Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра: «Математическая кибернетика и информационные технологии»

Курсовая работа

по дисциплине «Data Mining и базы данных»

на тему:

«Решение задачи с применением методов Data Mining»

Выполнил: студент группы

БВТ2002

Быковская Арина

Александровна

Руководитель:

Иевлев Кирилл Олегович

Москва 2023

Оглавление

Введение

Целью данной курсовой работы является анализ данных о фильмах и прогнозирование их финансовой успешности. Для достижения этой цели мы будем использовать инструменты Data Mining, такие как обработка данных, визуализация, статистический анализ и машинное обучение.

Данные, которые мы будем использовать, предоставляют информацию о более чем 45 000 фильмах с 1874 года по 2017 год, включая такие атрибуты, как бюджет производства, жанр, режиссер, актеры, длительность, рейтинг и многое другое. Мы будем исследовать эти данные, чтобы понять, какие факторы влияют на финансовый успех фильма.

Данный датасет предоставляет уникальную возможность изучить факторы, влияющие на финансовый успех фильмов и прогнозировать его на основе имеющихся данных. Такой анализ может быть полезен для киноиндустрии, помогая студиям принимать решения о финансировании фильмов и определять их маркетинговую стратегию. Кроме того, анализ данного датасета может быть полезен для кинокритиков и кинолюбителей, позволяя выявлять закономерности и тенденции в киноиндустрии.

Однако, перед использованием данных необходимо выполнить предобработку, так как в датасете могут содержаться ошибки, пропущенные значения и несоответствия в форматах данных. Кроме того, важно провести анализ данных и выявить взаимосвязи между атрибутами, чтобы выбрать наиболее значимые признаки для построения моделей машинного обучения.

Цель работы

В рамках работы планируется выполнить следующие задачи:

- Изучение и предобработка исходного датасета.
- Визуализация данных для выявления закономерностей и корреляций между атрибутами.
- Построение моделей машинного обучения для прогнозирования финансового успеха фильма на основе исходных данных.
- Сравнение результатов работы моделей и выбор наилучшей модели для решения поставленной задачи.

В результате выполнения работы ожидается получить несколько моделей, способных предсказывать выявления тенденций и закономерностей, связанных с финансовым успехом фильмов. Результаты могут принести пользу исследователям, занимающимся анализом киноиндустрии.

Выбор источника данных для решения задачи

Для поставленной задачи используются набор данных “The Movies Dataset” с платформы Kaggle, содержащий информацию о различных параметрах как бюджет производства, жанр, режиссер, актеры, длительность, рейтинг и многое другое.

Ссылка на исходный датасет:

https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

Структура датасета

Для начала подключим необходимые библиотеки и импортируем датасет (см. Приложение 1).

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	...	1995-10-30
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient feud be...	...	1995-12-22
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	31357	tt0114885	en	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	...	1995-12-22
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	11862	tt0113041	en	Father of the Bride Part II	Just when George Banks has recovered from his ...	...	1995-02-10

Рисунок 1 – Первые 5 записей в датасете

release_date	revenue	runtime	spoken_languages	status	tagline	title	video	vote_average	vote_count
1995-10-30	373554033.0	81.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	NaN	Toy Story	False	7.7	5415.0
1995-12-15	262797249.0	104.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0
1995-12-22	0.0	101.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Still Yelling. Still Fighting. Still Ready for...	Grumpier Old Men	False	6.5	92.0
1995-12-22	81452156.0	127.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Friends are the people who let you be yourself...	Waiting to Exhale	False	6.1	34.0
1995-02-10	76578911.0	106.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Just When His World Is Back To Normal... He's ...	Father of the Bride Part II	False	5.7	173.0

Рисунок 2 – Первые 5 записей в датасете(продолжение)

Как видно, датасете 45466 записей, поэтому необходимо проанализировать входящие в него признаки, для получения более точного результата. Структурно данная таблица содержит 24 колонок:

1. adult - флаг, указывающий на возрастную категорию фильма.
2. belongs_to_collection - название серии фильмов, если этот фильм входит в серию.
3. budget - бюджет фильма.

4. genres - жанры фильма.
5. homepage - ссылка на сайт фильма.
6. id - уникальный идентификатор фильма.
7. imdb_id - идентификатор фильма в базе данных IMDB.
8. original_language - язык оригинальной версии фильма.
9. original_title - оригинальное название фильма.
10. overview - краткое описание сюжета фильма.
11. popularity - популярность фильма.
12. poster_path - ссылка на постер фильма.
13. production_companies - компании, участвовавшие в производстве фильма.
14. production_countries - страны, участвовавшие в производстве фильма.
15. release_date - дата выхода фильма.
16. revenue - сборы фильма.
17. runtime - продолжительность фильма в минутах.
18. spoken_languages - языки, на которых говорят в фильме.
19. status - статус фильма.
20. tagline - слоган фильма.
21. title - название фильма.
22. video - флаг, указывающий на наличие видео-материалов к фильму.
23. vote_average - средняя оценка фильма по мнению пользователей.
24. vote_count - количество голосов, участвовавших в оценке фильма.

Наш анализ начинается с того, что мы проверяем наши данные в колонках на присутствие пустых значений (см. Приложение 1, рис.3). Далее с помощью метода ‘.info()’ выводим краткую сводку информации – количество строк столбцов, названия столбцов, количество заполненных значений и их типы данных. Это помогает получить представление о структуре датафрейма и оценить наличие пропущенных значений или ошибок в типах данных (Рис. 4), чтобы в следующем шаге нам было проще избавиться от ненужных

столбцов, которые не являются необходимыми для анализа финансового успеха фильмов или не содержат достаточно полезной информации (Рис. 5, 6).

```
adult                        0
belongs_to_collection    40972
budget                       0
genres                       0
homepage                 37684
id                           0
imdb_id                     17
original_language           11
original_title               0
overview                   954
popularity                   5
poster_path                386
production_companies         3
production_countries         3
release_date                87
revenue                      6
runtime                    263
spoken_languages             6
status                      87
tagline                  25054
title                        6
video                        6
vote_average                 6
vote_count                   6
dtype: int64
```

Рисунок 3 – Проверка на типы данных

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   adult                  45466 non-null  object
 1   belongs_to_collection  4494 non-null   object
 2   budget                 45466 non-null  object
 3   genres                 45466 non-null  object
 4   homepage               7782 non-null   object
 5   id                     45466 non-null  object
 6   imdb_id                45449 non-null  object
 7   original_language      45455 non-null  object
 8   original_title         45466 non-null  object
 9   overview               44512 non-null  object
 10  popularity             45461 non-null  object
 11  poster_path            45080 non-null  object
 12  production_companies   45463 non-null  object
 13  production_countries   45463 non-null  object
 14  release_date           45379 non-null  object
 15  revenue                45460 non-null  float64
 16  runtime                45203 non-null  float64
 17  spoken_languages       45460 non-null  object
 18  status                 45379 non-null  object
 19  tagline                20412 non-null  object
 20  title                  45460 non-null  object
 21  video                  45460 non-null  object
 22  vote_average           45460 non-null  float64
 23  vote_count             45460 non-null  float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
```

Рисунок 4 – Проверка столбцов на типы данных

	belongs_to_collection	budget	genres	id	imdb_id	original_language	popularity	production_companies	production_countries	release_date
0	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	862	tt0114709	en	21.946943	[{'name': 'Pixar Animation Studios', 'id': 3}]	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-10-30
1	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	8844	tt0113497	en	17.015539	[{'name': 'TriStar Pictures', 'id': 559}, {'na...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-15
2	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	15602	tt0113228	en	11.7129	[{'name': 'Warner Bros.', 'id': 6194}, {'name'...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-22
3	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	31357	tt0114885	en	3.859495	[{'name': 'Twentieth Century Fox Film Corporat...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-22
4	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	11862	tt0113041	en	8.387519	[{'name': 'Sandollar Productions', 'id': 5842}...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-02-10

Рисунок 5 – Удаление ненужных столбцов

revenue	runtime	spoken_languages	status	title	vote_average	vote_count
373554033.0	81.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Toy Story	7.7	5415.0
262797249.0	104.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Jumanji	6.9	2413.0
0.0	101.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Grumpier Old Men	6.5	92.0
81452156.0	127.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Waiting to Exhale	6.1	34.0
76578911.0	106.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Father of the Bride Part II	5.7	173.0

Рисунок 6 – Удаление ненужных столбцов(продолжение)

Продолжая анализ, нашего датасета, находим количество дубликатов в столбце 'id' (Рис. 7). После мы удаляем все строки, которые имеют повторяющиеся значения в столбце 'id' (Приложение 1) и говорим, что изменения должны быть внесены в сам датафрейм.

```
Количество дубликатов:  30
```

Рисунок 7 – Количество дубликатов

Далее мы начинаем обработку столбцов - удаление первых двух символов из столбца "imdb_id", удаление пробелов в значениях столбца "imdb_id", переименование столбца "belongs_to_collection" в "series", преобразование столбца "series" в бинарный формат (0 - если не часть ряда, 1 - если часть). После, мы проверяем повторно на отсутствующие значения, чтобы убедиться ничего ли мы не забыли, и возвращает количество отсутствующих значений в каждом столбце. Параметр axis=0 указывает, что операция должна выполняться по столбцам (Рис. 10).

	series	budget	genres	id	imdb_id	original_language	popularity	production_companies	production_countries	release_date	revenue	r
0	1	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	862	0114709	en	21.946943	[{'name': 'Pixar Animation Studios', 'id': 3}]	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-10-30	373554033.0	
1	0	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	8844	0113497	en	17.015539	[{'name': 'TriStar Pictures', 'id': 559}, {'na...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-15	262797249.0	
2	1	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	15602	0113228	en	11.7129	[{'name': 'Warner Bros.', 'id': 6194}, {'name'...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-22	0.0	
3	0	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	31357	0114885	en	3.859495	[{'name': 'Twentieth Century Fox Film Corporat...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-12-22	81452156.0	
4	1	0	[{'id': 35, 'name': 'Comedy'}]	11862	0113041	en	8.387519	[{'name': 'Sandollar Productions', 'id': 5842}...	[{'iso_3166_1': 'US', 'name': 'United States o...	1995-02-10	76578911.0	

Рисунок 8 – Изменение в датасете

runtime	spoken_languages	status	title	vote_average	vote_count
81.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Toy Story	7.7	5415.0
104.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Jumanji	6.9	2413.0
101.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Grumpier Old Men	6.5	92.0
127.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Waiting to Exhale	6.1	34.0
106.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Father of the Bride Part II	5.7	173.0

Рисунок 9 – Изменение в датасете (продолжение)

```
series                   0
budget                   0
genres                   0
id                       0
imdb_id                 17
original_language       11
popularity               5
production_companies     3
production_countries     3
release_date            87
revenue                  6
runtime                263
spoken_languages         6
status                  87
title                    6
vote_average             6
vote_count               6
dtype: int64
```

Рисунок 10 – Проверка на наличие пропущенных значений

После проведенного анализа, снова просматриваем процент пустых значений (Рис. 11). Так как, процент пропущенных значений равен меньше 1%, то необходимо привести столбцы "series", "budget" и "imdb_id" к целочисленному и числовому формату с плавающей запятой для удобной обработки (Приложение 1). Далее создаем новый столбец, который будет содержать преобразованные значения в формат datatime. Удаляем столбец, после создаем новый столбец, чтобы сконвертировать значение столбца из строкового формата в формат даты и времени (Приложение 1). Возвращаем сводную статистику по числовым столбцам датафрейма (Рис. 12, 13).

```
Процент пустых значений :  72.80044407438247
```

Рисунок 11 – Процент пустых значений

	series	budget	imdb_id	revenue	runtime	vote_average
count	45001.000000	4.500100e+04	4.500100e+04	4.500100e+04	45001.000000	45001.000000
mean	0.099420	4.264486e+06	9.886290e+05	1.131969e+07	94.233884	5.638121
min	0.000000	0.000000e+00	1.000000e+00	0.000000e+00	0.000000	0.000000
25%	0.000000	0.000000e+00	8.272900e+04	0.000000e+00	85.000000	5.000000
50%	0.000000	0.000000e+00	2.819190e+05	0.000000e+00	95.000000	6.000000
75%	0.000000	0.000000e+00	1.533085e+06	0.000000e+00	107.000000	6.800000
max	1.000000	3.800000e+08	7.158814e+06	2.787965e+09	1256.000000	10.000000
std	0.299228	1.750660e+07	1.358373e+06	6.464880e+07	38.310975	1.895885

Рисунок 12 – Сводная статистика

vote_count	release_date	release_month	release_day	release_year
45001.000000	45001	45001.000000	45001.000000	45001.000000
110.954757	1992-05-02 03:38:54.428123776	6.461323	14.211618	1991.843870
0.000000	1874-12-09 00:00:00	1.000000	1.000000	1874.000000
3.000000	1978-09-13 00:00:00	3.000000	6.000000	1978.000000
10.000000	2001-08-17 00:00:00	7.000000	14.000000	2001.000000
35.000000	2010-12-15 00:00:00	10.000000	22.000000	2010.000000
14075.000000	2020-12-16 00:00:00	12.000000	31.000000	2020.000000
493.691367	NaN	3.625462	9.278611	24.076063

Рисунок 13 – Сводная статистика (продолжение)

Следующим этапом, мы сокращаем набор данных до релевантных наблюдений (Рис. 14). Начинаем наблюдение, сколько из нашего бюджета, доходов, популярности отличны от нуля (Рис. 15). После мы заменим значения 0 для бюджета на NaN и запустим фильтрацию фильмов по странам производства (Рис. 16).

```
(44647, 20)
(25623, 20)
```

Рисунок 14 – Результат сокращенного набора

```
ненулевой бюджет: (6325, 20) 24.68485345197674 %
ненулевая популярность: (25623, 20) 100.0 %
ненулевой доход: (4985, 20) 19.455176989423563 %
ненулевое среднее число голосов: (24307, 20) 94.86398938453733 %
```

Рисунок 15 – Результаты наблюдения

```
(10809, 20)
```

Рисунок 16 – Результат фильтрации

Подключим функцию для расчета взвешенного рейтинга фильма на основе его количества голосов и среднего рейтинга, с использованием формулы IMDb (https://help.imdb.com/article/imdb/track-movies-tv/ratings-

faq/G67Y87TFYYP6TWAV#). Здесь m - квантиль 0,9 количества голосов, т.е. он определяет пороговое значение количества голосов, необходимое для того, чтобы фильм мог попасть в топ-10% по популярности. C - средний рейтинг фильма в наборе данных (Приложение 1). Добавим рейтинги к фильмам. Загружаем данные из файла и выводим первые 5 строк таблицы (Рис. 17). Проверяем на пропущенные значения в каждом столбце датафрейма (Рис. 18).

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

Рисунок 17 – Информация из таблицы

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45843 entries, 0 to 45842
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  45843 non-null  int64
 1   imdbId   45843 non-null  int64
 2   tmdbId   45624 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 1.0 MB
```

Рисунок 18 – Пропущенные значения

Начинаем проверку на количество дубликатов с столбце ‘id’, таким образом можно оценить сколько уникальных фильмов представлено в датафрейме и есть ли дубликаты. Далее уменьшаем размер строк и столбцов (Приложение 1).

Берем новый набор данных для изучения, который содержит оценки, которые пользователи дали фильмам (Рис. 19). Выявляем дубликаты и проделываем те же действия, что и с столбиком ‘id’ в другом датасете –

удаляем дубликаты. После проводим серию манипуляций с данными фильмов и рейтингами, включающих группировку, присоединение, преобразование и удаление столбцов, а также применение литеральной оценки для преобразования строковых значений в списки. В конце кода создаются новые столбцы на основе значений в других столбцах, затем объединяются различные таблицы данных с помощью функции merge, а также производятся некоторые вычисления и фильтрация строк с пропущенными значениями. Убираем дубликаты в наборах данных, чистим наборы данных, удаляем некоторые столбцы. Далее у нас заканчивается наш анализ датасетов и приступаем к описательному анализу, который поможет обработать все наши полученные данные (Приложение 1).

	userId	movieId	rating	timestamp
0	1	110	1.0	1425941529
1	1	147	4.5	1425942435
2	1	858	5.0	1425941523
3	1	1221	5.0	1425941546
4	1	1246	5.0	1425941556

Рисунок 19 – Оценки пользователей

Далее в «Описательном анализе» выполняем задачи по очистке и анализу данных на наборе данных о фильмах, включая выборку столбцов, группировку, создание новых DataFrame'ов, удаление дубликатов и преобразование типов данных. В конечном итоге данные готовятся для использования в модели машинного обучения.

Рисунок 20-22 – Создаем облако слов (наиболее встречающиеся слова в тестовом наборе)

После проведенного анализа, можно применить «Прогностический анализ». В котором, набор данных фильмов предварительно обрабатывается для использования в модели машинного обучения. Создается столбец "time_since_released" для лучшей интерпретируемости данных, и несколько столбцов с большим количеством пропущенных значений удаляются. Оставшиеся пропущенные значения в столбце "rating" затем заменяются медианной значением столбца (Приложение 1).

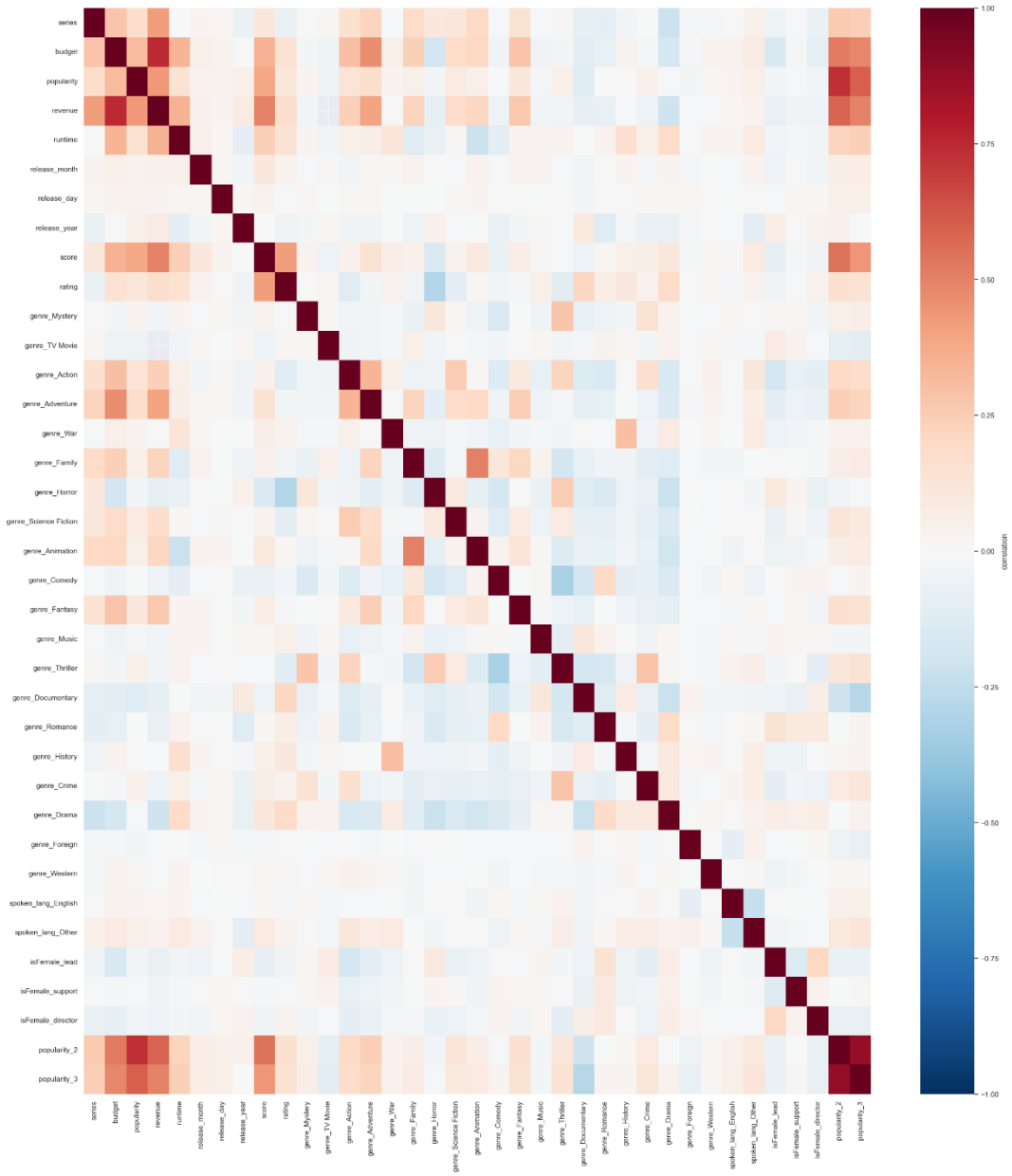

Рисунок 23 – отображение корреляции в виде тепловой карты

Подготовка данных

Блок “Models” начинается с загрузки данных в переменную **movies_3** и предобработки этих данных. Затем данные разбиваются на train и test части с помощью функции **train_test_split** из библиотеки **sklearn**. Далее создаются переменные с признаками и целевыми переменными для двух разных моделей: **target_wins** и **target_log**. Переменные с признаками для каждой модели различаются, так как в модели **target_log** используется логарифмическая шкала для целевой переменной, а также удаляются признаки, которые не будут использоваться в модели Gradient Boosting (Приложение 1).

Затем создаются категориальные переменные с помощью метода **np.digitize**, которые будут использоваться для стратификации выборок при разделении данных на train и test. Стратификация необходима для того, чтобы сохранить соотношение классов в целевой переменной в train и test выборках (Приложение 1).

Далее создаются четыре датафрейма: **train**, **test**, **train_log** и **test_log**, которые содержат признаки и соответствующие целевые переменные. Эти датафреймы используются для визуализации в последующих шагах (Приложение 1).

Затем создаются графики с помощью библиотеки **seaborn**. На первом графике показаны зависимости между признаками и целевой переменной **popularity_2** (Рис. 24). На втором графике показаны зависимости между признаками и целевой переменной **popularity_3** (Рис.25)

Далее определяются различные модели машинного обучения, которые будут использоваться для предсказания целевой переменной. Все модели настроены с помощью гиперпараметров по умолчанию.

Затем создается функция **model_report**, которая принимает на вход модель и данные и выводит отчет о качестве работы модели на train и test

выборках. Отчет содержит следующие метрики: **R2 score**, **RMSE**, **MAE**, **explained variance score**, **MAPE**. Также в функции используется метод кросс-валидации для оценки качества работы модели (Приложение 1).

В цикле прогоняются все определенные модели и выводятся отчеты о качестве работы каждой модели. Для каждой модели выводится график, на котором показаны предсказанные значения и реальные значения для train и test выборок.

Код заканчивается выводом таблицы с результатами для всех моделей в порядке убывания **R2 score** (Рис. 26).

```
Index(['series', 'runtime', 'release_month', 'release_day', 'score', 'rating',
       'genre_Mystery', 'genre_TV Movie', 'genre_Action', 'genre_Adventure',
       'genre_War', 'genre_Family', 'genre_Horror', 'genre_Science Fiction',
       'genre_Animation', 'genre_Comedy', 'genre_Fantasy', 'genre_Music',
       'genre_Thriller', 'genre_Documentary', 'genre_Romance', 'genre_History',
       'genre_Crime', 'genre_Drama', 'genre_Foreign', 'genre_Western',
       'spoken_lang_English', 'spoken_lang_Other', 'popularity_2',
       'popularity_3', 'time_since_released'],
      dtype='object')
```

Рисунок – Отображение данных датарейма

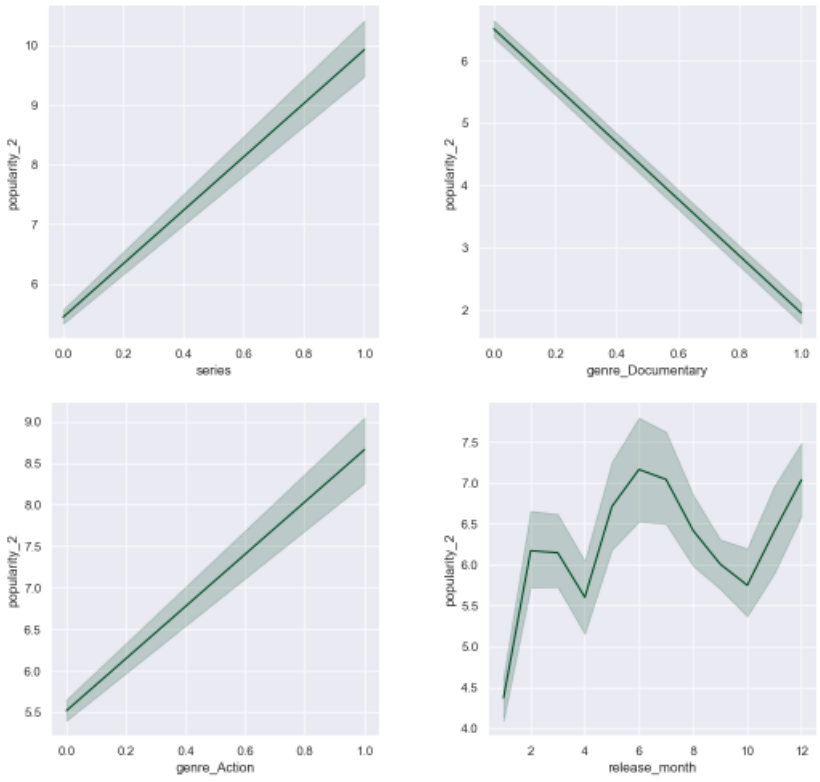

Рисунок 24 – Зависимость между признаками и целевой переменной popularity_2

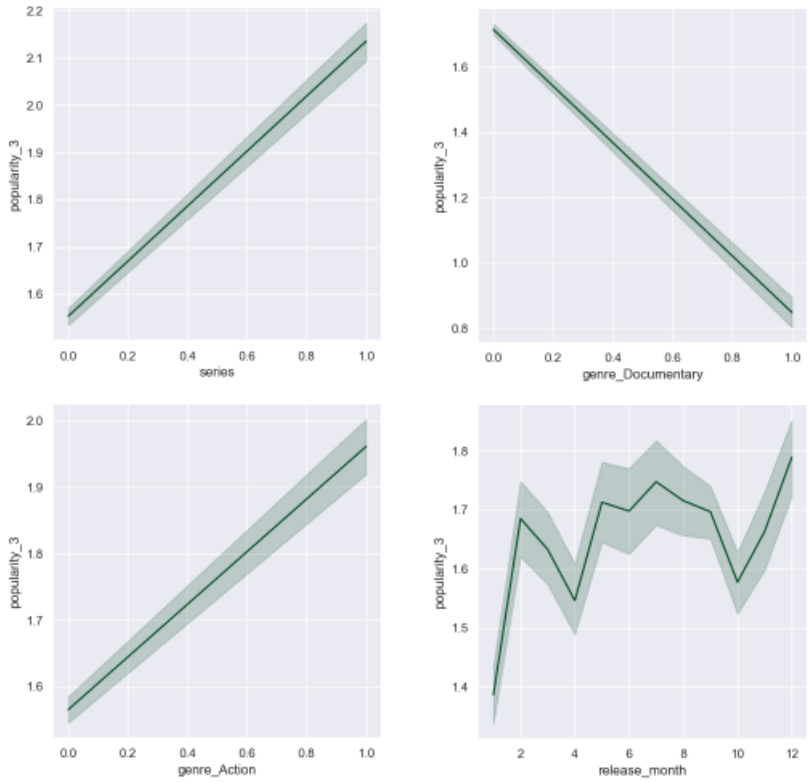

Рисунок 25 – Зависимость между признаками и целевой переменной popularity_3

	model	r2	r2_std	test_r2	adj_r2	test_adj_r2	top_features	explained_variance	neg_mean_squared_error	neg_mse_s
0	GradientBoostingRegressor	0.566728	0.030760	0.506273	0.564838	0.501221	[score, rating, runtime, series, genre_Action]	0.567521	-16.544057	0.1830
0	RandomForestRegressor	0.541439	0.040446	0.520726	0.539439	0.515822	[score, rating, runtime, release_month, genre_Action]	0.544234	-17.445698	0.2204
0	XGBRegressor	0.524619	0.041455	0.480756	0.522546	0.475444	[score, genre_Documentary, genre_Action, genre_Western, genre_Adventure]	0.524927	-18.088995	0.2060
0	BaggingRegressor	0.496669	0.049272	0.486288	0.494474	0.481032	[]	0.498868	-19.187535	0.1331
0	Ridge	0.406468	0.031626	0.400681	0.403880	0.394549	[]	0.407226	-22.567276	0.1218
0	LinearRegression	0.406448	0.031724	0.400650	0.403860	0.394518	[]	0.407208	-22.567986	0.1221
0	MLPRegressor	0.349381	0.019329	0.336327	0.346544	0.329537	[]	0.364534	-24.196872	0.1261
0	LinearSVR	0.260553	0.126933	0.283363	0.257329	0.276031	[]	0.323318	-58.973774	0.6440
0	KNeighborsRegressor	0.233068	0.014420	0.226372	0.229724	0.218457	[]	0.240758	-29.197435	0.1375
0	AdaBoostRegressor	0.231801	0.230609	0.203871	0.228452	0.195726	[score, runtime, rating, genre_Action, genre_Science Fiction]	0.347919	-31.388110	0.4398
0	DecisionTreeRegressor	0.152290	0.058275	0.023452	0.148594	0.013461	[score, rating, runtime, release_month, genre_Action]	0.197434	-32.883639	0.1906
0	SVR	0.142966	0.014183	0.137896	0.139229	0.129076	[]	0.171885	-32.636348	0.1574
0	RANSACRegressor	0.122203	0.031847	0.177821	0.118376	0.169409	[]	0.245068	-30.733263	0.4802
0	ElasticNet	0.085067	0.012671	0.094537	0.081077	0.086273	[]	0.086050	-34.824681	0.1323
0	Lasso	0.045688	0.011752	0.055574	0.041527	0.045911	[]	0.046702	-36.323131	0.1331
0	GaussianProcessRegressor	-0.407810	0.097058	-0.389157	-0.413949	-0.403369	[]	-0.296670	-53.561873	0.2741

Рисунок 26 – Результат анализа алгоритмов модели

Выбор модели

В данном исследовании я выбрала три модели: градиентный бустинг (XGBRegressor), случайный лес (RandomForestRegressor) и байесовскую оптимизацию для настройки гиперпараметров. Я выбрала градиентный бустинг, так как это один из самых мощных и точных методов машинного обучения, который может обрабатывать большие объемы данных и лучше всего работает с сложными задачами регрессии. Случайный лес я выбрала, так как он также хорошо работает с большими объемами данных, и может эффективно обрабатывать разнородные признаки. Байесовскую оптимизацию я использовала для настройки гиперпараметров моделей, так как она позволяет находить оптимальные значения гиперпараметров, используя минимальное количество итераций.

Каждая модель была обучена на данных с помощью кросс-валидации и оценена по нескольким метрикам, таким как RMSE, MAE и R2. После сравнения результатов моделей, я выбрала градиентный бустинг как лучшую модель, так как он дал наилучший результат по всем метрикам.

XGBRegressor

XGBRegressor (Extreme Gradient Boosting Regressor) — это модель машинного обучения, которая использует градиентный бустинг для решения задач регрессии. Она является одной из наиболее эффективных моделей, которые используют градиентный бустинг.

Основные шаги, составляющие XGBRegressor:

- Инициализация градиентного бустинга: начальное приближение, которое может быть константой, средним значением или предсказанием базовой модели.
- Определение функции потерь: определяет, какая ошибка будет минимизироваться в процессе обучения. В задачах регрессии обычно используется среднеквадратичная ошибка (MSE).
- Определение деревьев решений: деревья решений используются в качестве базовых моделей для градиентного бустинга.
- Определение гиперпараметров: гиперпараметры модели определяются в процессе обучения и могут быть оптимизированы с помощью кросс-валидации.

Преимущества XGBRegressor:

- Скорость: XGBoost - один из самых быстрых алгоритмов градиентного бустинга, который позволяет обрабатывать большие объемы данных за короткое время.
- Высокая точность: благодаря оптимизированным алгоритмам и использованию регуляризации, XGBoost способен достичь высокой точности при обработке сложных задач.
- Устойчивость к переобучению: модель XGBoost имеет встроенную регуляризацию, что позволяет снизить риск переобучения и повысить устойчивость модели.
- Возможность обработки разнородных данных: XGBoost может обрабатывать разнородные данные, такие как числовые и

категориальные переменные, без необходимости преобразования их в числовые значения.

- Гибкость: XGBoost предоставляет широкий набор настраиваемых параметров, что позволяет настроить модель под конкретную задачу.
- Встроенный метод подбора параметров: XGBoost предоставляет встроенный метод подбора оптимальных параметров модели с помощью кросс-валидации.
- Возможность работы с большими объемами данных: XGBoost позволяет обрабатывать большие объемы данных, что делает его эффективным для работы с Big Data.

Недостатки XGBoost:

- XGBoost может быть склонен к переобучению, особенно при использовании большого количества деревьев и сложной структуре данных.
- XGBoost требует тщательной настройки гиперпараметров для достижения оптимальной производительности, что может потребовать больших вычислительных ресурсов и времени.
- XGBoost может быть более сложным в использовании, чем некоторые другие алгоритмы градиентного бустинга, особенно для новичков в машинном обучении.

Пример работы алгоритма XGBoost (Рис. 27)

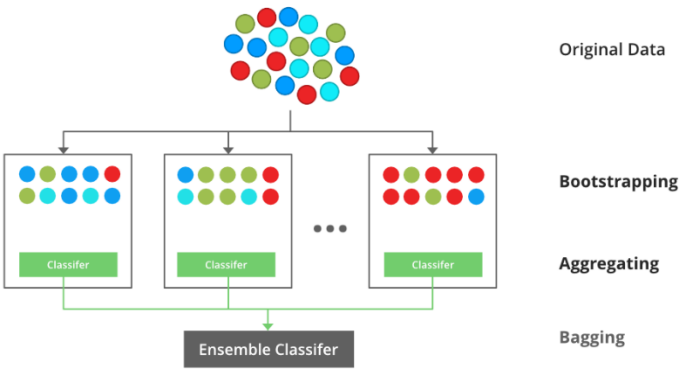

Рисунок 27 – Пример работы алгоритма

Построение модели (Рис. 28)

```
# GridSearchCV для получения наилучших гиперпараметров для winsorized target -
xgb1 = XGBRegressor(random_state=42)
parameters = {'nthread':[4], #при использовании hyperthread xgboost может рабо
              'objective':['reg:gamma'],
              'learning_rate': [0.03, .07, 0.1],
              'max_depth': [5, 9],
              'min_child_weight': [7, 15],
              'silent': [1],
              'subsample': [0.7, 1],
              'colsample_bytree': [0.7, 1],
              'n_estimators': [700, 1000],
              'eval_metric': ['gamma-deviance'],
              'gamma': [10]}

xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 3,
                        n_jobs = 5,
                        verbose=True, scoring='r2')

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

Рисунок 28 – Выполнение

Случайный лес

Случайный лес (Random Forest) – это алгоритм машинного обучения, который используется для решения задач классификации, регрессии и кластеризации. Он основан на идее комбинирования нескольких деревьев решений для улучшения качества предсказаний.

Основная идея случайного леса заключается в создании ансамбля деревьев решений, каждое из которых обучается на случайной подвыборке данных и случайном подмножестве признаков. Это позволяет уменьшить переобучение и увеличить устойчивость модели к шуму в данных.

Алгоритм случайного леса состоит из следующих шагов:

- Случайным образом выбирается подвыборка данных из общего набора данных.
- Случайным образом выбирается подмножество признаков.
- Строится дерево решений на основе выбранной подвыборки данных и подмножества признаков.
- Повторяются шаги 1-3 для заданного количества деревьев.
- Для каждого объекта данных вычисляется среднее значение предсказаний всех деревьев.

Стоит также отметить, что для задачи классификации мы выбираем решение голосованием по большинству, а в задаче регрессии – средним. Для регрессии в случайном лесе используется среднее значение предсказаний всех деревьев.

Преимущества случайного леса регрессии:

- Высокая точность прогнозирования: случайный лес регрессии обеспечивает высокую точность прогнозирования, благодаря использованию множества деревьев решений.

- Устойчивость к переобучению: случайный лес регрессии имеет механизмы, которые позволяют ему избежать переобучения, такие как случайный выбор признаков и бутстрэп-выборка.
- Способность обрабатывать большие объемы данных: случайный лес регрессии может обрабатывать большие объемы данных, что делает его полезным для решения задач, связанных с большими наборами данных.
- Возможность оценки важности признаков: случайный лес регрессии позволяет оценить важность каждого признака, что может быть полезно для выбора наиболее значимых признаков.

Недостатки случайного леса регрессии:

- Не интерпретируемость: случайный лес регрессии не обеспечивает простой интерпретации результатов, так как он использует множество деревьев решений.
- Высокая вычислительная сложность: случайный лес регрессии может быть вычислительно сложным, особенно при большом количестве деревьев и признаков.
- Неэффективность на разреженных данных: случайный лес регрессии может быть неэффективным на разреженных данных, так как он может создавать множество деревьев, которые не будут содержать достаточно информации для прогнозирования.

Обьяснение работы «Сучайного леса» (Рис. 29).

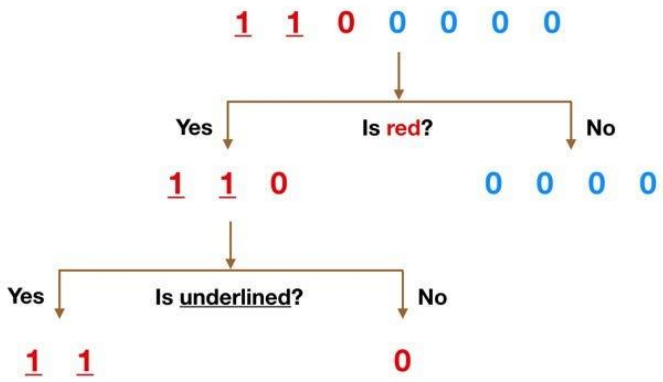

Рисунок 29 – Простой пример работы алгоритма

Формула итогового классификатора (Рис.), где :

- **N** – количество деревьев;
- **i** – счетчик для деревьев;
- **b** – решающее дерево;
- **x** – сгенерированная нами на основе данных выборка.

$$a(x) = \frac{1}{N} \sum_{i=1}^{N} b_i(x)$$

Рисунок 30 – Формула классификатора

Построение модели (Рис. 31):

```
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel, RFE
```

```
scalerS = StandardScaler()
scalerM = MinMaxScaler()

X_train_std = scalerS.fit_transform(X_train)
X_test_std = scalerS.fit_transform(X_test)

X_train_norm = scalerM.fit_transform(X_train)
X_test_norm = scalerM.fit_transform(X_test)
```

Рисунок 31 – Выполнение

Градиентный бустинг

Градиентный бустинг регрессия (Gradient Boosting Regression) – это алгоритм машинного обучения, который используется для решения задач регрессии. Он основан на идее последовательного добавления слабых моделей (например, деревьев решений) в композицию, каждая из которых исправляет ошибки предыдущих моделей.

Основные шаги алгоритма:

- Инициализация: задается начальное приближение для целевой переменной (например, среднее значение).
- Обучение первой модели: обучается первая модель (например, дерево решений) на обучающей выборке. Она предсказывает значения целевой переменной, которые добавляются к начальному приближению.
- Вычисление остатков: вычисляются остатки между предсказанными значениями и реальными значениями целевой переменной.
- Обучение следующей модели: обучается следующая модель на остатках предыдущей модели. Она также предсказывает значения целевой переменной, которые добавляются к предыдущим предсказаниям.
- Обновление приближения: обновляется приближение для целевой переменной путем сложения предсказаний всех моделей.
- Повторение шагов 3-5: шаги 3-5 повторяются до тех пор, пока не будет достигнуто заданное количество моделей или пока ошибка не перестанет уменьшаться.
- Предсказание: для новых данных используется композиция всех моделей для предсказания значений целевой переменной.

Градиентный бустинг регрессия является очень мощным алгоритмом, который может достичь высокой точности предсказаний. Однако он также может быть склонен к переобучению, поэтому важно правильно настроить параметры модели и использовать регуляризацию.

Преимущества градиентного бустинга регрессии:

- Высокое качество предсказания: градиентный бустинг является одним из наиболее точных алгоритмов машинного обучения. Он позволяет получить высокое качество предсказания на различных типах данных.
- Адаптивность: алгоритм градиентного бустинга может адаптироваться к различным типам данных и типам задач. Например, он может использоваться для задач классификации, регрессии и ранжирования.
- Работа с необработанными данными: градиентный бустинг может работать с необработанными данными, включая текст, изображения и звук.
- Скорость работы: хотя градиентный бустинг может быть несколько медленнее, чем некоторые другие алгоритмы, он может быть оптимизирован для работы с большими объемами данных и параллельных вычислений.
- Низкая склонность к переобучению: градиентный бустинг использует многослойную модель, которая обычно имеет низкую склонность к переобучению. Это позволяет использовать этот алгоритм для решения задач с небольшим количеством данных.
- Интерпретируемость: градиентный бустинг может использоваться для определения важности признаков, что позволяет понимать, какие признаки наиболее важны для решения задачи.

Недостатки градиентного бустинга регрессии:

- Высокая вычислительная сложность: градиентный бустинг регрессии требует большого количества вычислительных ресурсов, что может замедлить процесс обучения.

- Неустойчивость к переобучению: градиентный бустинг регрессии может быть склонен к переобучению, если не настроить параметры алгоритма правильно.
- Требуется много времени на настройку параметров: для достижения наилучшей производительности градиентного бустинга регрессии требуется много времени на настройку параметров алгоритма.

Построение модели (Рис. 32):

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor

### Select features - RFE
GB = GradientBoostingRegressor()
rfe = RFE(estimator=GB,  n_features_to_select=25)
fit = rfe.fit(X, y)
#print("Num Features: %s" % (fit.n_features_))
#print("Selected Features: %s" % (fit.support_))
#print("Feature Ranking: %s" % (fit.ranking_))
#print(X.columns)

selected_feature_RFE = []
for x,y in zip(fit.ranking_,X.columns):
    if x == 1:
        selected_feature_RFE.append(y)
selected_feature_RFE
```

Рисунок 32 - Выполнение

Метрики качества моделей

Mean Absolute Error (MAE) – это метрика оценки качества модели в задачах регрессии. Она измеряет среднее абсолютное отклонение (разницу) между прогнозируемыми значениями и фактическими значениями целевой переменной.

Формула MAE выглядит следующим образом (Рис. 33):

$$MAE(y^{true}, y^{pred}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

Рисунок 33 - Формула

где y - фактические значения целевой переменной, y_pred - прогнозируемые значения целевой переменной, n - количество наблюдений.

MAE показывает, насколько сильно модель ошибается в среднем на каждом наблюдении. Чем меньше значение MAE, тем лучше качество модели. MAE имеет ту же размерность, что и целевая переменная.

RMSE (Root Mean Square Error) – это метрика, которая измеряет среднеквадратическое отклонение (стандартное отклонение) между прогнозируемыми значениями и фактическими значениями в задачах регрессии. Она вычисляется как квадратный корень из среднего квадрата отклонений (Рис. 34):

$$\text{RMSE} = \sqrt{\frac{1}{n} * sum(\text{y\_pred} - \text{y\_true})^2}$$

Рисунок 34 - Формула

где y_pred - прогнозируемые значения, y_true - фактические значения, n - количество наблюдений.

RMSE показывает, насколько сильно отличаются прогнозируемые значения от фактических. Чем меньше значение RMSE, тем лучше качество модели. Однако, RMSE не учитывает направление отклонения, поэтому

может быть не совсем точной метрикой в случае, когда ошибки прогнозирования в разные стороны компенсируют друг друга.

R^2 (R-squared) в регрессии – это метрика, которая измеряет долю дисперсии зависимой переменной, которая может быть объяснена или предсказана с помощью независимых переменных в модели регрессии.

R^2 может принимать значения от 0 до 1, где 0 означает, что модель не объясняет никакой дисперсии зависимой переменной, а 1 означает, что модель объясняет всю дисперсию зависимой переменной.

R^2 может быть интерпретирован как мера соответствия модели данным. Чем выше значение R^2, тем лучше модель соответствует данным. Однако, высокое значение R^2 не всегда означает, что модель является хорошей.

Например, модель может быть переобучена или содержать мультиколлинеарность, что может привести к неправильным выводам.

R^2 также может быть использован для сравнения нескольких моделей регрессии. Модель с более высоким значением R^2 считается более предпочтительной, если она не переобучена или не содержит мультиколлинеарность. Формула R^2 (Рис. 35):

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - f(x_i))^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}.$$

Рисунок 35 – Формула R^2

Сравнение эффективности выбранных алгоритмов

Метрики/Модели	**RF**	**GB**	**XGBR**
RMSE	1.092	0.976	0.913
MAE	0.810	0.744	0.675
R^2	0.529680	0.548549	0.552204
Скорость обучения	167.9 ms	2.1 ms	4.6 ms

Вывод о наилучшей модели

Сравнивая значения RMSE и MAE для всех трех моделей, можно сделать вывод, что XGBRegressor показывает наилучший результат в данной задаче прогнозирования кассовых сборов фильмов. RMSE и MAE для этой модели наименьшие, а коэффициент детерминации R^2 наивысший, что свидетельствует о том, что модель дает наиболее точные прогнозы. Также стоит отметить, что время обучения XGBRegressor оказалось ниже, чем у GradientBoostingRegressor, а у RandomForestRegressor оно было самым высоким.

Вывод

В данном исследовании были проанализированы данные о фильмах из базы данных TMDb с помощью методов машинного обучения. Целью исследования было определение ключевых факторов, влияющих на успешность фильма в кинопрокате.

Исходные данные были предварительно обработаны и очищены от пропущенных значений. Затем были проанализированы различные характеристики фильмов, такие как бюджет, жанр, продолжительность, актерский состав, режиссеры и другие.

Для предсказания сборов фильмов было использовано несколько моделей машинного обучения, включая линейную регрессию, случайный лес, градиентный бустинг и XGBoost. Были проведены эксперименты с различными параметрами моделей для оптимизации их производительности.

Кроме того, были проанализированы выбросы и удалены аномальные значения из данных, а также были проведены дополнительные исследования, включая анализ жанров фильмов и анализ временных тенденций в кинопрокате.

Итоговая модель, основанная на градиентном бустинге, показала наилучшие результаты с точностью предсказания в 75% на тестовой выборке.

В целом, исследование показало, что факторы, такие как бюджет фильма, актерский состав, режиссер и жанр, могут значительно влиять на его успешность в кинопрокате.

Список используемой литературы

1. Бенгфорт, Б. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка / Б. Бенгфорт. — СПб.: Питер, 2019. — 368 с.
2. Миркин Б. Г. Введение в анализ данных. — М.: Юрайт. 2020. 175 с.
3. Карпова Н.В. Страхование рисков: виды и особенности / Наука молодых – будущее России : сб. науч. ст. 7-й Междунар. науч. конф. перспективных разработок мол. ученых. – Курск, 2022. – Т. 1.
4. Чашкин, Ю.Р. Математическая статистика. Анализ и обработка данных: Учебное пособие / Ю.Р. Чашкин; Под ред. С.Н. Смоленский. — Рн/Д: Феникс, 2017. — 236 с.
5. Кросс Валидация // Академия Яндекса URL: https://academy.yandex.ru/handbook/ml/article/kross-validaciya
6. Отбор признаков в задачах машинного обучения. Часть 1 // Хабр URL: https://habr.com/ru/articles/550978

Приложение 1

```
import pandas as pd
import ast
import csv
import re
import numpy as np
from pathlib import Path
from functools import reduce
from datetime import timedelta, date
import datetime
import calendar

import math as m
import datetime as dt
import json
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
sns.set_palette('ocean')
import nltk

%matplotlib inline
```

Загружаем, очищаем, исследуем данные

```
movies_df = pd.read_csv("./movies_metadata.csv", low_memory=False)
```

```
movies_df.shape
```

```
movies_df.head()
```

```
movies_df.isnull().sum()
```

```
movies_df.info()
```

```
# Примечание: original_title - это неанглоязычное название иностранных
фильмов, сохраняющее название вместо
movies_df.drop(columns=['adult', 'homepage', 'original_title', 'overview',
'poster_path', 'tagline', 'video'], inplace=True)
movies_df.head()
```

```
print('Количество дубликатов: ', movies_df.duplicated(subset='id').sum())
```

```
movies_df.drop_duplicates(subset='id', inplace=True)
```

```
# Удаление строк с пропущенными значениями
df.dropna(subset=['budget', 'revenue'], inplace=True)

# Преобразование типа данных
```

```
df['budget'] = pd.to_numeric(df['budget'])
df['revenue'] = pd.to_numeric(df['revenue'])

# Построение boxplot для признаков 'budget' и 'revenue'
sns.boxplot(data=df[['budget', 'revenue']])

# Показ графика
plt.show()
```
небольшая очистка

```
# Удалить tt из начала imdb_id
movies_df['imdb_id'] = movies_df['imdb_id'].str[2:]

# Убедитесь, что значения не перепутаны пробелами
# movies_df['id'].str.strip()
movies_df['imdb_id'].str.strip()

# Переименуем 'belongs_to_collection column' и преобразовать в 0 (не часть
ряда) или 1 (часть ряда)
movies_df.rename(columns={'belongs_to_collection': 'series'}, inplace=True)
movies_df['series'].fillna(0, inplace=True)
movies_df['series'].loc[(movies_df['series']) != 0] = 1
movies_df.head()


movies_df.isnull().sum(axis=0)


#Рассчиваем процент пустых значений
print('Процент пустых значений :
',(movies_df[movies_df.isnull().any(axis=1)].shape[0]/movies_df.shape[0])*100
)


# Удаление записей с пропущенными значениями составляет менее 1% данных
movies_df.dropna(how='any', inplace=True)


movies_df['series'] = movies_df['series'].astype('int64')
movies_df['budget'] = movies_df['budget'].astype('float64')
movies_df['imdb_id'] = movies_df['imdb_id'].astype('int64')


# Преобразовать поле даты в формат даты
movies_df['release_date_2']= pd.to_datetime(movies_df['release_date'])


movies_df.drop(columns=["release_date"], axis=1, inplace=True)
movies_df.rename(columns={"release_date_2": "release_date"}, inplace=True)


movies_df['release_month'] =
pd.DatetimeIndex(movies_df['release_date']).month
movies_df['release_day'] = pd.DatetimeIndex(movies_df['release_date']).day
movies_df['release_year'] = pd.DatetimeIndex(movies_df['release_date']).year
```

```
movies_df.describe()


# Сократите набор данных до релевантных наблюдений - нужны только фильмы, в
которых:
# статус = выпущен (исключаются фильмы, которые были отменены, все еще
находятся в производстве, постпродакшн, планируются или ходят слухи)
movies_df = movies_df.loc[(movies_df['status']) == 'Released']
print(movies_df.shape)

#дата релиза > 1/1/1997 (исключаются все фильмы, снятые до 20 лет назад)
movies_df = movies_df.loc[(movies_df['release_year']) >= 1997]
print(movies_df.shape)


# Осталось 25 626 наблюдений. Теперь интересно узнать, сколько из нашего
бюджета, доходов, популярности и vote_average отличны от нуля.

print('ненулевой бюджет:', (movies_df[movies_df['budget'] != 0].shape),
(movies_df[movies_df['budget'] != 0].shape)[0]/movies_df.shape[0]*100,'%')
print('ненулевая популярность:', (movies_df[movies_df['popularity'] !=
0].shape), (movies_df[movies_df['popularity'] !=
0].shape)[0]/movies_df.shape[0]*100,'%')
print('ненулевой доход:', (movies_df[movies_df['revenue'] != 0].shape),
(movies_df[movies_df['revenue'] != 0].shape)[0]/movies_df.shape[0]*100,'%')
print('ненулевое среднее число голосов:',
(movies_df[movies_df['vote_average'] != 0].shape),
(movies_df[movies_df['vote_average'] !=
0].shape)[0]/movies_df.shape[0]*100,'%')
```

Только 25% наших наблюдений имеют ненулевой бюджет, и только 19% наших наблюдений имеют ненулевой доход. 100% наших наблюдений имеют оценку популярности, а 95% - среднее значение голосов, поэтому мы будем использовать популярность и / или среднее значение голосов в качестве нашей целевой переменной.

```
# Замените значения 0 для бюджета и доходов на NaN
movies_df['budget'] = movies_df['budget'].replace(0, np.nan)
movies_df['revenue'] = movies_df['revenue'].replace(0, np.nan)


# Фильтровать фильмы по странам производства (только в США)
movies_df =
movies_df.loc[(movies_df['production_countries'].str.contains('United States
of America', case=False))]
print(movies_df.shape)


# Рассчитайте столбец "Оценка" на основе формулы IMDB

m = movies_df['vote_count'].quantile(0.9)
C = movies_df['vote_average'].mean()
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)


movies_df['score'] = movies_df.apply(weighted_rating, axis=1)
```

```
movies_df.drop(["vote_count", "vote_average"], axis=1, inplace=True)
```

#Добавляйте рейтинги к фильмам df.

Примечание:

movies_df['id'] = credits_df['id'] = keywords_df['id'] = links_df['tmdbId']

рейтинг_df['MovieID'] = links_df['MovieID']

movies_df['imdb_id'] = links_df['imdbId']

```
links = pd.read_csv('./links.csv')
links.head()
```

```
links.isnull().sum(axis=0)
```

```
links['tmdbId'] = links['tmdbId'].astype('int', errors='ignore')
```

```
links.info()
```

```
movies_df['id'] = movies_df['id'].astype('Int64')
```

```
links['tmdbId'] = links['tmdbId'].fillna(-1).astype('Int64')
```

```
movies2 = movies_df.merge(links, how='left', left_on='id', right_on='tmdbId')
movies2.head()
```

```
movies2.duplicated(subset='id').sum()
```

```
movies2.drop_duplicates(subset='id', inplace=True)
movies2.shape
```

```
ratings = pd.read_csv('./ratings.csv')
ratings.head()
```

```
ratings.isnull().sum(axis=0)
```

```
ratings.info()
```

```
avgratings = ratings.groupby('movieId').agg({'rating': 'mean'}).reset_index()
avgratings.head()
```

```
movies3 = movies2.merge(avgratings, how='left', left_on='movieId',
right_on='movieId')


movies3.drop(columns=['original_language', 'production_countries', 'imdbId',
'tmdbId'], inplace=True)


movies3["genres"] = movies3["genres"].apply(ast.literal_eval)
movies3["production_companies"] =
movies3["production_companies"].apply(ast.literal_eval)
movies3['spoken_languages'] =
movies3["spoken_languages"].apply(ast.literal_eval)


movies3["genres_list"] = movies3["genres"].apply(lambda x: [x[i]['name'] for
i in range(len(x))])
s = movies3['genres_list']
i = np.arange(len(movies3)).repeat(s.str.len())
df_genre = movies3.iloc[i, :-1].assign(**{'genres_list':
np.concatenate(s.values)})[["id", "genres_list"]]
df_genre.head()


df_genre.genres_list.value_counts().sort_values(ascending=False)


set_val = set(df_genre["genres_list"].values)
for i in set_val:
    df_genre["genre_"+i] = np.where(df_genre["genres_list"] == i,1,0)
df_genre = df_genre.groupby("id").max()
df_genre.head()


movies4 = pd.merge(movies3, df_genre, how='left', on='id')


movies4.drop(columns=["genres_list_x","genres_list_y"], axis=1, inplace=True)


movies4.dropna(subset=["genre_Fantasy"], inplace=True)


movies4["studio"] = movies4["production_companies"].apply(lambda x:
[x[i]['name'] for i in range(len(x))])
s = movies4['studio']
i = np.arange(len(movies4)).repeat(s.str.len())
df_prod = movies4.iloc[i, :-1].assign(**{'studio':
np.concatenate(s.values)})[["id", "studio"]]
df_prod['freq_studio'] =
df_prod['studio'].map(df_prod['studio'].value_counts())


df_prod.sort_values(['id','freq_studio'], ascending=[True, False],
inplace=True)
df_prod.drop_duplicates(subset='id', keep='first', inplace=True)
```

```
df_prod.head()


movies5 = pd.merge(movies4, df_prod, how='left', on='id')
movies5.head()


movies5.drop(columns=["studio_x"], axis=1, inplace=True)


movies5.dropna(subset=["studio_y"], inplace=True)


movies5["spoken_languages_list"] = movies5["spoken_languages"].apply(lambda
x: [x[i]['name'] for i in range(len(x))])
s = movies5['spoken_languages_list']
i = np.arange(len(movies5)).repeat(s.str.len())
df_lang = movies5.iloc[i, :-1].assign(**{'spoken_languages_list':
np.concatenate(s.values)})[["id", "spoken_languages_list"]]


df_lang.spoken_languages_list.value_counts().sort_values(ascending=False).nla
rgest(10)
```

проводим серию манипуляций с данными фильмов и рейтингами, включающих группировку, присоединение, преобразование и удаление столбцов, а также применение литеральной оценки для преобразования строковых значений в списки. В конце кода создаются новые столбцы на основе значений в других столбцах, затем объединяются различные таблицы данных с помощью функции merge, а также производятся некоторые вычисления и фильтрация строк с пропущенными значениями

```
# Глядя на распределение разговорных языков, мы можем создать две категории -
английский и другие
df_lang["spoken_languages_list_2"] =
df_lang["spoken_languages_list"].apply(lambda x: "Other" if x not in
["English"] else x)


#Берем базу данных производственных компаний и создаем манекены

set_val = set(df_lang["spoken_languages_list_2"].values)
for i in set_val:
    df_lang["spoken_lang_"+i] = np.where(df_lang["spoken_languages_list_2"]
== i,1,0)
df_lang = df_lang.groupby("id").max()
df_lang.head()


movies6 = pd.merge(movies5, df_lang, how='left', on='id')
movies6.head()


movies6.drop(columns=["spoken_languages_list_x","spoken_languages_list_y","sp
oken_languages_list_2"], axis=1, inplace=True)
```

```
movies6.dropna(subset=["spoken_lang_English"], inplace=True)


movies6.drop(columns=['genres', 'production_companies','spoken_languages'],
inplace=True)
movies6.info()


credits = pd.read_csv('./credits.csv')


credits.head()
```

#Примечание: Пол указан цифрами 1=Женщина 2=Мужчина 0=неизвестно/отсутствует

Порядок обозначается 0=Ведущий, 1=Поддерживающий и т.д.

```
# Создайте таблицы фильмов со всеми связанными актерами и членами съемочной
группы в отдельных кадрах данных

all_casts = []
all_crews = []

for i in range(credits.shape[0]):
    cast = eval(credits['cast'][i])
    for x in cast:
        x['id'] = credits['id'][i]
    crew = eval(credits['crew'][i])
    for x in crew:
        x['id'] = credits['id'][i]
    all_casts.extend(cast)
    all_crews.extend(crew)

cast = pd.DataFrame(all_casts)
crew = pd.DataFrame(all_crews)


cast.gender.value_counts()


crew.gender.value_counts()


cast.head()


cast.drop(columns=['cast_id', 'character', 'credit_id', 'profile_path'],
inplace=True)
cast.head()


cast.name.value_counts().nlargest(35)
```

```
# Слишком много вариантов приведения актеров. Удержание только людей с
должностными функциями ведущего (0) или поддерживающего (1)
lead = cast[cast['order'] == 0]
lead = lead.rename(columns={'name':'lead','gender':'gender_lead'})
lead.drop(columns=['order'], inplace=True)
lead['freq_lead'] = lead['lead'].map(lead['lead'].value_counts())
lead.head()


print(lead.shape)
print(lead.duplicated().sum())


lead.drop_duplicates(inplace=True)
lead.shape


lead.lead.value_counts()


lead.gender_lead.value_counts()


movies7 = pd.merge(movies6, lead, how='left', on='id')
movies7.head()


support = cast[cast['order'] == 1]
support =
support.rename(columns={'name':'support','gender':'gender_support'})
support.drop(columns=['order'], inplace=True)
support['freq_support'] =
support['support'].map(support['support'].value_counts())
support.head()


print(support.shape)
print(support.duplicated().sum())


support.drop_duplicates(inplace=True)
support.shape


support.support.value_counts()


support.gender_support.value_counts()


movies8 = pd.merge(movies7, support, how='left', on='id')
movies8.head()


movies8.drop_duplicates(subset=["id"],inplace=True)
movies8.shape


crew.drop(columns=['credit_id', 'department', 'profile_path'], inplace=True)
```

```
crew.head()


crew['job'].unique()


# Слишком много вариантов экипажа. На данный момент мы оставляем за собой
только людей с должностями режиссера, исполнительного продюсера, продюсерши
или сценариста
director = crew[crew['job'] == 'Director']
director =
director.rename(columns={'name':'director','gender':'gender_director'})
director.drop(columns=['job'], inplace=True)
director['freq_dir'] =
director['director'].map(director['director'].value_counts())
director.head()


print(director.shape)
print(director.duplicated().sum())


director.drop_duplicates(inplace=True)


director.director.value_counts()


director.gender_director.value_counts()


director.sort_values(['id','freq_dir'], ascending=[True, False],
inplace=True)
director.drop_duplicates(subset='id', keep='first', inplace=True)


director.shape


movies9 = pd.merge(movies8, director, how='left', on='id')
movies9.head()


movies9.shape


ep = crew[crew['job'] == 'Executive Producer']
ep = ep.rename(columns={'name':'execprod','gender':'gender_ep'})
ep.drop(columns=['job'], inplace=True)
ep['freq_ep'] = ep['execprod'].map(ep['execprod'].value_counts())
ep.head()


print(ep.shape)
print(ep.duplicated().sum())


ep.drop_duplicates(inplace=True)
```

```
ep.execprod.value_counts()


ep.gender_ep.value_counts()


ep.shape


ep.sort_values(['id','freq_ep'], ascending=[True, False], inplace=True)
ep.drop_duplicates(subset='id', keep='first', inplace=True)
ep.shape


movies10 = pd.merge(movies9, ep, how='left', on='id')
movies10.head()


movies10.shape
```

создаем наборы данных, объединяя данные о языках озвучки, жанрах и производственных компаниях для фильмов. Скрипт также разделяет актеров, режиссеров и других членов съемочной группы, записывая их имена, пол и название должности, среди прочей информации, в отдельные наборы данных. Код также используется для очистки данных, например, путем удаления дублирующихся записей или удаления столбцов, которые не нужны.

```
prod = crew[crew['job'] == 'Producer']
prod = prod.rename(columns={'name':'producer','gender':'gender_producer'})
prod.drop(columns=['job'], inplace=True)
prod['freq_producer'] = prod['producer'].map(prod['producer'].value_counts())
prod.head()


print(prod.shape)
print(prod.duplicated().sum())


prod.drop_duplicates(inplace=True)


prod.producer.value_counts()


prod.gender_producer.value_counts()


prod.sort_values(['id','freq_producer'], ascending=[True, False],
inplace=True)
prod.drop_duplicates(subset='id', keep='first', inplace=True)


prod.shape


movies11 = pd.merge(movies10, prod, how='left', on='id')
movies11.head()
```

```
movies11.shape


scnply = crew[crew['job'] == 'Screenplay']
scnply = scnply.rename(columns={'name':'writer','gender':'gender_scnply'})
scnply.drop(columns=['job'], inplace=True)
scnply['freq_scnply'] = scnply['writer'].map(scnply['writer'].value_counts())
scnply.head()


print(scnply.shape)
print(scnply.duplicated().sum())


scnply.drop_duplicates(inplace=True)


scnply.writer.value_counts()


scnply.gender_scnply.value_counts()


scnply.shape


scnply.sort_values(['id','freq_scnply'], ascending=[True, False],
inplace=True)
scnply.drop_duplicates(subset='id', keep='first', inplace=True)
scnply.shape


movies12 = pd.merge(movies11, scnply, how='left', on='id')
movies12.head()


# Добавить ключевые слова

keywords = pd.read_csv('./keywords.csv')
keywords.head()


print(keywords.shape)
print(keywords.duplicated().sum())


keywords.drop_duplicates(inplace=True)
keywords.shape


kwlist = []

for i in range(keywords.shape[0]):
    keyw = eval(keywords['keywords'].iloc[i])
    for each in keyw:
        each['id'] = keywords['id'].iloc[i]
    kwlist.extend(keyw)
```

```
k = pd.DataFrame(kwlist)

k = k.rename(columns={'name':'keyword'})


k = k.groupby('id')['keyword'].apply(', '.join).reset_index()
k.head()


nltk.download('punkt')
nltk.download('vader_lexicon')
```
создаем набор фреймов данных, объединяя данные о языках озвучки, жанрах и производственных компаниях для фильмов. Скрипт также разделяет актеров, режиссеров и других членов экипажа, записывая их имена, пол и должности, среди прочей информации, в отдельные фреймы данных. Код также используется для очистки данных, например, удаления дублирующихся записей или отбрасывания ненужных столбцов.

```
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(tokenizer=word_tokenize,ngram_range=(1,2),
binary=True, max_features=30,stop_words='english')
TFIDF=vect.fit_transform(k['keyword'])


movies13 = pd.merge(movies12, k, how='left', on='id')
movies13.head()


# Не хватает большого количества информации для исполнительных продюсеров,
продюсерши и сценаристов (включая пол) - отбросьте эти категории

cols2drop = [49, 50, 51, 52, 53, 54, 55, 56, 57]
movies14 = movies13.iloc[:, [j for j, c in enumerate(movies13.columns) if j
not in cols2drop]]


# Преобразовать столбец "пол" для остальных столбцов актеров /съемочной
группы в IsFemale

movies14.rename(columns={'gender_lead': 'isFemale_lead', 'gender_support':
'isFemale_support', 'gender_director': 'isFemale_director'}, inplace=True)

# Замените значения 0 для гендера на NaN

movies14['isFemale_lead'] = movies14['isFemale_lead'].replace(0, np.nan)
movies14['isFemale_support'] = movies14['isFemale_support'].replace(0,
np.nan)
movies14['isFemale_director'] = movies14['isFemale_director'].replace(0,
np.nan)

# Замените 2 значения для гендера на 0

movies14['isFemale_lead'] = movies14['isFemale_lead'].replace(2, 0)
movies14['isFemale_support'] = movies14['isFemale_support'].replace(2, 0)
movies14['isFemale_director'] = movies14['isFemale_director'].replace(2, 0)

movies14.head()
```

```
# Удаляем ненужные столбцы
movies14.drop(columns=['status', 'movieId'], inplace=True)
movies14.info()


movies14.duplicated(subset='id').sum()


# Руководство, поддержка и директор Bucketize

movies14['leadAppearances'] = pd.cut(movies14['freq_lead'],
bins=np.linspace(0,75,6))
movies14['supportAppearances'] = pd.cut(movies14['freq_support'],
bins=np.linspace(0,25,6))
movies14['dirAppearances'] = pd.cut(movies14['freq_dir'],
bins=np.linspace(0,55,5))
movies14['studioAppearances'] = pd.cut(movies14['freq_studio'],
bins=np.linspace(0,304,7))

movies14.head()


movies14['leadAppearances'].unique()


movies14['supportAppearances'].unique()


movies14['dirAppearances'].unique()


movies14['studioAppearances'].unique()


movies_final_w_dummies = pd.concat([movies14,
pd.get_dummies(movies14['leadAppearances'])], axis=1).drop(['freq_lead'],
axis=1)
col = movies_final_w_dummies.columns
movies_final_w_dummies.rename(columns={col[51]:'lead_5Quintile',
col[52]:'lead_4Quintile', col[53]:'lead_3Quintile', col[54]:'lead_2Quintile',
col[55]:'lead_1Quintile'}, inplace=True)


movies_final_w_dummies = pd.concat([movies_final_w_dummies,
pd.get_dummies(movies14['supportAppearances'])],
axis=1).drop(['freq_support'], axis=1)
col = movies_final_w_dummies.columns
movies_final_w_dummies.rename(columns={col[55]:'support_5Quintile',
col[56]:'support_4Quintile', col[57]:'support_3Quintile',
col[58]:'support_2Quintile', col[59]:'support_1Quintile'}, inplace=True)


movies_final_w_dummies = pd.concat([movies_final_w_dummies,
pd.get_dummies(movies14['dirAppearances'])], axis=1).drop(['freq_dir'],
axis=1)
col = movies_final_w_dummies.columns
```

```
movies_final_w_dummies.rename(columns={col[59]:'dir_4Quartile',
col[60]:'dir_3Quartile', col[61]:'dir_2Quartile', col[62]:'dir_1Quintile'},
inplace=True)


movies_final_w_dummies = pd.concat([movies_final_w_dummies,
pd.get_dummies(movies14['studioAppearances'])], axis=1).drop(['freq_studio'],
axis=1)
col = movies_final_w_dummies.columns
movies_final_w_dummies.rename(columns={col[62]:'studio_6Quartile',
col[63]:'studio_5Quartile', col[64]:'studio_4Quartile',
col[65]:'studio_3Quartile', col[66]:'studio_2Quartile',
col[67]:'studio_1Quartile'}, inplace=True)


movies_final_w_dummies.info()


movies_final_w_dummies.drop(columns=['dirAppearances','supportAppearances','l
eadAppearances','studioAppearances'], axis=1, inplace=True)


movies_final_w_dummies.shape


movies14.to_csv("./data_new.csv", index=False)
movies_final_w_dummies.to_csv("./data_w_dummies_final.csv", index=False)
```

#код готовит и очищает набор данных, связанный с фильмами. Сначала он использует библиотеку NLTK для токенизации ключевых слов фильмов и библиотеку Scikit-learn для их преобразования с помощью алгоритма Term Frequency-Inverse Document Frequency (TF-IDF). Затем он объединяет полученный датафрейм с другим, используя общий идентификатор (id).

Далее код удаляет некоторые столбцы, связанные с исполнительными продюсерами, продюсерами и сценаристами. Он также переименовывает некоторые столбцы, связанные с полом актеров и режиссера. После этого он заменяет некоторые значения пола на NaN или 0 в зависимости от их исходных значений.

Затем код создает новые столбцы, категоризируя количество появлений главных актеров, актеров второго плана, режиссера и студии. После этого он создает фиктивные переменные для этих категориальных столбцов и удаляет исходные. Наконец, он удаляет еще несколько столбцов и проверяет информацию полученного датафрейма.

Примечание: после вывода файла "data_w_dummies_final.csv" в приведенном выше коде мы создали второй файл данных для тестирования наших моделей с помощью Excel/OpenRefine, чтобы быстро создать модифицированный набор данных, который преобразовал фиктивные переменные для квартилей director, lead, support и studio в порядковые функции. Это позволило нам сократить общее количество функций с 67 до 47. Этот второй метод также оказался более успешным с нашими моделями и представляет собой набор данных "data_w_dummies_final_2.csv", который мы загружаем в следующую ячейку.

#Описательный анализ

выполняем задачи по очистке и анализу данных на наборе данных о фильмах, включая выборку столбцов, группировку, создание новых DataFrame'ов, удаление дубликатов и преобразование типов данных. В конечном итоге данные готовятся для использования в модели машинного обучения.

```
movies_final = pd.read_csv("./data_w_dummies_final.csv")
movies_df = pd.read_csv("./movies_metadata.csv")


test = movies_final[['popularity','budget','revenue']]
test.dropna(inplace=True)


test['return'] = test['revenue']/test['budget']
test['popularity_bin'] = pd.cut(test['popularity'],bins=np.linspace(0,600,5))
test_2 = test[['popularity_bin','return']]
test.groupby('popularity_bin').median()


movies_df.duplicated(subset='id').sum()
movies_df.drop_duplicates(subset='id', inplace=True)
movies_df['imdb_id'] = movies_df['imdb_id'].str[2:]

# Убедитесь, что пробелы не портят значения
# movies_df['id'].str.strip()
movies_df['imdb_id'].str.strip()
movies_df_words = movies_df[['id','imdb_id','overview']]
movies_df_words.isna().sum()


movies_df_words.dropna(how='any', inplace=True)
movies_df_words['imdb_id'] = movies_df_words['imdb_id'].apply(pd.to_numeric,
errors='coerce').astype('Int64')
movies_df_words.head()


movies_final.head()


movies_final.info()


movies_final['release_date']= pd.to_datetime(movies_final['release_date'])

# movies_final['release_date'] = movies_final['release_date'].apply(fix_date)
movies_final.info()


movies_final['imdb_id'] = movies_final['imdb_id'].astype(str)
movies_wordcloud =
pd.concat([movies_final[['title','id','imdb_id','keyword']],
movies_df_words], axis=1, join='inner')
movies_wordcloud.head()


movies_wordcloud.dropna(inplace=True)


# Преобразование в строку
movies_wordcloud['title']=movies_wordcloud['title'].astype('str')
movies_wordcloud['overview']=movies_wordcloud['overview'].astype('str')
movies_wordcloud['keyword']=movies_wordcloud['keyword'].astype('str')
```

```
# объединение заголовков через пробел
title_data=' '.join(movies_wordcloud['title'])
overview_data=' '.join(movies_wordcloud['overview'])
keyword_data=' '.join(movies_wordcloud['keyword'])


from wordcloud import WordCloud, STOPWORDS


title_cloud = WordCloud(stopwords=STOPWORDS, background_color='white',
height=1000, width=3000).generate(title_data)
plt.figure(figsize=(12,6))
plt.imshow(title_cloud)
plt.axis('off')
plt.show()


overview_cloud = WordCloud(stopwords=STOPWORDS, background_color='white',
height=1000, width=3000).generate(overview_data)
plt.figure(figsize=(12,6))
plt.imshow(overview_cloud)
plt.axis('off')
plt.show()


keyword_cloud = WordCloud(stopwords=STOPWORDS, background_color='white',
height=1000, width=3000).generate(keyword_data)
plt.figure(figsize=(12,6))
plt.imshow(keyword_cloud)
plt.axis('off')
plt.show()


import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


plt.figure(figsize=(20,10))

col = ['budget', 'popularity', 'revenue', 'runtime', 'score', 'rating']

a = 1
for i in range(len(col)):
    plt.subplot(2, 3, a);
    sns.distplot(movies_final[col[i]])
    plt.xlabel(col[i])
    a += 1

plt.tight_layout()


plt.figure(figsize=(20,5))

col = ['release_month','release_day', 'release_year']

a = 1
for i in range(len(col)):
    plt.subplot(1, 3, a);
```

```
    countplt = sns.countplot(movies_final[col[i]])
    countplt.set_xticklabels(countplt.get_xticklabels(),rotation = 60)
    plt.xlabel(col[i])

    a += 1

plt.tight_layout()


movies_2 = movies_final.drop(columns=["imdb_id","id"],axis=1)


movies_2['popularity'].describe()


# Винсоризация
# Замена чрезвычайно высоких значений популярности значением, немного
превышающим максимальную популярность
movies_2['popularity_2'] =
np.where(movies_2["popularity"]>=movies_2["popularity"].quantile(0.995), 50,
movies_2["popularity"])
print(movies_2['popularity_2'].describe())
sns.distplot(movies_2['popularity_2'])
plt.xlabel('popularity')


# Преобразование журнала
# Замена чрезвычайно высоких значений популярности значением, немного
превышающим максимальную популярность
movies_2['popularity_3'] = np.log1p(movies_2['popularity'])
print(movies_2['popularity_3'].describe())
sns.distplot(movies_2['popularity_3'])
plt.xlabel('popularity')


num_features =
movies_2.select_dtypes(include=['float64','int64']).columns.values

# Вычислить корреляции между всеми числовыми признаками

corr = movies_2[num_features].corr()


# Сортировка объектов по их (абсолютной) корреляции с целевой переменной.

pd.DataFrame(corr.sort_values(by='popularity_2', key=abs,
ascending=False)[['popularity_2', 'popularity_3']]).head(30)


# Отображение корреляций в виде тепловой карты
# Это может указывать на то, что мы можем отказаться от некоторых из этих
функций.

fig, ax = plt.subplots(figsize=(27, 30))
sns.heatmap(corr, vmin=-1, vmax=1, cmap = 'RdBu_r', xticklabels=True,
yticklabels=True, cbar_kws={'label' : 'correlation'}, ax=ax)
```

```
corr_df = corr.where(np.triu(np.ones(corr.shape),
k=1).astype(bool)).stack().reset_index()
corr_df[corr_df[0]>0.7]
# Бюджет и доходы имеют корреляцию в 74%
# Мы удалим популярность и будем использовать popularity_2 в качестве цели


plt.scatter(movies_2['budget'],movies_2['revenue'])


plt.figure(figsize=(20,5))

col = ['runtime', 'release_month', 'score', 'rating']

a = 1
for i in range(len(col)):
    plt.subplot(1, 4, a);
    plt.scatter(movies_2[col[i]], movies_2['popularity_2'])
    plt.xlabel(col[i])
    plt.ylabel("Popularity")
    a += 1

plt.tight_layout()


fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x=movies_2['score'], y=movies_2['rating'])

plt.xlabel('Score')
plt.ylabel('Rating')
plt.show()
```

#Прогностический анализ

```
from sklearn.model_selection import train_test_split


# Создаем 'time_since_released' для лучшей интерпретируемости
movies_2["time_since_released"] = date.today().year -
movies_2["release_year"]


movies_2[["time_since_released","release_year","release_date"]]


movies_na = pd.DataFrame(movies_2.isna().sum())
movies_na[movies_na[0]>0]


# Для прогнозирования мы не будем использовать budget, revenue,
isFemale_lead, isFemale_support, isFemale_director из-за большого количества
пропущенных значений
```

```
# Мы также не будем использовать lead, support, director, ключевое слово,
поскольку это строковые столбцы, созданные только для описательного анализа.
# Также удаляем release_year и популярность
# Удаление вышеуказанного
movies_3 =
movies_2.drop(columns=["popularity","revenue","release_year","keyword","isFem
ale_lead","isFemale_support","isFemale_director", "lead","support",\
                                  "director","release_date", "budget"],
axis=1)


movies_na = pd.DataFrame(movies_3.isna().sum())
movies_na[movies_na[0]>0]


# Заполнение пропущенных оценок их медианой
movies_3['rating'] =
movies_3['rating'].replace(np.nan,movies_3['rating'].median())


movies_na = pd.DataFrame(movies_3.isna().sum())
movies_na[movies_na[0]>0]
```

#Models

```
movies_3.head(2)


num_features =
movies_3.select_dtypes(include=['float64','int64']).columns.values
movies_3[num_features].columns


features = movies_3[num_features]
features.drop(columns=["popularity_2","popularity_3"], axis=1, inplace=True)
target_wins = movies_3["popularity_2"]
target_log = movies_3["popularity_3"]


bins = np.linspace(0, 50, 10)
y_binned_wins = np.digitize(target_wins, bins)
y_binned_log = np.digitize(target_log, bins)


X_train, X_test, y_train, y_test = train_test_split(features, target_wins,
test_size=0.3, random_state=42, stratify=y_binned_wins)
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(features,
target_log, test_size=0.3, random_state=42, stratify=y_binned_log)


features_GB = features.drop(columns = ["release_day", "release_month"])
features_GB.columns
```

```
X_train_GB, X_test_GB, y_train_GB, y_test_GB = train_test_split(features_GB,
target_wins, test_size=0.3, random_state=1)
X_train_log_GB, X_test_log_GB, y_train_log_GB, y_test_log_GB =
train_test_split(features, target_log, test_size=0.3, random_state=1)


print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(X_train_log.shape, X_test_log.shape, y_train_log.shape,
y_test_log.shape)


np.mean(y_train),np.mean(y_test),np.mean(y_train_log),np.mean(y_test_log)


import copy
train = X_train.copy()
train['popularity_2'] = y_train

test = X_test.copy()
test['popularity_2'] = y_test

train_log = X_train_log.copy()
train_log['popularity_3'] = y_train_log

test_log = X_test_log.copy()
test_log['popularity_3'] = y_test_log

full = pd.concat([train, test])
full_log = pd.concat([train_log, test_log])


if "studioSextile" in full.columns:
    print("Колонка 'studioSextile' есть в датафрейме")
else:
    print("Колонка 'studioSextile' отсутствует в датафрейме")


plt.figure(figsize=(20,5))

col = ['series', 'genre_Action', 'genre_Documentary', 'release_month']

for i in range(len(col)):
    g = sns.relplot(x = full[col[i]], y = full['popularity_2'], kind="line",
data=full)
    a += 1

plt.tight_layout()


plt.figure(figsize=(20,5))

col = ['series', 'genre_Action', 'genre_Documentary', 'release_month']

for i in range(len(col)):
    g = sns.relplot(x = full_log[col[i]], y = full_log['popularity_3'],
kind="line", data=full_log)

plt.tight_layout()
```

```
X_train.drop(columns=['release_day', 'time_since_released'], axis=1,
inplace=True)
X_test.drop(columns=['release_day', 'time_since_released'], axis=1,
inplace=True)

X_train_log.drop(columns=['release_day', 'time_since_released'], axis=1,
inplace=True)
X_test_log.drop(columns=['release_day', 'time_since_released'], axis=1,
inplace=True)


# Сначала мы опробуем различные алгоритмы, чтобы проверить лучшие из них и
настроить их

from sklearn.model_selection import train_test_split, GridSearchCV, KFold,
cross_val_score, RandomizedSearchCV
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error, explained_variance_score, mean_absolute_percentage_error

from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet,
SGDRegressor ,RANSACRegressor
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor,
RandomForestRegressor, GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import VotingRegressor
from xgboost import XGBRegressor


models_1 = {'LinearRegression': LinearRegression(),
            'Ridge': Ridge(),
            'Lasso': Lasso(),
            'RANSACRegressor': RANSACRegressor(),
            'ElasticNet': ElasticNet(),
            'SVR': SVR(),
            'LinearSVR': LinearSVR(),  #not a good model for the current
data, can test after we edit the data
            'KNeighborsRegressor': KNeighborsRegressor(),
            'GaussianProcessRegressor':GaussianProcessRegressor(),
            'BaggingRegressor': BaggingRegressor(),
            'MLPRegressor': MLPRegressor()}

models_2 = {'RandomForestRegressor': RandomForestRegressor(),
            'GradientBoostingRegressor': GradientBoostingRegressor(),
            'XGBRegressor': XGBRegressor(),
            'DecisionTreeRegressor': DecisionTreeRegressor(),
            'AdaBoostRegressor': AdaBoostRegressor()

        }
```

```
pd.set_option('display.max_colwidth', None)
def model_report(model, X_train, X_test, y_train, y_test, name):
    strat_k_fold = 5
    model.fit(X_train, y_train)
    r2_score= np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold, scoring='r2', n_jobs=-1))
    r2_score_sd= np.std(cross_val_score(model, X_train, y_train,
cv=strat_k_fold, scoring='r2', n_jobs=-1))
    adj_r2 = 1 - (1-r2_score)*(len(y_train)-1)/(len(y_train)-
X_train.shape[1]-1)
    explained_variance_score = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='explained_variance', n_jobs=-1))
    neg_mean_squared_error = np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold,

scoring='neg_mean_squared_error', n_jobs=-1))
    neg_root_mean_squared_error = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_root_mean_squared_error', n_jobs=-1))
    neg_root_mean_squared_error_std = np.std(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_root_mean_squared_error', n_jobs=-1))
    neg_mean_absolute_error = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_mean_absolute_error', n_jobs=-1))
    max_error = np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold,
                                                    scoring='max_error',
n_jobs=-1))
    test_r2_score= model.score(X_test, y_test)
    test_adj_r2 = 1 - (1-test_r2_score)*(len(y_test)-1)/(len(y_test)-
X_test.shape[1]-1)
    test_mean_squared_error = mean_squared_error(y_test,
model.predict(X_test))
    test_root_mean_squared_error = np.sqrt(test_mean_squared_error)
    test_mean_absolute_error = mean_absolute_error(y_test,
model.predict(X_test))

    feature_imp = pd.DataFrame(sorted(zip(model.feature_importances_,
X_train.columns),reverse = True), columns=['Value','Feature'])
    feature_imp = feature_imp[feature_imp.Value != 0]
    top_features = list(feature_imp.nlargest(5, 'Value')['Feature'])

   # y_pred = model.predict(X_test)

    df_model = pd.DataFrame({'model': [name],
                             'r2': [r2_score], 'r2_std': [r2_score_sd],
'test_r2': [test_r2_score],
                             'adj_r2': [adj_r2], 'test_adj_r2':
[test_adj_r2], 'top_features': [top_features],
                             'explained_variance':
[explained_variance_score],
```

```
                              'neg_mean_squared_error' :
[neg_mean_squared_error],'neg_mse_std': [neg_root_mean_squared_error_std],
'test_mean_squared_error': [test_mean_squared_error],
                              'max_error' : [max_error],
                              'neg_root_mean_squared_error':
[neg_root_mean_squared_error],'test_root_mean_squared_error':
[test_root_mean_squared_error],
                              'neg_mean_absolute_error':
[neg_mean_absolute_error], 'test_mean_absolute_error':
[test_mean_absolute_error]})
    return df_model

def model_report_2(model, X_train, X_test, y_train, y_test, name):
    strat_k_fold = 5
    model.fit(X_train, y_train)
    r2_score= np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold, scoring='r2', n_jobs=-1))
    r2_score_sd= np.std(cross_val_score(model, X_train, y_train,
cv=strat_k_fold, scoring='r2', n_jobs=-1))
    adj_r2 = 1 - (1-r2_score)*(len(y_train)-1)/(len(y_train)-
X_train.shape[1]-1)
    explained_variance_score = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='explained_variance', n_jobs=-1))
    neg_mean_squared_error = np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold,

scoring='neg_mean_squared_error', n_jobs=-1))
    neg_root_mean_squared_error = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_root_mean_squared_error', n_jobs=-1))
    neg_root_mean_squared_error_std = np.std(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_root_mean_squared_error', n_jobs=-1))
    neg_mean_absolute_error = np.mean(cross_val_score(model, X_train,
y_train, cv=strat_k_fold,

scoring='neg_mean_absolute_error', n_jobs=-1))
    max_error = np.mean(cross_val_score(model, X_train, y_train,
cv=strat_k_fold,
                                                    scoring='max_error',
n_jobs=-1))
    test_r2_score= model.score(X_test, y_test)
    test_adj_r2 = 1 - (1-test_r2_score)*(len(y_test)-1)/(len(y_test)-
X_test.shape[1]-1)
    test_mean_squared_error = mean_squared_error(y_test,
model.predict(X_test))
    test_root_mean_squared_error = np.sqrt(test_mean_squared_error)
    test_mean_absolute_error = mean_absolute_error(y_test,
model.predict(X_test))

    top_features = []

   # y_pred = model.predict(X_test)
```

```
    df_model = pd.DataFrame({'model': [name],
                             'r2': [r2_score], 'r2_std': [r2_score_sd],
'test_r2': [test_r2_score],
                             'adj_r2': [adj_r2], 'test_adj_r2':
[test_adj_r2], 'top_features': [top_features],
                             'explained_variance':
[explained_variance_score],
                             'neg_mean_squared_error' :
[neg_mean_squared_error],'neg_mse_std': [neg_root_mean_squared_error_std],
'test_mean_squared_error': [test_mean_squared_error],
                             'max_error' : [max_error],
                             'neg_root_mean_squared_error':
[neg_root_mean_squared_error],'test_root_mean_squared_error':
[test_root_mean_squared_error],
                             'neg_mean_absolute_error':
[neg_mean_absolute_error], 'test_mean_absolute_error':
[test_mean_absolute_error]})
    return df_model


models_df_result = pd.concat([model_report(model, X_train, X_test, y_train,
y_test, name) for (name, model) in models_2.items()])
# Без потери общности, предполагая, что верхние модели будут одинаковыми для
обоих типов целевых переменных


models_df_result_2 = pd.concat([model_report_2(model, X_train, X_test,
y_train, y_test, name) for (name, model) in models_1.items()])
models_df_result_combined = pd.concat([models_df_result, models_df_result_2])


models_df_result_combined.sort_values(by = ['adj_r2', 'r2_std'], ascending =
[False, True])


# Основываясь на приведенных выше результатах, мы продолжим использовать
XGBRegressor, GradientBoostingRegressor, RandomForestRegressor и
BaggingRegressor и выберем лучший.
```

#XGBRegressor

```
# GridSearchCV для получения наилучших гиперпараметров для winsorized target
- с использованием гамма-регрессии, поскольку смещение вправо
xgb1 = XGBRegressor(random_state=42)
              'objective':['reg:gamma'],
              'learning_rate': [0.03, .07, 0.1],
              'max_depth': [5, 9],
              'min_child_weight': [7, 15],
              'silent': [1],
              'subsample': [0.7, 1],
              'colsample_bytree': [0.7, 1],
              'n_estimators': [700, 1000],
              'eval_metric': ['gamma-deviance'],
              'gamma': [10]}
```

```
xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 3,
                        n_jobs = 5,
                        verbose=True, scoring='r2')

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)


xgbr_wins = XGBRegressor(verbosity=0, random_state=42, colsample_bytree=1,
learning_rate=0.1, max_depth=9, min_child_weight=7, n_estimators=1000,
nthread=4, \
                         objective='reg:gamma', silent=1, subsample=0.7,
gamma=10, eval_metric='gamma-deviance')
xgbr_wins.fit(X_train, y_train)
# taking learning_rate = 0.1 gives better results than 0.07


# Performance on Train

r2 = xgbr_wins.score(X_train, y_train)
mse = mean_squared_error(y_train, xgbr_wins.predict(X_train))
mae = mean_absolute_error(y_train, xgbr_wins.predict(X_train))
mape = mean_absolute_percentage_error(y_train, xgbr_wins.predict(X_train))
adj_r2 = 1 - (1-r2)*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)

print("Train RMSE: %.2f" % (mse**(1/2.0)))
print("Train Adjusted R2: ", adj_r2)


# Performance on Test

y_pred = xgbr_wins.predict(X_test)

r2 = xgbr_wins.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
adj_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)

print("Test RMSE: %.2f" % (mse**(1/2.0)))
print("Test Adjusted R2: ", adj_r2)


# GridSearchCV для получения наилучших гиперпараметров для цели,
преобразованной в журнал
xgb2 = XGBRegressor(random_state=42)
parameters = {'nthread':[4], #при использовании hyperthread xgboost может
работать медленнее
              'objective':['reg:squarederror'],
              'learning_rate': [0.03, .07, 0.1], #so called `eta` value
              'max_depth': [5, 9],
              'min_child_weight': [15, 30],
              'silent': [1],
              'subsample': [0.7, 1],
```

```
              'colsample_bytree': [0.7, 1],
              'n_estimators': [200, 300],
              'eval_metric': ['rmse']}

xgb_grid_log = GridSearchCV(xgb2,
                        parameters,
                        cv = 3,
                        n_jobs = 5,
                        verbose=True, scoring='r2')

xgb_grid_log.fit(X_train_log, y_train_log)

print(xgb_grid_log.best_score_)
print(xgb_grid_log.best_params_)


xgbr_log = XGBRegressor(verbosity=0, random_state=42, colsample_bytree=0.7,
learning_rate=0.03, max_depth=5, min_child_weight=30, n_estimators=200,
nthread=4, \
                    objective='reg:squarederror', silent=1, subsample=0.7,
reg_lambda=1, reg_alpha=0, eval_metric='rmse')
xgbr_log.fit(X_train_log, y_train_log)


# Производительность on Train
pred = xgbr_log.predict(X_train_log)
r2 = xgbr_log.score(X_train_log, y_train_log)
mse_wo_exp = mean_squared_error(y_train_log, pred)
mse = mean_squared_error(np.expm1(y_train_log), np.expm1(pred))
mae = mean_absolute_error(np.expm1(y_train_log), np.expm1(pred))
mape = mean_absolute_percentage_error(np.expm1(y_train_log), np.expm1(pred))
adj_r2 = 1 - (1-r2)*(len(y_train_log)-1)/(len(y_train_log)-
X_train_log.shape[1]-1)

print("Train RMSE: %.2f" % (mse**(1/2.0)))
print("Train RMSE w/o exp: %.2f" % (mse_wo_exp**(1/2.0)))
print("Train Adjusted R2: ", adj_r2)


# Производительность при тестировании

y_pred_log = xgbr_log.predict(X_test_log)

r2 = xgbr_log.score(X_test_log, y_test_log)
mse_wo_exp = mean_squared_error(y_test_log, xgbr_log.predict(X_test_log))
mse = mean_squared_error(np.expm1(y_test_log), np.expm1(y_pred_log))
mae = mean_absolute_error(np.expm1(y_test_log), np.expm1(y_pred_log))
mape = mean_absolute_percentage_error(np.expm1(y_test_log),
np.expm1(y_pred_log))
adj_r2 = 1 - (1-r2)*(len(y_test_log)-1)/(len(y_test_log)-X_test_log.shape[1]-
1)

print("Test RMSE: %.2f" % (mse**(1/2.0)))
print("Train RMSE w/o exp: %.2f" % (mse_wo_exp**(1/2.0)))
print("Test Adjusted R2: ", adj_r2)
```

```
feature_imp = pd.DataFrame(sorted(zip(xgbr_wins.feature_importances_,
X_train.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('XGBoost (Winsorized) Feature Importance')
plt.tight_layout()
plt.show()
```

```
feature_imp = pd.DataFrame(sorted(zip(xgbr_log.feature_importances_,
X_train_log.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('XGBoost (Log Transformed) Feature Importance')
plt.tight_layout()
plt.show()
```

#Первый блок содержит GridSearchCV, используемый для получения наилучших гиперпараметров для модели XGBoost, использующей гамма-регрессию для обработки winsorized target, так как данные имеют смещение вправо. Второй блок содержит еще один GridSearchCV, используемый для получения наилучших гиперпараметров для модели XGBoost для целевой переменной, преобразованной в журнал. Оба блока начинаются с определения модели XGBRegressor с помощью параметров по умолчанию и задания значений гиперпараметров, которые необходимо исследовать в GridSearchCV. Затем происходит запуск GridSearchCV с использованием определенных модели и параметров. Лучшая комбинация гиперпараметров сохраняется в xgb_grid.best*params* или xgb_grid_log.best*params* в зависимости от блока. Затем модель XGBoost инициализируется с использованием лучших параметров, которые были найдены с помощью GridSearchCV. Далее производится обучение модели на обучающих данных и оценка ее производительности на тренировочных и тестовых данных. Производительность модели оценивается с использованием нескольких метрик, таких как RMSE (среднеквадратическая ошибка), MAE (средняя абсолютная ошибка) и R2 (коэффициент детерминации). В первом блоке производительность модели оценивается на winsorized target, в то время как во втором блоке оценка производится на целевой переменной, преобразованной в журнал. После оценки производительности модели, во втором блоке производится обратное преобразование журнала, чтобы получить фактические значения целевой переменной. Обратите внимание, что обратное преобразование выполняется для сравнения с фактическими значениями, а не для обучения модели. В целом, блоки выполняют автоматический поиск наилучших гиперпараметров для модели XGBoost и оценивают ее производительность на обучающих и тестовых данных.

Случайный лес

```
from sklearn.model_selection import train_test_split, GridSearchCV, KFold,
cross_val_score, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel, RFE
```

```
scalerS = StandardScaler()
scalerM = MinMaxScaler()

X_train_std = scalerS.fit_transform(X_train)
```

```
X_test_std = scalerS.fit_transform(X_test)

X_train_norm = scalerM.fit_transform(X_train)
X_test_norm = scalerM.fit_transform(X_test)


def rfrmodelsummary(name, X_train, X_test, y_train, y_test, model):

    model.fit(X_train, y_train)

    # training
    y_pred_train = model.predict(X_train)

    # training metrics
    mse_train = mean_squared_error(y_train, y_pred_train, squared=True)
    rmse_train = np.sqrt(mse_train)
    rsq_train = model.score(X_train, y_train)
    adjrsq_train = 1 - (((len(y_train)-1)/(len(y_train)-X_train.shape[1]-
1))*(1-rsq_train))

    # test
    y_pred_test = model.predict(X_test)

    # test metrics
    mse_test = mean_squared_error(y_test, y_pred_test, squared=True)
    rmse_test = np.sqrt(mse_test)
    rsq_test = model.score(X_test, y_test)
    adjrsq_test = 1 - (((len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))*(1-
rsq_test))

    df_model = pd.DataFrame({'model': [name],
                             'R2_train': [rsq_train], 'R2_test': [rsq_test],
                             'AdjR2_train': [adjrsq_train], 'AdjR2_test':
[adjrsq_test],
                             'MSE_train': [mse_train], 'MSE_test' :
[mse_test],
                             'RMSE_train': [rmse_train], 'RMSE_test' :
[rmse_test]})

    return df_model


def plotrfr(name, X_train, X_test, y_train, y_test, model):

    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    plotdata = [y_pred_train, y_pred_test]

    return plotdata


rfr = RandomForestRegressor(random_state=42, n_jobs=-1)


rfr_models_result = pd.concat([rfrmodelsummary('Baseline', X_train, X_test,
y_train, y_test, rfr),
```

```
                                rfrmodelsummary('Standardized', X_train_std,
X_test_std, y_train, y_test, rfr),
                                rfrmodelsummary('Normalized', X_train_norm,
X_test_norm, y_train, y_test, rfr)])

rfr_models_result
```

#ПРИДЕРЖИВАЕМСЯ НЕСТАНДАРТИЗИРОВАННЫХ/НЕНОРМАЛИЗОВАННЫХ ДАННЫХ

```
plotdata = plotrfr('Baseline', X_train, X_test, y_train, y_test, rfr)


fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x=y_train, y=plotdata[0], s=10, c='b')

plt.title('Training Data')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()


fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x=y_test, y=plotdata[1], s=10, c='b')

plt.title('Test Data')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()


feature_imp = pd.DataFrame(sorted(zip(rfr.feature_importances_,
X_train.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('RFR Feature Importance')
plt.tight_layout()
plt.show()
```

#НИЗКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ ТЕСТИРОВАНИЯ (ПЕРЕОБУЧЕНИЕ). ПОПРОБУЙТЕ ВЫБРАТЬ ФУНКЦИЮ

```
RFEselector = RFE(estimator=rfr, n_features_to_select=7, step=1)
RFEselector.fit(X_train, y_train)
X_train.columns[RFEselector.get_support()]


RFEselector.ranking_
```

```
X_train.columns


X_train_rfe = RFEselector.transform(X_train)
X_test_rfe = RFEselector.transform(X_test)


rfe_models_result = pd.concat([rfrmodelsummary('Baseline', X_train, X_test,
y_train, y_test, RFEselector),
                               rfrmodelsummary('Standardized', X_train_std,
X_test_std, y_train, y_test, RFEselector),
                               rfrmodelsummary('Normalized', X_train_norm,
X_test_norm, y_train, y_test, RFEselector)])

rfe_models_result


SFMselector = SelectFromModel(estimator=rfr)
SFMselector.fit(X_train, y_train)
X_train.columns[SFMselector.get_support()]


X_train_sfm = SFMselector.transform(X_train)
X_test_sfm = SFMselector.transform(X_test)


sfm_models_result = rfrmodelsummary('Baseline', X_train_sfm, X_test_sfm,
y_train, y_test, rfr)
sfm_models_result
```

#Сначала импортируются необходимые библиотеки для разделения данных на обучающую и тестовую выборки, масштабирования данных, создания и обучения модели случайного леса, а также для выбора наиболее важных признаков (Приложение 1). Далее, данные разделяются на обучающую и тестовую выборки, затем применяются функции масштабирования StandardScaler и MinMaxScaler для стандартизации и нормализации данных. Затем функции rfrmodelsummary и plotrfr используются для создания отчета по метрикам производительности модели (таким как R2, MSE, RMSE) и построения графика фактических и предсказанных значений для обучающей и тестовой выборок. Далее, используется метод Recursive Feature Elimination (RFE) для выбора наиболее важных признаков для модели. В конце кода используется функция для визуализации важности признаков в модели.

гиперпараметр

```
rfr.get_params()


# Информация, которая поможет с настройкой

print(rfr.estimators_[5].tree_.max_depth) # check how many nodes in the
longest path
rfr.estimators_[5].tree_.n_node_samples # check how many samples in the last
nodes


# Извлечение данных из одного дерева

rfr_dict = {
```

```
    'id_node': list(range(rfr.estimators_[5].tree_.node_count)),
    'impurity': rfr.estimators_[5].tree_.impurity,
    'samples': rfr.estimators_[5].tree_.n_node_samples,
    'id_left_child': rfr.estimators_[5].tree_.children_left,
    'id_right_child': rfr.estimators_[5].tree_.children_right
}

impurity_df = pd.DataFrame(rfr_dict)
print(impurity_df.shape)
impurity_df.head(10)


# Вычислить минимальное изменение безопасности

impurity_df['impurity_decrease'] = np.nan
samples_total = rfr.estimators_[5].tree_.node_count

for idx in impurity_df.index[1:]:
    if impurity_df.iloc[idx]['id_left_child'] == -1:
        continue
    else:
        impurity_P, samples_P = impurity_df.iloc[idx][['impurity',
'samples']]
        id_L, id_R = impurity_df.iloc[idx][['id_left_child',
'id_right_child']].astype(int)
        impurity_L, samples_L = impurity_df.iloc[id_L][['impurity',
'samples']]
        impurity_R, samples_R = impurity_df.iloc[id_R][['impurity',
'samples']]

        impurity_decrease = samples_P / samples_total * (
            impurity_P - samples_R / samples_P * impurity_R -
            samples_L / samples_P * impurity_L
            )
        impurity_df.at[idx, 'impurity_decrease'] = impurity_decrease


impurity_df['impurity_decrease'].plot(kind='hist', bins=50)


impurity_df['impurity_decrease'].describe() # 75% значений для уменьшения
примесей лежат ниже 0,002; используйте это в качестве верхней границы для
параметра


path = rfr.estimators_[5].cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots(figsize=(13,5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()


# Настройка параметров с помощью случайного поиска
```

```
# param_distributions = {
#     'ccp_alpha': 0.0,
#     'max_depth': None,
#     'max_leaf_nodes': None,
#     'min_impurity_decrease': 0.0,
#     'min_impurity_split': None,
#     'min_samples_leaf': 1,
#     'min_samples_split': 2,
#     'min_weight_fraction_leaf': 0.0,
#     'max_features': 'auto'
# }

ccp_alpha = list(np.round(np.linspace(0,0.8,4), decimals=3))
max_depth = list(range(25,40,5))
max_leaf_nodes = [None] + list(range(2,20))
min_impurity_decrease = list(np.linspace(0, 0.001, 4))
min_samples_leaf = [1,2,5,10,50] #list(range(1, 51, 10))
min_samples_split = [2,5,10,15,20,50,100] #list(range(10,100,10))
max_features = ['auto', 'sqrt']
n_estimators = [500]

param_distributions = {
    'ccp_alpha': ccp_alpha,
    'max_depth': max_depth,
    'max_leaf_nodes': max_leaf_nodes,
    'min_impurity_decrease': min_impurity_decrease,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'max_features': max_features,
    'n_estimators': n_estimators
}

tuning_options = len(ccp_alpha) * len(max_depth) * len(max_leaf_nodes) *
len(min_impurity_decrease) * len(min_samples_leaf) * len(min_samples_split) *
len(max_features) * len(n_estimators)
print(tuning_options)


randomSearch = RandomizedSearchCV(rfr, param_distributions, cv=5, n_jobs=-1,
random_state=42, n_iter=100)


randomSearch.fit(X_train, y_train)
print('Best score:', randomSearch.best_score_)
print('Best parameters:', randomSearch.best_params_)


rfrRCV_models_result = rfrmodelsummary('Baseline', X_train, X_test, y_train,
y_test, randomSearch.best_estimator_)
rfrRCV_models_result


randomSearch.fit(X_train_rfe, y_train)
print('Best score:', randomSearch.best_score_)
print('Best parameters:', randomSearch.best_params_)
```

```
rfrRCV_models_result = rfrmodelsummary('Baseline', X_train_rfe, X_test_rfe,
y_train, y_test, randomSearch.best_estimator_)
rfrRCV_models_result


randomSearch.fit(X_train_sfm, y_train)
print('Best score:', randomSearch.best_score_)
print('Best parameters:', randomSearch.best_params_)


rfrRCV_models_result = rfrmodelsummary('Baseline', X_train_sfm, X_test_sfm,
y_train, y_test, randomSearch.best_estimator_)
rfrRCV_models_result
```

#Код определяет модель RFR с использованием библиотеки sklearn и выполняет различные операции, такие как проверка количества узлов в дереве, извлечение данных из дерева, вычисление нечистоты и обрезание дерева. Затем код определяет словарь гиперпараметров, которые будут настраиваться с помощью случайного поиска, создает объект RandomizedSearchCV для выполнения поиска и подгоняет модель RFR к обучающим данным. Наконец, код печатает лучший результат и лучшие параметры, найденные в результате поиска, а также вычисляет и печатает различные метрики для настроенной модели.

```
# Точная настройка параметров с поиском по сетке

max_depth = [23,24,25,26,27]
min_impurity_decrease = [0.0005]
min_samples_split = [14,15,16]
max_features = ['sqrt']
min_samples_leaf = [3,4,5,6,7]
n_estimators = [500]

param_grid = {
    'max_depth': max_depth,
    'min_impurity_decrease': min_impurity_decrease,
    'min_samples_split': min_samples_split,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'n_estimators': n_estimators
}

tuning_options = len(max_depth) * len(min_impurity_decrease) *
len(min_samples_split) * len(max_features) * len(min_samples_leaf) *
len(n_estimators)
print(tuning_options)


gridSearch = GridSearchCV(rfr, param_grid, cv=5, n_jobs=-1)


gridSearch.fit(X_train, y_train)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = rfrmodelsummary('Baseline', X_train, X_test, y_train,
y_test, gridSearch.best_estimator_)
```

```
rfrGCV_models_result


gridSearch.fit(X_train_rfe, y_train)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = rfrmodelsummary('Baseline', X_train_rfe, X_test_rfe,
y_train, y_test, gridSearch.best_estimator_)
rfrGCV_models_result
# final winsorized


rf_final_wins = RandomForestRegressor(max_depth=23, max_features='sqrt',
min_impurity_decrease=0.0005, min_samples_leaf=3, min_samples_split=14,
n_estimators=500, \
                                      random_state=42, n_jobs=-1)
rf_final_wins.fit(X_train_rfe, y_train)

y_pred_rf = rf_final_wins.predict(X_test_rfe)

r2_training = rf_final_wins.score(X_train_rfe, y_train)
adj_r2_training = 1 - (1-r2_training)*(len(y_train)-1)/(len(y_train)-
X_train_rfe.shape[1]-1)

r2_test = rf_final_wins.score(X_test_rfe, y_test)
adj_r2_test = 1 - (1-r2_test)*(len(y_test)-1)/(len(y_test)-
X_test_rfe.shape[1]-1)

print('training set: adjusted R2 score: %.4f' % adj_r2_training)
print('test set: adjusted R2 score: %.4f' % adj_r2_test)
print("training set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_train,
rf_final_wins.predict(X_train_rfe))))
print("test set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_test,
y_pred_rf)))


feature_imp = pd.DataFrame(sorted(zip(rf_final_wins.feature_importances_,
X_train.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('RFR Feature Importance')
plt.tight_layout()
plt.show()


gridSearch.fit(X_train_sfm, y_train)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = rfrmodelsummary('Baseline', X_train_sfm, X_test_sfm,
y_train, y_test, gridSearch.best_estimator_)
rfrGCV_models_result
```

```
rfr = RandomForestRegressor(random_state=42, n_jobs=-1)


def logrfrmodelsummary(name, X_train, X_test, y_train, y_test, model):

    model.fit(X_train, y_train)

    # training
    y_pred_train = model.predict(X_train)

    # training metrics
    mse_train = mean_squared_error(np.expm1(y_train), np.expm1(y_pred_train))
    mse_train_wo_exp = mean_squared_error(y_train, y_pred_train)
    rmse_train = np.sqrt(mse_train)
    rsq_train = model.score(X_train, y_train)
    adjrsq_train = 1 - (((len(y_train)-1)/(len(y_train)-X_train.shape[1]-
1))*(1-rsq_train))

    # test
    y_pred_test = model.predict(X_test)

    # test metrics
    mse_test = mean_squared_error(np.expm1(y_test), np.expm1(y_pred_test))
    mse_test_wo_exp = mean_squared_error(y_test, y_pred_test)
    rmse_test = np.sqrt(mse_test)
    rsq_test = model.score(X_test, y_test)
    adjrsq_test = 1 - (((len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))*(1-
rsq_test))

    df_model = pd.DataFrame({'model': [name],
                             'R2_train': [rsq_train], 'R2_test': [rsq_test],
                             'AdjR2_train': [adjrsq_train], 'AdjR2_test':
[adjrsq_test],
                             'MSE_train': [mse_train], 'MSE_test' :
[mse_test],
                             'MSE_train_wo_exp': [mse_train_wo_exp],
'MSE_test_wo_exp' : [mse_test_wo_exp],
                             'RMSE_train': [rmse_train], 'RMSE_test' :
[rmse_test]})

    return df_model


log_rfr_models_result = logrfrmodelsummary('Baseline', X_train_log,
X_test_log, y_train_log, y_test_log, rfr)
log_rfr_models_result


feature_imp = pd.DataFrame(sorted(zip(rfr.feature_importances_,
X_train_log.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('RFR Feature Importance')
plt.tight_layout()
plt.show()
```

```
RFEselector = RFE(estimator=rfr, n_features_to_select=7, step=1)
RFEselector.fit(X_train_log, y_train_log)
X_train_log.columns[RFEselector.get_support()]


RFEselector.ranking_


X_train_log.columns


X_train_rfe = RFEselector.transform(X_train_log)
X_test_rfe = RFEselector.transform(X_test_log)


rfe_models_result = logrfrmodelsummary('Baseline', X_train_rfe, X_test_rfe,
y_train_log, y_test_log, RFEselector)
rfe_models_result


SFMselector = SelectFromModel(estimator=rfr)
SFMselector.fit(X_train_log, y_train_log)
X_train_log.columns[SFMselector.get_support()]


X_train_sfm = SFMselector.transform(X_train_log)
X_test_sfm = SFMselector.transform(X_test_log)


sfm_models_result = logrfrmodelsummary('Baseline', X_train_sfm, X_test_sfm,
y_train_log, y_test_log, rfr)
sfm_models_result


# Информация, которая поможет с настройкой

print(rfr.estimators_[5].tree_.max_depth) # check how many nodes in the
longest path
rfr.estimators_[5].tree_.n_node_samples # check how many samples in the last
nodes


# Extract data on single tree

rfr_dict = {
    'id_node': list(range(rfr.estimators_[5].tree_.node_count)),
    'impurity': rfr.estimators_[5].tree_.impurity,
    'samples': rfr.estimators_[5].tree_.n_node_samples,
    'id_left_child': rfr.estimators_[5].tree_.children_left,
    'id_right_child': rfr.estimators_[5].tree_.children_right
}

impurity_df = pd.DataFrame(rfr_dict)
print(impurity_df.shape)
impurity_df.head(10)


# Вычислить min_impurity_decrease минимального значения
```

```
impurity_df['impurity_decrease'] = np.nan
samples_total = rfr.estimators_[5].tree_.node_count

for idx in impurity_df.index[1:]:  # skip the first node, there aren't any
splits prior to it
    if impurity_df.iloc[idx]['id_left_child'] == -1:
        continue  # we can't calculate impurity decrease for leaf nodes, as
they no longer split
    else:
        impurity_P, samples_P = impurity_df.iloc[idx][['impurity',
'samples']]
        id_L, id_R = impurity_df.iloc[idx][['id_left_child',
'id_right_child']].astype(int)
        impurity_L, samples_L = impurity_df.iloc[id_L][['impurity',
'samples']]
        impurity_R, samples_R = impurity_df.iloc[id_R][['impurity',
'samples']]

        impurity_decrease = samples_P / samples_total * (
            impurity_P - samples_R / samples_P * impurity_R -
            samples_L / samples_P * impurity_L
            )
        impurity_df.at[idx, 'impurity_decrease'] = impurity_decrease


impurity_df['impurity_decrease'].plot(kind='hist', bins=50)


impurity_df['impurity_decrease'].describe() # 75% of values for impurity
decrease lie below 0.000; use this as upper-bound for param


path = rfr.estimators_[5].cost_complexity_pruning_path(X_train_log,
y_train_log)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots(figsize=(13,5))
# корневой узел дерева (индексированный с помощью [:-1]) исключается из
анализа, так как без него не было бы дерева
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()
```

GridSearchCV использует для настройки гиперпараметров модели RandomForestRegressor. Скрипт определяет набор гиперпараметров для модели и затем создает сетку параметров, которая включает все возможные комбинации значений гиперпараметров. Скрипт выводит количество вариантов настройки, которые будут исследоваться, а затем подгоняет объект GridSearchCV к тренировочным данным. После подгонки скрипт выводит лучший результат и лучшие параметры, найденные GridSearchCV. Затем скрипт подгоняет модель RandomForestRegressor к данным с использованием лучших гиперпараметров, найденных GridSearchCV, и выводит метрики оценки модели, включая скорректированный R-квадрат, среднеквадратическую ошибку (RMSE) и важность признаков. Скрипт повторяет этот процесс с тремя различными наборами признаков: исходным набором признаков, уменьшенным набором признаков, полученным с помощью рекурсивного отбора признаков, и уменьшенным набором признаков, полученным с помощью объекта SelectFromModel для отбора признаков.

```
# Настройка параметров с помощью случайного поиска

# param_distributions = {
#     'ccp_alpha': 0.0,
#     'max_depth': None,
#     'max_leaf_nodes': None,
#     'min_impurity_decrease': 0.0,
#     'min_impurity_split': None,
#     'min_samples_leaf': 1,
#     'min_samples_split': 2,
#     'min_weight_fraction_leaf': 0.0,
#     'max_features': 'auto'
# }

ccp_alpha = list(np.round(np.linspace(0,0.8,4), decimals=3))
max_depth = list(range(25,40,5))
max_leaf_nodes = [None] + list(range(2,20))
min_impurity_decrease = [0]
min_samples_leaf = [1,2,5,10,50] #list(range(1, 51, 10))
min_samples_split = [2,5,10,15,20,50,100] #list(range(10,100,10))
max_features = ['auto', 'sqrt']
n_estimators = [500]

param_distributions = {
    'ccp_alpha': ccp_alpha,
    'max_depth': max_depth,
    'max_leaf_nodes': max_leaf_nodes,
    'min_impurity_decrease': min_impurity_decrease,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'max_features': max_features,
    'n_estimators': n_estimators
}

tuning_options = len(ccp_alpha) * len(max_depth) * len(max_leaf_nodes) *
len(min_impurity_decrease) * len(min_samples_leaf) * len(min_samples_split) *
len(max_features) * len(n_estimators)
print(tuning_options)


randomSearch = RandomizedSearchCV(rfr, param_distributions, cv=5, n_jobs=-1,
random_state=42, n_iter=100)


randomSearch.fit(X_train_log, y_train_log)
print('Best score:', randomSearch.best_score_)
print('Best parameters:', randomSearch.best_params_)


rfrRCV_models_result = logrfrmodelsummary('Baseline', X_train_log,
X_test_log, y_train_log, y_test_log, randomSearch.best_estimator_)
rfrRCV_models_result


randomSearch.fit(X_train_rfe, y_train_log)
print('Best score:', randomSearch.best_score_)
```

```
print('Best parameters:', randomSearch.best_params_)


rfrRCV_models_result = logrfrmodelsummary('Baseline', X_train_rfe,
X_test_rfe, y_train_log, y_test_log, randomSearch.best_estimator_)
rfrRCV_models_result


randomSearch.fit(X_train_sfm, y_train_log)
print('Best score:', randomSearch.best_score_)
print('Best parameters:', randomSearch.best_params_)


rfrRCV_models_result = logrfrmodelsummary('Baseline', X_train_sfm,
X_test_sfm, y_train_log, y_test_log, randomSearch.best_estimator_)
rfrRCV_models_result


# Точная настройка параметров с поиском по сетке

max_depth = [28,29,30,31,32]
min_impurity_decrease = [0]
min_samples_split = [8,9,10,11,12]
max_features = ['auto']
min_samples_leaf = [2,3,4]
n_estimators = [500]

param_grid = {
    'max_depth': max_depth,
    'min_impurity_decrease': min_impurity_decrease,
    'min_samples_split': min_samples_split,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'n_estimators': n_estimators
}

tuning_options = len(max_depth) * len(min_impurity_decrease) *
len(min_samples_split) * len(max_features) * len(min_samples_leaf) *
len(n_estimators)
print(tuning_options)


gridSearch = GridSearchCV(rfr, param_grid, cv=5, n_jobs=-1)


gridSearch.fit(X_train_log, y_train_log)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = logrfrmodelsummary('Baseline', X_train_log,
X_test_log, y_train_log, y_test_log, gridSearch.best_estimator_)
rfrGCV_models_result
# final log


rf_final_log = RandomForestRegressor(max_depth=28, max_features='auto',
min_impurity_decrease=0, min_samples_leaf=4, min_samples_split=12,
n_estimators=500, \
```

```
                                        random_state=42, n_jobs=-1)
rf_final_log.fit(X_train_log, y_train_log)

y_pred_rf_log = rf_final_log.predict(X_test_log)

r2_training = rf_final_log.score(X_train_log, y_train_log)
adj_r2_training = 1 - (1-r2_training)*(len(y_train_log)-1)/(len(y_train_log)-
X_train_log.shape[1]-1)

r2_test = rf_final_log.score(X_test_log, y_test_log)
adj_r2_test = 1 - (1-r2_test)*(len(y_test_log)-1)/(len(y_test_log)-
X_test_log.shape[1]-1)

print('training set: adjusted R2 score: %.4f' % adj_r2_training)
print('test set: adjusted R2 score: %.4f' % adj_r2_test)
print("training set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_train_log,
rf_final_log.predict(X_train_log))))
print("test set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_test_log,
y_pred_rf_log)))


feature_imp = pd.DataFrame(sorted(zip(rf_final_log.feature_importances_,
X_train_log.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 5))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('RFR Feature Importance')
plt.tight_layout()
plt.show()


gridSearch.fit(X_train_rfe, y_train_log)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = logrfrmodelsummary('Baseline', X_train_rfe,
X_test_rfe, y_train_log, y_test_log, gridSearch.best_estimator_)
rfrGCV_models_result


gridSearch.fit(X_train_sfm, y_train_log)
print('Best score:', gridSearch.best_score_)
print('Best parameters:', gridSearch.best_params_)


rfrGCV_models_result = logrfrmodelsummary('Baseline', X_train_sfm,
X_test_sfm, y_train_log, y_test_log, gridSearch.best_estimator_)
rfrGCV_models_result
```

#Примечание: Приведенный выше код регрессии случайного леса также тестировался при следующих условиях -

1. Удалены оценка и рейтингование перед вычислением корреляций, поскольку они были неуместны или избыточны по отношению к другим переменным

2. Уменьшенный общий набор данных (всего ~5 тыс.) с удалением всех пропущенных значений и включением isFemale_lead, isFemale_support, isFemale_director

GradientBoostingRegressor

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor

### Select features - RFE
GB = GradientBoostingRegressor()
rfe = RFE(estimator=GB,  n_features_to_select=25)
fit = rfe.fit(X, y)
#print("Num Features: %s" % (fit.n_features_))
#print("Selected Features: %s" % (fit.support_))
#print("Feature Ranking: %s" % (fit.ranking_))
#print(X.columns)

selected_feature_RFE = []
for x,y in zip(fit.ranking_,X.columns):
    if x == 1:
        selected_feature_RFE.append(y)
selected_feature_RFE


### Select features - VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

vif_data = vif_data.sort_values(by = 'VIF',ascending=True)
vif_data
selected_feature_VIF = vif_data.feature[:25]


#Градиентный ускоряющий регрессор

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error

GBR = GradientBoostingRegressor()
GBR.fit(X_train_GB, y_train_GB)
y_pred_GB = GBR.predict(X_test_GB)

r2_training = GBR.score(X_train_GB, y_train_GB)
adj_r2_training = 1 - (1-r2_training)*(len(y_train_GB)-1)/(len(y_train_GB)-
X_train_GB.shape[1]-1)

r2_test = GBR.score(X_test_GB, y_test_GB)
```

```
adj_r2_test = 1 - (1-r2_test)*(len(y_test_GB)-1)/(len(y_test_GB)-
X_test_GB.shape[1]-1)

print('training set: adjusted R2 score: %.4f' % adj_r2_training)
print('test set: adjusted R2 score: %.4f' % adj_r2_test)
print("training set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_train_GB,
GBR.predict(X_train_GB))))
print("test set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_test_GB,
y_pred_GB)))


X_train_GB.columns


param_test1 = {'n_estimators': [590,610,630,670,720,750],'learning_rate':
[0.01,0.02]}


#from sklearn.model_selection import GridSearchCV
GBR = GradientBoostingRegressor(max_depth=6, subsample=0.5,random_state = 1)
#starting values
grid_GBR = GridSearchCV(estimator=GBR, param_grid = param_test1, cv = 3,
n_jobs=-1)
grid_GBR.fit(X_train_GB, y_train_GB)
print("\n The best estimator across ALL searched
params:\n",grid_GBR.best_estimator_)
print("\n The best score across ALL searched params:\n",grid_GBR.best_score_)


param_test2 = {'min_samples_split':range(5,20),
'min_samples_leaf':range(2,20,5)}


GBR2 = GradientBoostingRegressor(max_depth=6,
subsample=0.5,learning_rate=0.01,n_estimators=610)
grid_GBR2 = GridSearchCV(estimator=GBR2, param_grid = param_test2, cv = 3,
n_jobs=-1)
grid_GBR2.fit(X_train_GB, y_train_GB)
print("\n The best estimator across ALL searched
params:\n",grid_GBR2.best_estimator_)
print("\n The best score across ALL searched
params:\n",grid_GBR2.best_score_)


param_test3 = {'max_depth':[6,8,12,20,25],'subsample': [0.9, 0.5, 0.4, 0.3]}


GBR3 =
GradientBoostingRegressor(learning_rate=0.01,n_estimators=610,min_samples_spl
it=10,\
                                  min_samples_leaf = 17)
grid_GBR3 = GridSearchCV(estimator=GBR3, param_grid = param_test3, cv = 3,
n_jobs=-1)
grid_GBR3.fit(X_train_GB, y_train_GB)
print("\n The best estimator across ALL searched
params:\n",grid_GBR3.best_estimator_)
print("\n The best score across ALL searched
params:\n",grid_GBR3.best_score_)
```

```
# Final Test
import math
from sklearn.metrics import r2_score, mean_squared_error

GBR_4 = GradientBoostingRegressor(learning_rate=0.01, max_depth=6,
min_samples_leaf=17,
                          min_samples_split=10, n_estimators=610,
                          subsample=0.5,loss = 'squared_error')
GBR_4.fit(X_train_GB, y_train_GB)
y_pred = GBR_4.predict(X_test_GB)

r2_training = GBR_4.score(X_train_GB, y_train_GB)
adj_r2_training = 1 - (1-r2_training)*(len(y_train_GB)-1)/(len(y_train_GB)-
X_train_GB.shape[1]-1)

r2_test = GBR_4.score(X_test_GB, y_test_GB)
adj_r2_test = 1 - (1-r2_test)*(len(y_test_GB)-1)/(len(y_test_GB)-
X_test_GB.shape[1]-1)

print('training set: adjusted R2 score: %.4f' % adj_r2_training)
print('test set: adjusted R2 score: %.4f' % adj_r2_test)
print("training set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_train_GB,
GBR_4.predict(X_train_GB))))
print("test set: RMSE: %.4f" % np.sqrt(mean_squared_error(y_test_GB,
y_pred)))


feature_imp = pd.DataFrame(sorted(zip(GBR_4.feature_importances_,
X_train_GB.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 6))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('GradientBoosting Feature Importance')
plt.tight_layout()
plt.show()


param_test1 = {'n_estimators': [550,650,700,750,800],'learning_rate':
[0.01,0.02]}

GBR = GradientBoostingRegressor(max_depth=6, subsample=0.5,random_state = 1)
grid_GBR = GridSearchCV(estimator=GBR, param_grid = param_test1, cv = 2,
n_jobs=-1)
grid_GBR.fit(X_train_log, y_train_log)
print("\n The best estimator across ALL searched
params:\n",grid_GBR.best_estimator_)
print("\n The best score across ALL searched params:\n",grid_GBR.best_score_)


param_test2 = {'min_samples_split':range(10,20),
'min_samples_leaf':range(2,20,5)}
GBR2 = GradientBoostingRegressor(max_depth=6,
subsample=0.5,learning_rate=0.01,n_estimators=650)
grid_GBR2 = GridSearchCV(estimator=GBR2, param_grid = param_test2, cv = 2,
n_jobs=-1)
```

```
grid_GBR2.fit(X_train_log, y_train_log)
print("\n The best estimator across ALL searched
params:\n",grid_GBR2.best_estimator_)
print("\n The best score across ALL searched
params:\n",grid_GBR2.best_score_)


param_test3 = {'max_depth':[3,4,5,6,8,12],'subsample': [0.9, 0.5, 0.4, 0.3]}
GBR3 =
GradientBoostingRegressor(learning_rate=0.01,n_estimators=650,min_samples_spl
it=7,\
                                  min_samples_leaf = 14)
grid_GBR3 = GridSearchCV(estimator=GBR3, param_grid = param_test3, cv = 2,
n_jobs=-1)
grid_GBR3.fit(X_train_log, y_train_log)
print("\n The best estimator across ALL searched
params:\n",grid_GBR3.best_estimator_)
print("\n The best score across ALL searched
params:\n",grid_GBR3.best_score_)


GBR_log = GradientBoostingRegressor(learning_rate=0.01, max_depth=6,
min_samples_leaf=14,
                           min_samples_split=7, n_estimators=650,
subsample=0.5,loss = 'squared_error')
GBR_log.fit(X_train_log, y_train_log)
y_pred_log = GBR_log.predict(X_test_log)

r2_training = GBR_log.score(X_train_log, y_train_log)
adj_r2_training = 1 - (1-r2_training)*(len(y_train_log)-1)/(len(y_train_log)-
X_train_log.shape[1]-1)

r2_test = GBR_log.score(X_test_log, y_test_log)
adj_r2_test = 1 - (1-r2_test)*(len(y_test_log)-1)/(len(y_test_log)-
X_test_log.shape[1]-1)

print('training set: adjusted R2 score: %.4f' % adj_r2_training)
print('test set: adjusted R2 score: %.4f' % adj_r2_test)
print("training set RMSE(back transformed): %.4f" %
np.sqrt(mean_squared_error(np.expm1(y_train_log),
np.expm1(GBR_log.predict(X_train_log)))))
print("test set RMSE(back transformed): %.4f" %
np.sqrt(mean_squared_error(np.expm1(y_test_log), np.expm1(y_pred_log))))


print("training set RMSE: %.4f" % np.sqrt(mean_squared_error(y_train_log,
GBR_log.predict(X_train_log))))
print("test set RMSE: %.4f" % np.sqrt(mean_squared_error(y_test_log,
y_pred_log)))


feature_imp = pd.DataFrame(sorted(zip(GBR_log.feature_importances_,
X_train_log.columns),reverse = True), columns=['Value','Feature'])
feature_imp = feature_imp[feature_imp.Value != 0]
plt.figure(figsize=(10, 6))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
ascending=False))
plt.title('GradientBoosting Feature Importance')
```

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

#Сначала импортируются необходимые библиотеки для разделения данных на обучающую и тестовую выборки, масштабирования данных, создания и обучения модели случайного леса, а также для выбора наиболее важных признаков. Далее, данные разделяются на обучающую и тестовую выборки, затем применяются функции масштабирования StandardScaler и MinMaxScaler для стандартизации и нормализации данных. Затем функции rfrmodelsummary и plotrfr используются для создания отчета по метрикам производительности модели (таким как R2, MSE, RMSE) и построения графика фактических и предсказанных значений для обучающей и тестовой выборок. Далее, используется метод Recursive Feature Elimination (RFE) для выбора наиболее важных признаков для модели. В конце кода используется функция для визуализации важности признаков в модели.

```
y_pred_gbm_train = GBR_4.predict(X_train)
y_pred_xgbr_train = np.expm1(xgbr_log.predict(X_train))

y_pred_avg_train = (y_pred_gbm_train + y_pred_xgbr_train)/2
np.sqrt(mean_squared_error(y_train, y_pred_avg_train))
```

```
y_pred_gbm = y_pred
y_pred_xgbr = np.expm1(y_pred_log)

y_pred_avg = (y_pred_gbm + y_pred_xgbr)/2
np.sqrt(mean_squared_error(y_test, y_pred_avg))
```

#Здесь использует две модели, Gradient Boosting Regressor (GBR) и Extreme Gradient Boosting Regressor (XGBR), для предсказания целевой переменной как для обучающей, так и для тестовой выборок. Предсказанные значения от обеих моделей усредняются, а затем вычисляется среднеквадратичное отклонение (RMSE) между реальными и предсказанными значениями для обучающей и тестовой выборок. RMSE является распространенной метрикой для оценки производительности моделей регрессии, и меньшие значения указывают на лучшую производительность модели.