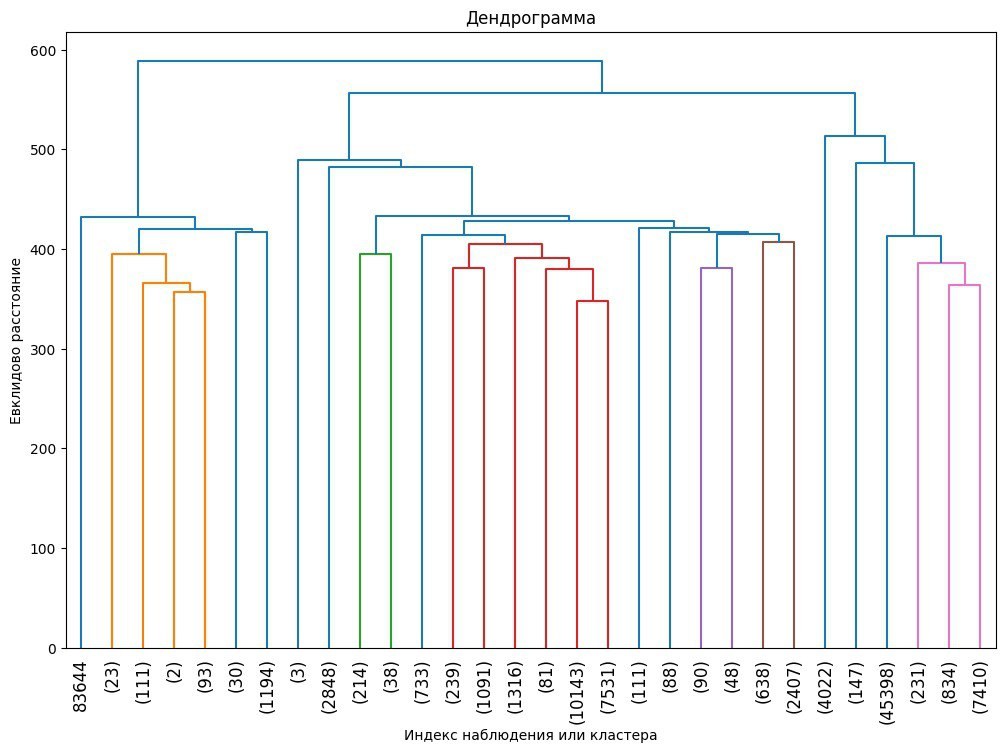


Рисунок 6. Дендограмма для метода иерархической кластеризации (Ward) с удалением выбросов

**Результаты:**

* Дендрограмма показала, что оптимальное количество кластеров составляет 4.

Для каждого количества кластеров была рассчитана метрика качества:

n\_clusters = 4

labels = fcluster(linkage\_matrix, n\_clusters, criterion='maxclust')

silhouette\_avg = silhouette\_score(df\_cleaned, labels)

calinski\_harabasz = calinski\_harabasz\_score(df\_cleaned, labels)

davies\_bouldin = davies\_bouldin\_score(df\_cleaned, labels)

print(f"Silhouette Score: {silhouette\_avg:.3f}")

print(f"Calinski-Harabasz Index: {calinski\_harabasz:.3f}")

print(f"Davies-Bouldin Index: {davies\_bouldin:.3f}")

**Результаты:**

* Silhouette Score: 0.0938 (указание на слабую сегментацию кластеров).
* Calinski-Harabasz Index: 2421.0.
* Davies-Bouldin Index: 2.4.
* Распределение по кластерам также оставалось несбалансированным.

**Выводы:**

* Кластеризация методом Ward показала немного лучшие результаты по сравнению с K-means, однако силуэтный коэффициент остаётся низким, что указывает на слабую структуру кластеров.
* Удаление выбросов улучшило метрики, но качество кластеризации остаётся неудовлетворительным.
* Данный метод также оказался малоэффективным для данной задачи.

#### Этап 5.3. Кластеризация метод DBSCAN

Метод DBSCAN был выбран благодаря своей устойчивости к выбросам и способности выявлять кластеры произвольной формы. Основная задача заключалась в определении оптимального значения параметра eps (максимального расстояния между двумя точками в одном кластере). Для этого был построен график локтя, показывающий зависимости расстояния до ближайших соседей от упорядоченных точек данных. Анализ проводился как на исходных данных, так и на данных с удалёнными выбросами.

**Кластеризация без удаления выбросов**

Для определения гиперпараметра eps сначала использовалось значение k=5 для анализа ближайших соседей:

from sklearn.neighbors import NearestNeighbors

k = 5

neighbors = NearestNeighbors(n\_neighbors=k)

neighbors\_fit = neighbors.fit(df\_cleaned)

distances, indices = neighbors\_fit.kneighbors(df\_cleaned)

distances = np.sort(distances[:, k-1])

plt.figure(figsize=(10, 6))

plt.plot(distances)

plt.title("График локтя для определения eps в DBSCAN")

plt.xlabel("Точки данных (упорядоченные)")

plt.ylabel(f"Расстояние до {k}-го ближайшего соседа")

plt.grid()

plt.show()

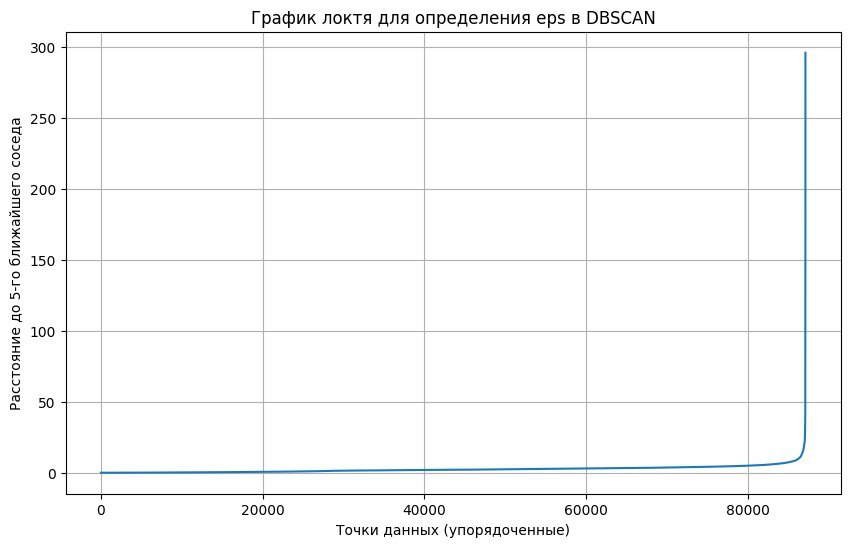


Рисунок 7. График локтя для метода DBSCAN без удаления выбросов

**Результаты:**

* График локтя оказался слишком резким, без чётко выраженного изгиба, что указывает на сложность определения оптимального значения eps.

Была предпринята попытка повторного анализа с увеличением количества соседей до k=10, однако результат остался аналогичным — график оставался резким, без чётко выраженного перелома.

**Кластеризация с удалёнными выбросами**

После удаления выбросов метод DBSCAN был протестирован повторно. Однако график локтя для значений k=5 и k=10 оставался неудовлетворительным, без чётко выраженного изгиба, что затрудняло выбор оптимального значения eps. Из-за этого построение модели DBSCAN не представлялось возможным.

**Микровывод:**  
На данном этапе была предпринята попытка кластеризации методом DBSCAN, который обладает высокой устойчивостью к выбросам и хорошо подходит для данных с произвольной формой кластеров. Однако, несмотря на различные настройки параметров, таких как количество ближайших соседей (k=5 и k=10), и удаление выбросов, график локтя не продемонстрировал чёткого изгиба, необходимого для определения оптимального значения параметра eps. Это затруднило применение метода DBSCAN, и построение модели оказалось невозможным.

#### Этап 5.4. Общие выводы по этапу кластеризации:

1. **Метрики качества:**
   * Все протестированные методы показали низкий силуэтный коэффициент, что указывает на слабую сегментацию кластеров.
   * Индексы Калинского-Харабаса и Дэвиса-Болдина также не продемонстрировали значимых улучшений.
2. **Удаление выбросов:**
   * Удаление выбросов улучшило метрики в некоторых случаях, однако не привело к значительному повышению качества кластеризации.
3. **Графики локтя:**
   * Для метода DBSCAN графики локтя были слишком резкими, без чётко выраженных изгибов, что подтверждает отсутствие явных плотных кластеров в данных.
4. **Итоговое решение:**
   * Этап кластеризации был исключён из проекта, так как ни один из методов не продемонстрировал удовлетворительных результатов.
   * Дальнейшие эксперименты с кластеризацией были признаны нецелесообразными из-за высокого уровня шума в данных и сложности их сегментации.
   * **Не удалять выбросы**, поскольку их значительное количество объясняется природой реальных данных и отражает естественную изменчивость наблюдений. Исключение выбросов привело бы к потере значительной части данных, что нежелательно для анализа.

### Этап 6. Построение модели рекомендаций

На данном этапе была построена модель для рекомендаций игр на основе их характеристик. Для реализации использовался алгоритм ближайших соседей, который показал свою эффективность в задачах поиска схожих объектов по набору признаков.

**Выбор метода**

Метод ближайших соседей (Nearest Neighbors) был выбран по следующим причинам:

* **Простота и интерпретируемость**: Алгоритм не требует обучения, что делает его лёгким для понимания и реализации.
* **Эффективность для рекомендательных систем**: Метод позволяет найти объекты, наиболее близкие по характеристикам к заданному объекту, что идеально подходит для рекомендаций.
* **Гибкость**: Алгоритм поддерживает различные метрики расстояний, что даёт возможность адаптироваться под специфику данных.

Для поиска ближайших соседей использовалась евклидова метрика, которая хорошо подходит для данных, прошедших масштабирование.

**Реализация модели**

1. **Загрузка данных:** На вход модели подавались данные из обработанного датасета, где целевая переменная — AppID — использовалась для идентификации игр, а остальные признаки — для поиска схожести:

file\_path = '/content/drive/My Drive/games\_ym\_raz.csv'

games = pd.read\_csv(file\_path)

features = games.drop(columns=['AppID']) # 'AppID' — целевая переменная

target = games['AppID']

1. **Создание модели:** Для реализации модели использовался алгоритм Nearest Neighbors с 6 соседями:

from sklearn.neighbors import NearestNeighbors

knn\_model = NearestNeighbors(n\_neighbors=6, metric='euclidean')

knn\_model.fit(features)

1. **Функция для получения рекомендаций:** Была реализована функция, которая принимает на вход ID игры, находит ближайшие к ней игры и возвращает список их идентификаторов:

def recommend\_games\_v1(game\_id, data, model, n\_recommendations=10):

# Получаем фичи игры по ID

game\_features = data[data['AppID'] == game\_id].drop(columns=['AppID'])

if game\_features.empty:

return f"Игра с ID {game\_id} не найдена!"

# Находим соседей

distances, indices = model.kneighbors(game\_features, n\_neighbors=n\_recommendations+1)

# Возвращаем рекомендованные ID игр

recommended\_ids = data.iloc[indices[0]]['AppID'].values[1:] # Пропускаем саму игру

return recommended\_ids

1. **Тестирование:** Модель была протестирована на примере игры с AppID = 730 (игра Counter-Strike: Global Offensive). Результаты показали корректную работу алгоритма:

game\_id\_to\_recommend = 730

recommendations = recommend\_games\_v1(game\_id\_to\_recommend, games, knn\_model)

print(f"Рекомендованные игры для ID {game\_id\_to\_recommend}: {recommendations}")

**Результаты:**

Рекомендованные игры для ID 730: [ 578080 271590 488822 105600 252490 4000 292030 431960 1091500 553850]

**Заключение**

На данном этапе была построена базовая модель рекомендаций, которая успешно прошла проверку на корректность работы. Следует отметить, что тестирование было проведено исключительно для подтверждения работоспособности модели, а не для оценки её качества. Выбор метода ближайших соседей был оправдан с точки зрения простоты и эффективности для поставленной задачи. В дальнейшем возможны улучшения модели, такие как добавление более сложных метрик или оптимизация гиперпараметров.

### Этап 7. Разработка интерфейса и условная оценка модели

На заключительном этапе проекта был разработан пользовательский интерфейс для взаимодействия с рекомендательной системой. Это позволяет пользователю вводить название игры, получать рекомендации по похожим играм и видеть результаты в удобном формате. Интерфейс был реализован с использованием библиотеки CustomTkinter, которая предоставляет современный и адаптивный дизайн для приложений.

**Шаг 1. Подготовка данных для интерфейса**

Для отображения названий игр вместе с их идентификаторами был создан дополнительный датасет, содержащий только два столбца: AppID и name. Это позволяет связывать числовые идентификаторы с их текстовыми названиями при работе с рекомендациями:

import pandas as pd

from rapidfuzz import process

# Загрузка исходного датасета с полным набором данных

df = pd.read\_csv('/content/drive/My Drive/games\_cleaned.csv')

# Создание нового датасета с двумя ключевыми столбцами: AppID и name

new\_df = df[['AppID', 'name']]

# Сохранение нового датасета для дальнейшего использования

new\_df.to\_csv('/content/drive/My Drive/id-name.csv', index=False)

**Шаг 2. Создание графического интерфейса**

Графический интерфейс приложения был реализован с использованием библиотеки CustomTkinter. Основные функции интерфейса:

* Ввод названия игры.
* Нечёткий поиск названия игры в датасете (используется библиотека RapidFuzz для обработки пользовательского ввода).
* Отображение результатов рекомендаций с названиями игр.

Программа была разделена на несколько функциональных частей:

1. **Загрузка данных и модели:**

Данные и модель Nearest Neighbors были загружены заранее, чтобы ускорить обработку пользовательских запросов:

import pandas as pd

import joblib

dataset = pd.read\_csv('id-name.csv')

dataset['name\_normalized'] = dataset['name'].str.lower().str.strip()

file\_path = 'games\_ym\_raz.csv'

games = pd.read\_csv(file\_path)

knn\_model = joblib.load('nearest\_neighbors\_model.pkl')

features = games.drop(columns=['AppID'])

1. **Поиск игры и генерация рекомендаций:**

Функции для поиска игры по названию и генерации рекомендаций:

def get\_name\_by\_appid(appid):

row = dataset.loc[dataset['AppID'] == appid]

if not row.empty:

return row.iloc[0]['name']

return None

def get\_appid\_by\_name(name):

name\_normalized = name.lower().strip()

match = process.extractOne(name\_normalized, dataset['name\_normalized'])

if match and match[1] > 80:

row = dataset.loc[dataset['name\_normalized'] == match[0]]

if not row.empty:

return row.iloc[0]['AppID']

return None

def recommend\_games\_v1(game\_id, data, model, n\_recommendations=10):

game\_features = data[data['AppID'] == game\_id].drop(columns=['AppID'])

if game\_features.empty:

return f"Игра с ID {game\_id} не найдена!"

distances, indices = model.kneighbors(game\_features, n\_neighbors=n\_recommendations+1)

recommended\_ids = data.iloc[indices[0]]['AppID'].values[1:]

return recommended\_ids

1. **Создание графического интерфейса:**

Основные элементы интерфейса включают:

* + Поле для ввода названия игры.
  + Кнопку для запуска процесса поиска.
  + Поле для отображения рекомендаций.

import customtkinter as ctk

import tkinter as tk

root = ctk.CTk()

root.title("Рекомендация игр")

root.geometry("550x380")

root.resizable(False, False)

def process\_input():

user\_input = input\_entry.get()

if user\_input.strip():

game\_id\_to\_recommend = get\_appid\_by\_name(user\_input)

recommendations = recommend\_games\_v1(game\_id\_to\_recommend, games, knn\_model)

results = [get\_name\_by\_appid(i) for i in recommendations]

input\_entry.delete(0, tk.END)

processed\_label.configure(text=f"Обработанная строка: {get\_name\_by\_appid(game\_id\_to\_recommend)}")

results\_text.configure(state="normal")

results\_text.delete(1.0, "end")

results\_text.insert("end", "\n".join(results))

results\_text.configure(state="disabled")

input\_label = ctk.CTkLabel(root, text="Введите игру:", font=("Comic Sans MS", 14))

input\_entry = ctk.CTkEntry(root, placeholder\_text="Введите название игры", width=250)

process\_button = ctk.CTkButton(root, text="Найти!", command=process\_input, width=100)

processed\_label = ctk.CTkLabel(root, text="Рекомендованные игры:", font=("Comic Sans MS", 14))

results\_label = ctk.CTkLabel(root, text="Топ 10 игр:", font=("Comic Sans MS", 14))

results\_text = ctk.CTkTextbox(root, width=500, height=190, state="disabled", font=("Comic Sans MS", 12))

# Расположение элементов на экране

input\_label.grid(row=0, column=0, padx=10, pady=15, sticky="w")

input\_entry.grid(row=0, column=1, padx=10, pady=15, sticky="w")

process\_button.grid(row=0, column=2, padx=10, pady=15, sticky="w")

processed\_label.grid(row=1, column=0, columnspan=3, padx=10, pady=10, sticky="w")

results\_label.grid(row=2, column=0, columnspan=3, padx=10, pady=10, sticky="w")

results\_text.grid(row=3, column=0, columnspan=3, padx=10, pady=15)

root.mainloop()

**Условная оценка модели**

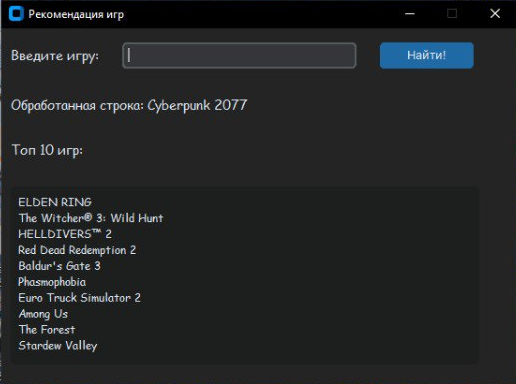
Тест 1  


Рисунок 8. Тестовый запуск с запросом «cyberpunk 2077»

Тест 2

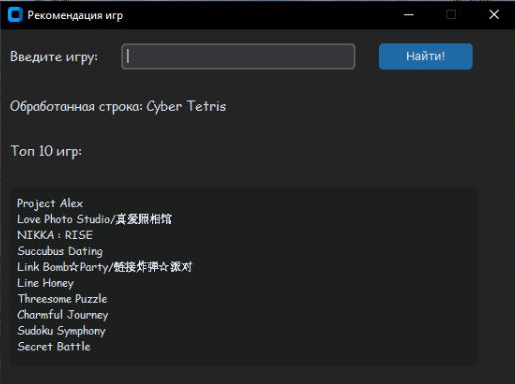


Рисунок 9. Тестовый запуск с запросом «tetris»

Тест 3

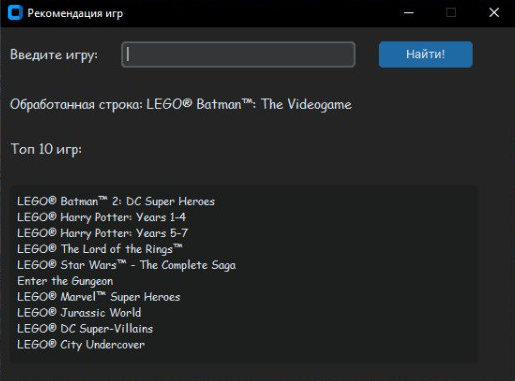


Рисунок 10. Тестовый запуск с запросом «lego batman the videogame»

Тест 4

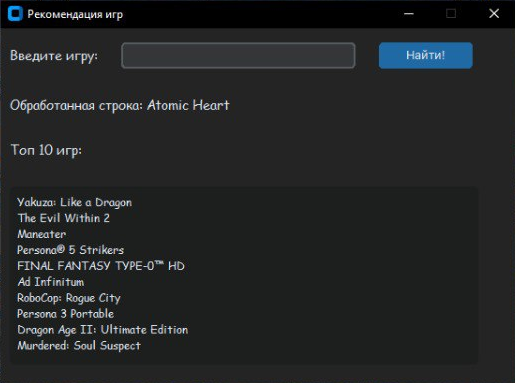


Рисунок 11. Тестовый запуск с запросом «atomic heart»

Тест 5



Рисунок 12. Тестовый запуск с запросом «mafia II»

Результаты тестирования модели проводились на выборке из пяти игр с использованием пользовательского интерфейса. Полученные рекомендации были проверены на релевантность с точки зрения человека, знакомого с игровой индустрией. Основные выводы по результатам тестов:

1. **Качество рекомендаций:**
   * Для игры Cyberpunk 2077 были предложены такие известные проекты, как ELDEN RING, The Witcher® 3: Wild Hunt и Red Dead Redemption 2. Все они имеют схожие жанровые черты и соответствуют высокому уровню проработки мира, что демонстрирует успешность подхода.
   * Для игры LEGO Batman: The Videogame результаты включали преимущественно другие проекты серии LEGO, что подчёркивает способность модели находить точные тематические совпадения.
   * Рекомендации для Atomic Heart и Mafia II подтвердили, что система успешно подбирает игры со схожими стилистическими или сюжетными характеристиками. Это подчёркивает высокий уровень релевантности выдачи.
2. **Недостатки:**
   * В случае игры Tetris рекомендации оказались менее релевантными, включив проекты, которые сложно назвать схожими. Это указывает на определённые ограничения алгоритма в обработке игр с уникальной игровой механикой или жанром. Тем не менее, такая ситуация возникла лишь с одним тестовым примером.
3. **Общий уровень работы модели:**
   * Для большинства протестированных игр рекомендации были логичными и полезными. Это говорит о том, что модель способна решать основную задачу с высоким уровнем точности. Недостатки в выдаче рекомендаций для отдельных игр незначительно снижают общий уровень восприятия модели.

**Субъективность оценки:**  
Оценка качества рекомендаций в текущем виде частично субъективна и основана на опыте проверяющего. Для повышения объективности в будущем могут быть использованы:

* **Опрос пользователей:** Сбор обратной связи от целевой аудитории.
* **Метрики соответствия:** Использование тестового набора с известными предпочтениями пользователей.
* **Кросс-валидация:** Тестирование на заранее размеченных данных.

**Заключение по условной оценке:**  
Результаты тестирования показывают, что разработанная модель в большинстве случаев выдаёт качественные и релевантные рекомендации. Исключения, такие как результаты для Tetris, являются скорее редкостью, чем правилом. Это свидетельствует о высокой эффективности подхода и значительном потенциале для дальнейшего совершенствования. Субъективная оценка качества работы модели приближается к 4,2 из 5, что указывает на успешное выполнение поставленной задачи и достижение цели проекта.

**Микровывод:** На данном этапе был успешно разработан пользовательский интерфейс для рекомендательной системы. Программа корректно принимает ввод пользователя, предоставляет список рекомендованных игр и отображает их в удобном формате. Этап реализован в соответствии с запланированными требованиями.

# Заключение

# В ходе данного проекта была проведена работа по анализу факторов, влияющих на сходство характеристик игр, с целью выявления наиболее схожих проектов на основе представленных данных. Работа охватывала полный цикл обработки данных: от предобработки и анализа структуры до построения модели и тестирования пользовательского интерфейса.

# Основные этапы проекта включали:

# Подготовку данных, включая удаление пропущенных значений, приведение признаков к числовому виду и уменьшение размерности.

# Построение модели анализа сходства, основанной на алгоритме ближайших соседей.

# Создание пользовательского интерфейса для удобного взаимодействия с системой.

# Во время реализации были использованы разнообразные методы обработки и анализа данных, включая кластеризацию, которая, несмотря на усилия, не дала удовлетворительных результатов и была исключена из финальной версии проекта. Тем не менее, данный опыт позволил глубже понять особенности данных и повысить качество аналитической модели.

# Результаты показывают, что разработанная система способна находить игры с высокой степенью сходства на основе характеристик. Интерфейс, созданный с использованием CustomTkinter, обеспечивает удобный доступ к функционалу системы, делая её интуитивно понятной для конечного пользователя.

# Проект может быть расширен и улучшен в следующих направлениях:

# Оптимизация гиперпараметров модели для повышения точности анализа.

# Интеграция дополнительных источников данных, таких как пользовательские оценки и отзывы.

# Расширение модели для использования гибридных методов анализа, что повысит точность и персонализацию.

# Итог: Проект выполнен успешно. Разработанная система анализа характеристик игр представляет собой работоспособное решение, которое может быть использовано как основа для дальнейших разработок в области изучения игровых данных.

# Список использованных источников и литературы

1. Герасимов, А. Н. "Машинное обучение и анализ данных." Москва: Издательство "Бином", 2021.
2. Pedregosa, F., et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 2011.
3. Aggarwal, C. C. "Recommender Systems: The Textbook." Springer, 2016.
4. Hastie, T., Tibshirani, R., Friedman, J. "The Elements of Statistical Learning." Springer, 2009.
5. McKinney, W. "Python for Data Analysis." O'Reilly Media, 2017.
6. Гусев, А. В. "Основы кластерного анализа." Наука и техника, 2018.
7. Официальная документация библиотеки Scikit-learn: <https://scikit-learn.org>
8. Официальная документация библиотеки CustomTkinter: <https://github.com/TomSchimansky/CustomTkinter>
9. Официальная документация библиотеки Pandas: <https://pandas.pydata.org>
10. Официальная документация библиотеки RapidFuzz: <https://maxbachmann.github.io/RapidFuzz>

# Приложение

Код для подпункта «Подготовка к анализу»:

# Импорт необходимых библиотек для анализа и визуализации данных

import pandas as pd           # Работа с таблицами и обработка данных

import matplotlib.pyplot as plt  # Построение графиков

import seaborn as sns         # Улучшенная визуализация графиков

import numpy as np            # Работа с массивами и числовыми операциями

# Подключение Google Drive для загрузки данных

from google.colab import drive

drive.mount('/content/drive')

# Настройка отображения всех столбцов в выводе DataFrame

pd.set\_option('display.max\_columns', None)

# Функция для обзора структуры и содержимого датасета

def overview(df, name):

    """

    Выводит информацию о структуре, описательные статистики

    и первые строки данных для заданного DataFrame.

    Параметры:

        df (pd.DataFrame): Анализируемый датасет

        name (str): Имя датасета для заголовка

    """

    print(f"\n--- Обзор датасета: {name} ---")

    print(df.info())       # Общая информация о датасете

    print(df.describe())   # Статистическое описание числовых столбцов

    print(f"Примеры данных:\n{df.head()}\n")  # Первые строки таблицы

Код для подпункта «Этап 1. Исключение отсутствующих значений»

# Шаг 1: Загрузка датасета для анализа

games\_cleaned = pd.read\_csv('/content/drive/My Drive/games\_may2024\_full.csv')

# Шаг 2: Обзор структуры и содержимого загруженного датасета

overview(games\_cleaned, 'games')

# Шаг 3: Проверка наличия пропущенных значений в столбцах

print("\nПропущенные значения в games.csv:")

print(games\_cleaned.isnull().sum())

# Шаг 1: Удаление строк с отсутствующими значениями в поле 'name'

games\_cleaned = games\_cleaned.dropna(subset=['name']).copy()

# Шаг 2: Удаление дублирующихся значений в столбце 'name'

games\_cleaned.drop\_duplicates(subset=['name'], inplace=True)

# Шаг 3: Заполнение пропусков значениями по умолчанию для текстовых полей

games\_cleaned['short\_description'] = games\_cleaned['short\_description'].fillna("Description not available")

games\_cleaned['about\_the\_game'] = games\_cleaned['about\_the\_game'].fillna("Information not available")

games\_cleaned['website'] = games\_cleaned['website'].fillna("not")

games\_cleaned['support\_url'] = games\_cleaned['support\_url'].fillna("not")

games\_cleaned['support\_email'] = games\_cleaned['support\_email'].fillna("not")

games\_cleaned['detailed\_description'] = games\_cleaned['detailed\_description'].fillna("Detailed description not available")

# Шаг 4: Удаление нерелевантных столбцов

columns\_to\_drop = [

    'header\_image', 'detailed\_description', 'about\_the\_game', 'short\_description',

    'reviews', 'notes', 'metacritic\_url', 'estimated\_owners',

    'num\_reviews\_recent', 'pct\_pos\_recent', 'tags', 'screenshots',

    'movies', 'packages', 'score\_rank'

]

games\_cleaned = games\_cleaned.drop(columns=columns\_to\_drop)

# Шаг 5: Проверка данных после обработки

print("Пропуски в games\_cleaned после финальной обработки:")

print(games\_cleaned.isnull().sum())

overview(games\_cleaned, 'games\_cleaned')

# Шаг 6: Вывод итоговой размерности обработанного датасета

print("\nИтоговая размерность games\_cleaned:", games\_cleaned.shape)

# Шаг 7: Сохранение обработанного датасета в файл

output\_path = '/content/drive/My Drive/games\_cleaned.csv'

games\_cleaned.to\_csv(output\_path, index=False)

# Подтверждение успешного сохранения

print("Обработанный датасет успешно сохранён:")

print(f"- {output\_path}")

Код для подпункта «Этап 2. Приведение всех признаков к числовому виду»

# Шаг 1: Загрузка обработанного датасета

df\_selected = pd.read\_csv("/content/drive/My Drive/games\_cleaned.csv")

# Шаг 2: Первичная проверка данных

print("Первая информация о датасете:")

print(df\_selected.head())  # Вывод первых строк для ознакомления

print("\nОписание структуры данных:")

print(df\_selected.info())  # Вывод информации о типах данных и пропущенных значениях

import ast

# Шаг 1: Функция для преобразования строк в списки

def convert\_str\_to\_list(df, column):

    """

    Преобразует строковые представления списков в объекты типа list.

    Args:

        df (pd.DataFrame): Датасет для преобразования.

        column (str): Название столбца для обработки.

    Returns:

        pd.DataFrame: Обновленный датасет.

    """

    df[column] = df[column].apply(lambda x: ast.literal\_eval(x) if isinstance(x, str) else x)

    return df

# Шаг 2: Преобразование строковых данных в списки для указанных столбцов

list\_columns = ['supported\_languages', 'full\_audio\_languages', 'developers', 'publishers']

for column in list\_columns:

    df\_selected = convert\_str\_to\_list(df\_selected, column)

# Шаг 3: Подсчёт уникальных значений в каждом из преобразованных столбцов

for column in list\_columns:

    unique\_values = df\_selected[column].apply(lambda x: len(set(x)) if isinstance(x, list) else 0)

    print(f"\nУникальные значения в столбце '{column}': {unique\_values.sum()}")

    print(f"Примеры уникальных значений в '{column}':")

    print(unique\_values.head())

from collections import Counter

# Шаг 1: Функция для подсчёта частоты значений

def count\_encode(column):

    """

    Преобразует списки в строковые представления, подсчитывает частоту их появления и заменяет на частотное значение.

    Args:

        column (pd.Series): Столбец с данными, содержащий списки или строки.

    Returns:

        pd.Series: Обновлённый столбец с частотными значениями.

    """

    # Преобразуем списки в строку, разделённую запятыми

    column\_as\_str = column.apply(lambda x: ','.join(x) if isinstance(x, list) else str(x))

    # Подсчитываем частоту каждого уникального значения

    counts = Counter(column\_as\_str)

    # Заменяем значения на их частотные представления

    return column\_as\_str.map(counts)

# Шаг 2: Применение кодировки к указанным столбцам

columns\_to\_encode = ['supported\_languages', 'full\_audio\_languages', 'developers', 'publishers']

for column in columns\_to\_encode:

    df\_selected[column] = count\_encode(df\_selected[column])

from sklearn.preprocessing import MultiLabelBinarizer

import numpy as np

# Шаг 3: Функция для вычисления разницы между текущей датой и датой релиза

def del\_data(date\_str):

    """

    Преобразует дату релиза игры в количество дней, прошедших с этой даты до 2024 года.

    Args:

        date\_str (str): Дата в формате 'YYYY-MM-DD'.

    Returns:

        int: Количество дней.

    """

    return (2024 - int(date\_str[:4])) \* 365 + (12 - int(date\_str[5:7])) \* 30 + int(date\_str[8:10])

# Шаг 4: Обработка столбцов с датами и ссылками

df\_processed = df\_selected.copy()

df\_processed['release\_date'] = df\_processed['release\_date'].apply(del\_data)

# Преобразование наличия информации о веб-сайте, поддержке в бинарные признаки

for column in ['website', 'support\_url', 'support\_email']:

    df\_processed[column] = np.where(df\_processed[column] == 'not', 0, 1)

# Шаг 5: Обработка столбца 'genres' (жанры)

# Убираем лишние символы из строк и разделяем жанры

df\_processed['genres'] = df\_processed['genres'].str.replace(r"[\[\]']", '', regex=True)

genres\_dummies = df\_processed['genres'].str.get\_dummies(sep=', ')

# Объединяем закодированные жанры с основным датафреймом и удаляем оригинальный столбец

df\_processed = pd.concat([df\_processed, genres\_dummies], axis=1)

df\_processed.drop(columns=['genres', 'name'], inplace=True)

# Шаг 6: Обработка столбца 'categories' (категории)

# Убираем лишние символы и преобразуем строки в списки

df\_processed['categories'] = df\_processed['categories'].str.replace(r"[\[\]']", '', regex=True)

df\_processed['categories'] = df\_processed['categories'].apply(lambda x: x.split(', ') if isinstance(x, str) else [])

# Бинарное кодирование категорий с использованием MultiLabelBinarizer

mlb = MultiLabelBinarizer()

categories\_encoded = mlb.fit\_transform(df\_processed['categories'])

# Создание датафрейма с закодированными категориями

categories\_df = pd.DataFrame(categories\_encoded, columns=mlb.classes\_)

# Добавляем бинарные категории в основной датафрейм и удаляем оригинальный столбец

df\_processed = pd.concat([df\_processed, categories\_df], axis=1)

df\_processed.drop(columns=['categories'], inplace=True)

# Шаг 7: Просмотр результата обработки

print("Данные после обработки жанров и категорий:")

print(df\_processed.head())

print("\nОписание данных:")

print(df\_processed.info())

# Шаг 6: Сохранение обработанного датасета

output\_path = '/content/drive/My Drive/df\_processed.csv'

df\_processed.to\_csv(output\_path, index=False)

# Уведомление о завершении сохранения

print("Обработанные датасеты успешно сохранены:")

print(f"- {output\_path}")

Код для подпункта «Этап 3. Избавление от сильно коррелирующих признаков»

# Этап 3. Шаг 1: Загрузка данных и первичный анализ

input\_path = "/content/drive/My Drive/df\_processed.csv"

df\_processed = pd.read\_csv(input\_path)

# Вывод первых строк для проверки структуры данных

print("Первая информация о датасете:")

print(df\_processed.head())

# Вывод описания данных (типы данных, пропуски, объем памяти)

print("\nОписание данных:")

print(df\_processed.info())

# Этап 3. Шаг 2: Отбор числовых столбцов и визуализация корреляции

# Выбираем только числовые столбцы (float, int, bool)

numerical\_columns = df\_processed.select\_dtypes(include=['float64', 'int64', 'bool']).columns

# Исключаем первый столбец (если это, например, индекс или идентификатор)

numerical\_columns = numerical\_columns[1:]  # Убираем первый столбец

# Рассчитываем корреляционную матрицу

correlation\_matrix = df\_processed[numerical\_columns].corr()

# Визуализация корреляционной матрицы с помощью тепловой карты

plt.figure(figsize=(15, 10))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', linewidths=0.5)

plt.title("Correlation Matrix of Features")

plt.show()

# Этап 3. Шаг 3: Удаление признаков с высокой корреляцией

# Устанавливаем порог для корреляции

threshold = 0.5

# Создаем копию датафрейма для дальнейшей работы, исключаем 'AppID'

df\_reduced = df\_processed.copy()

df\_reduced.drop(columns=['AppID'], inplace=True)

# Находим и удаляем признаки с низкой корреляцией (если сумма абсолютных корреляций по столбцу < порог)

uncorrelated\_features = correlation\_matrix.columns[(correlation\_matrix.abs().sum(axis=0) - 1) <= 0.5]

if not uncorrelated\_features.empty:

    print(f"Удаляем фичи без корреляции: {list(uncorrelated\_features)}")

    df\_reduced.drop(columns=uncorrelated\_features, inplace=True)

# Пересчитываем корреляционную матрицу после удаления некоррелирующих признаков

correlation\_matrix = df\_reduced.corr()

# Пока есть корреляции выше заданного порога, удаляем признаки

while True:

    # Ищем пары признаков с корреляцией, превышающей порог

    high\_corr = np.where(np.abs(correlation\_matrix) > threshold)

    high\_corr\_pairs = [(correlation\_matrix.index[i], correlation\_matrix.columns[j])

                       for i, j in zip(\*high\_corr) if i != j]

    # Если нет высококоррелирующих признаков, выходим из цикла

    if not high\_corr\_pairs:

        break

    # Выбираем первую пару с высокой корреляцией

    feature1, feature2 = high\_corr\_pairs[0]

    # Удаляем один из признаков в паре (например, второй)

    if feature2 in df\_reduced.columns:

        df\_reduced.drop(columns=[feature2], inplace=True)

    # Пересчитываем корреляционную матрицу после удаления признака

    correlation\_matrix = df\_reduced.corr()

# Восстановим 'AppID' обратно в датафрейм

df\_reduced = pd.concat([df\_reduced, df\_processed['AppID']], axis=1)

# Выводим информацию о результатах

print("Количество признаков после обработки:", df\_reduced.shape[1])

print("Новый DataFrame:")

print(df\_reduced.head())

# Визуализация новой корреляционной матрицы

plt.figure(figsize=(15, 10))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', linewidths=0.5)

plt.title("Correlation Matrix of Features (After Removing Highly Correlated Features)")

plt.show()

# Сохраняем обновленный датасет в новый файл

df\_reduced.to\_csv('/content/drive/My Drive/games\_cleaned\_filtered.csv', index=False)

print("Обновленный датасет сохранен в 'games\_cleaned\_filtered.csv'")

Код для подпункта «Этап 4. Уменьшение размерности»

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Загрузка данных

file\_path = '/content/drive/My Drive/games\_cleaned\_filtered.csv'

df = pd.read\_csv(file\_path)

# Удаляем 'AppID', так как он не участвует в анализе

data\_for\_reduction = df\_reduced.drop(columns=['AppID']).values

# Масштабирование данных

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(data\_for\_reduction)

# Уменьшение размерности с помощью PCA

pca = PCA()

pca\_result = pca.fit\_transform(scaled\_data)

# Объясненная дисперсия

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

cumulative\_variance = explained\_variance\_ratio.cumsum()

# Находим минимальное количество компонент, объясняющих 95% дисперсии

threshold = 0.95

num\_components = next(i for i, total in enumerate(cumulative\_variance) if total >= threshold) + 1

print(f"Общая объяснённая дисперсия (суммарно): {cumulative\_variance[-1]:.4f}")

print(f"Количество компонент для объяснения {threshold\*100}% дисперсии: {num\_components}")

# Применяем PCA с найденным числом компонент

pca = PCA(n\_components=num\_components)

reduced\_data = pca.fit\_transform(scaled\_data)

# Сохраняем результат уменьшения размерности

reduced\_df = pd.DataFrame(reduced\_data, columns=[f'PC{i+1}' for i in range(num\_components)])

reduced\_df['AppID'] = df\_reduced['AppID'].values

print("Этап уменьшения размерности завершён.")

# Просмотр первых строк и структуры данных

print("Первая информация о датасете:")

print(reduced\_df.head())

print("\nОписание данных:")

print(reduced\_df.info())

# Сохраняем обновленный датасет в новый файл

reduced\_df.to\_csv('/content/drive/My Drive/games\_ym\_raz.csv', index=False)

print("Обновленный датасет сохранен в 'games\_ym\_raz.csv'")

Код для подпункта «Удаление выбросов»

# Загрузка данных

file\_path = '/content/drive/My Drive/games\_ym\_raz.csv'

df = pd.read\_csv(file\_path)

# Удаляем 'AppID', так как он не участвует в анализе

df = df.drop(columns=['AppID'])

# Изначальное количество записей

initial\_count = len(df)

# Функция для удаления выбросов с мягким порогом

def remove\_outliers\_soft(df, factor=3):

    Q1 = df.quantile(0.25)

    Q3 = df.quantile(0.75)

    IQR = Q3 - Q1

    # Используем мягкий порог для удаления выбросов

    return df[~((df < (Q1 - factor \* IQR)) | (df > (Q3 + factor \* IQR))).any(axis=1)]

# Удаление выбросов с мягким порогом

df\_cleaned = remove\_outliers\_soft(df)

# Итоговое количество записей

final\_count = len(df\_cleaned)

# Процентное изменение

removed\_percentage = ((initial\_count - final\_count) / initial\_count) \* 100

# Результаты

print(f"Изначальное количество записей: {initial\_count}")

print(f"Итоговое количество записей: {final\_count}")

print(f"Процент удаленных выбросов: {removed\_percentage:.2f}%")

Код для подпункта «Кластеризация методом K-means»

from sklearn.cluster import KMeans

# вычислить данные графика локтя

inertias = []

abc=0

for k in range(1, 201):

    abc=abc+1

    kmeans = KMeans(n\_clusters=k, random\_state=0).fit(df)

    inertias.append(kmeans.inertia\_)

    print(abc/2,'%')

# построить график локтя

plt.plot(range(1, 201), inertias, 'bx-')

plt.xlabel('Количество кластеров')

plt.ylabel('Инерция')

plt.title('График локтя')

plt.show()

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

for i in range(44,55,2):

  print('при ',i,' кластерах')

  # применить кластеризацию с оптимальным количеством кластеров (определенным из графика локтя)

  n\_clusters =i # значение оптимального количества кластеров

  kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0).fit(df\_cleaned)

  # вывести метрики кластеризации

  print('Силуэтный коэффициент:', silhouette\_score(df\_cleaned, kmeans.labels\_))

  print('Индекс Калинского-Харабаса:', calinski\_harabasz\_score(df\_cleaned, kmeans.labels\_))

  print('Индекс Дэвиса-Болдина:', davies\_bouldin\_score(df\_cleaned, kmeans.labels\_))

  # вывести количество значений в каждом кластере

  for i in range(n\_clusters):

      print(f'Количество значений в кластере {i}: {len(df\_cleaned[kmeans.labels\_ == i])}')

Код для подпункта «Этап 5.2. Кластеризация методом иерархической кластеризации (Ward)»

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

# Выполнение иерархической кластеризации

# 'ward' - метод минимизации дисперсии, подходит для большинства данных

linkage\_matrix = linkage(df\_cleaned, method='ward')

# Построение дендрограммы

plt.figure(figsize=(12, 8))

dendrogram(linkage\_matrix, truncate\_mode='lastp', p=30, leaf\_rotation=90., leaf\_font\_size=12.)

plt.title("Дендрограмма")

plt.xlabel("Индекс наблюдения или кластера")

plt.ylabel("Евклидово расстояние")

plt.show()

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

# Задание количества кластеров

n\_clusters = 4  # Укажите предполагаемое количество кластеров

labels = fcluster(linkage\_matrix, n\_clusters, criterion='maxclust')

# Оценка качества кластеризации

silhouette\_avg = silhouette\_score(df\_cleaned, labels)

calinski\_harabasz = calinski\_harabasz\_score(df\_cleaned, labels)

davies\_bouldin = davies\_bouldin\_score(df\_cleaned, labels)

print(f"Silhouette Score: {silhouette\_avg:.3f}")

print(f"Calinski-Harabasz Index: {calinski\_harabasz:.3f}")

print(f"Davies-Bouldin Index: {davies\_bouldin:.3f}")

Код для подпункта «Этап 5.3. Кластеризация метод DBSCAN»

from sklearn.neighbors import NearestNeighbors

# Определяем количество соседей для анализа

k = 10

# Рассчитываем расстояния до ближайших соседей

neighbors = NearestNeighbors(n\_neighbors=k)

neighbors\_fit = neighbors.fit(df\_cleaned)

distances, indices = neighbors\_fit.kneighbors(df\_cleaned)

# Сортируем расстояния для построения графика локтя

distances = np.sort(distances[:, k-1])  # Берём расстояние до k-го соседа

# Строим график локтя

plt.figure(figsize=(10, 6))

plt.plot(distances)

plt.title("График локтя для определения eps в DBSCAN")

plt.xlabel("Точки данных (упорядоченные)")

plt.ylabel(f"Расстояние до {k}-го ближайшего соседа")

plt.grid()

plt.show()

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

from sklearn.decomposition import PCA

# Параметры DBSCAN (значения eps и min\_samples подберите по графику локтя)

eps = 0.5  # примерное значение, заменить на оптимальное

min\_samples = 5

# Выполняем кластеризацию DBSCAN

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)

labels = dbscan.fit\_predict(df\_cleaned)

# Сокращаем размерность данных до 2D для визуализации с помощью PCA

pca = PCA(n\_components=2)

reduced\_data = pca.fit\_transform(df\_cleaned)

# Визуализация результатов

plt.figure(figsize=(10, 6))

unique\_labels = set(labels)

colors = [plt.cm.tab10(i / len(unique\_labels)) for i in range(len(unique\_labels))]

for label, color in zip(unique\_labels, colors):

    if label == -1:  # Выделяем шум отдельным цветом (кластер -1)

        color = 'black'

        label\_name = "Шум"

    else:

        label\_name = f"Кластер {label}"

    plt.scatter(reduced\_data[labels == label, 0],

                reduced\_data[labels == label, 1],

                c=[color], label=label\_name, s=20)

plt.title("Результаты кластеризации DBSCAN")

plt.xlabel("Главный компонент 1")

plt.ylabel("Главный компонент 2")

plt.legend()

plt.grid()

plt.show()

Код для подпункта «Этап 6. Построение модели рекомендаций»

# Загрузка данных

file\_path = '/content/drive/My Drive/games\_ym\_raz.csv'

games = pd.read\_csv(file\_path)

features = games.drop(columns=['AppID'])  # 'AppID' — целевая переменная

target = games['AppID']

from sklearn.neighbors import NearestNeighbors

# Создаем модель Nearest Neighbors

knn\_model = NearestNeighbors(n\_neighbors=6, metric='euclidean')

knn\_model.fit(features)

# Функция для рекомендации

def recommend\_games\_v1(game\_id, data, model, n\_recommendations=10):

    # Получаем фичи игры по ID

    game\_features = data[data['AppID'] == game\_id].drop(columns=['AppID'])

    if game\_features.empty:

        return f"Игра с ID {game\_id} не найдена!"

    # Находим соседей

    distances, indices = model.kneighbors(game\_features, n\_neighbors=n\_recommendations+1)

    # Возвращаем рекомендованные ID игр

    recommended\_ids = data.iloc[indices[0]]['AppID'].values[1:]  # Пропускаем саму игру

    return recommended\_ids

import joblib

# Сохранение модели в файл

joblib.dump(knn\_model, 'nearest\_neighbors\_model.pkl')

# Тестируем на примере

game\_id\_to\_recommend = 730  # Введите ID игры для теста

recommendations = recommend\_games\_v1(game\_id\_to\_recommend, games, knn\_model)

print(f"Рекомендованные игры для ID {game\_id\_to\_recommend}: {recommendations}")

Код для подпункта «Этап 7. Разработка интерфейса и условная оценка модели»

# Шаг 1: Загрузка исходного датасета с полным набором данных

df = pd.read\_csv('/content/drive/My Drive/games\_cleaned.csv')

# Создание нового датасета с двумя ключевыми столбцами: AppID и name

new\_df = df[['AppID', 'name']]

# Сохранение нового датасета для дальнейшего использования

new\_df.to\_csv('/content/drive/My Drive/id-name.csv', index=False)

import customtkinter as ctk

import tkinter as tk

import pandas as pd

from rapidfuzz import process

# Загрузка датасета один раз

dataset = pd.read\_csv('id-name.csv')

# Приведение к единому формату

dataset['name\_normalized'] = dataset['name'].str.lower().str.strip()

def get\_name\_by\_appid(appid):

    """Получение имени игры по AppID"""

    row = dataset.loc[dataset['AppID'] == appid]

    if not row.empty:

        return row.iloc[0]['name']

    return None

def get\_appid\_by\_name(name):

    """Получение AppID игры по имени (с нечётким поиском)"""

    name\_normalized = name.lower().strip()

    match = process.extractOne(name\_normalized, dataset['name\_normalized'])

    if match and match[1] > 80:  # Порог точности (например, 80%)

        row = dataset.loc[dataset['name\_normalized'] == match[0]]

        if not row.empty:

            return row.iloc[0]['AppID']

    return None

# Загрузка данных

file\_path = 'games\_ym\_raz.csv'

games = pd.read\_csv(file\_path)

# Загрузка модели

import joblib

knn\_model = joblib.load('nearest\_neighbors\_model.pkl')

game\_ids = games['AppID']

features = games.drop(columns=['AppID'])

# Функция для рекомендации

def recommend\_games\_v1(game\_id, data, model, n\_recommendations=10):

    # Получаем фичи игры по ID

    game\_features = data[data['AppID'] == game\_id].drop(columns=['AppID'])

    if game\_features.empty:

        return f"Игра с ID {game\_id} не найдена!"

    # Находим соседей

    distances, indices = model.kneighbors(game\_features, n\_neighbors=n\_recommendations+1)

    # Возвращаем рекомендованные ID игр

    recommended\_ids = data.iloc[indices[0]]['AppID'].values[1:]  # Пропускаем саму игру

    return recommended\_ids

# Настройка темы и начальных параметров CustomTkinter

ctk.set\_appearance\_mode("dark")  # Вариант оформления: "light", "dark", "system"

ctk.set\_default\_color\_theme("blue")  # Цветовая тема: "blue", "dark-blue", "green"

# Создаем главное окно

root = ctk.CTk()

root.title("Рекомендация игр")

root.geometry("550x380")

root.resizable(False, False)

# Функция обработки введенной строки

def process\_input():

    user\_input = input\_entry.get()  # Получаем текст из поля ввода

    if user\_input.strip():  # Проверяем, что строка не пустая

        # Обрабатываем строку (переворачиваем)

        game\_id\_to\_recommend = get\_appid\_by\_name(user\_input)

        recommendations = recommend\_games\_v1(game\_id\_to\_recommend, games, knn\_model)

        results=[]

        for i in recommendations:

            results=results+[get\_name\_by\_appid(i)]

        # Очищаем поле ввода

        input\_entry.delete(0, tk.END)

        # Выводим обработанную строку и обновляем список

        processed\_label.configure(text=f"Обработанная строка: {get\_name\_by\_appid(game\_id\_to\_recommend)}")

        results\_text.configure(state="normal")  # Разрешаем запись

        results\_text.delete(1.0, "end")  # Очищаем текстовое поле

        results\_text.insert("end", "\n".join(results))  # Выводим список

        results\_text.configure(state="disabled")  # Запрещаем редактирование

# Создаем элементы интерфейса

input\_label = ctk.CTkLabel(root, text="Введите игру:", font=("Comic Sans MS", 14))

input\_label.grid(row=0, column=0, padx=10, pady=15, sticky="w")

input\_entry = ctk.CTkEntry(root, placeholder\_text="Введите название игры", width=250)

input\_entry.grid(row=0, column=1, padx=10, pady=15, sticky="w")

process\_button = ctk.CTkButton(root, text="Найти!", command=process\_input, width=100)

process\_button.grid(row=0, column=2, padx=10, pady=15, sticky="w")

processed\_label = ctk.CTkLabel(root, text="Рекомендованные игры:", font=("Comic Sans MS", 14))

processed\_label.grid(row=1, column=0, columnspan=3, padx=10, pady=10, sticky="w")

results\_label = ctk.CTkLabel(root, text="Топ 10 игр:", font=("Comic Sans MS", 14))

results\_label.grid(row=2, column=0, columnspan=3, padx=10, pady=10, sticky="w")

results\_text = ctk.CTkTextbox(root, width=500, height=190, state="disabled", font=("Comic Sans MS", 12))

results\_text.grid(row=3, column=0, columnspan=3, padx=10, pady=15)

# Запуск главного цикла приложения

root.mainloop()