

```

# Импортируем все необходимые библиотеки

import pandas as pd
import numpy as np
from sqlalchemy import create_engine
from dotenv import load_dotenv
import os
import re
import unicodedata
from typing import Tuple

# import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings

warnings.filterwarnings("ignore")

# Загружаем переменные окружения из .env файла
load_dotenv()

True

# Создаем URL для SQLAlchemy
SQLALCHEMY_DATABASE_URL = (
    f"postgresql://"
    f"{os.getenv('POSTGRES_USER')}:{os.getenv('POSTGRES_PASSWORD')}}@"
    f"{os.getenv('POSTGRES_HOST')}:{os.getenv('POSTGRES_PORT')}}/"
    f"{os.getenv('POSTGRES_DATABASE')}}"
)
engine = create_engine(SQLALCHEMY_DATABASE_URL)

```

Выгрузим три таблицы из базы данных

1. Таблица user_data

```

# Структура таблицы:
user_info = {
    "user_id": "Уникальный идентификатор пользователя (int64)",
    "gender": "Пол пользователя (int: 0/1)",
    "age": "Возраст пользователя (int32, 0-99)",
    "country": "Страна пользователя (str: 'Russian', 'Belarus', 'Iran', ...)",
    "city": "Город пользователя (str: 'Moscow', 'Abakan', ...)",
    "exp_group": "Экспериментальная группа (int8: 0-4)",
    "os": "Операционная система (str: 'iOS', 'Android', 'web')",
}

```

```
    "source": "Источник привлечения (str: 'ads', ...)"
}
```

1. Таблица `post_text_df`

```
# Структура таблицы:
posts_info = {
    "post_id": "Уникальный ID поста (int64)",
    "text": "Текст поста (str)",
    "topic": "Категория (str: 'politics', 'entertainment', ...)",
}
```

1. Таблица `feed_data`

```
# Структура таблицы:
feed_data = {
    "user_id": "ID юзера (int64)",
    "post_id": "ID поста (int64)",
    "text": "Текст поста (str)",
    "action": "view/like (str)",
    "timestamp": "Время просмотра поста (timestamp)",
    "target": "0/1 (int64): 1 - если пост был лайкнут в ближайшее время после просмотра (условно в первые 1-2 минуты)",
}
```

Данные по пользователям

```
user_info = pd.read_sql("""SELECT * FROM user_data""", con=engine)
```

```
user_info.head()
```

	user_id	gender	age	country	city	exp_group	os
source							
0	200	1	34	Russia	Degtyarsk	3	Android
ads							
1	201	0	37	Russia	Abakan	0	Android
ads							
2	202	1	17	Russia	Smolensk	4	Android
ads							
3	203	0	18	Russia	Moscow	1	iOS
ads							
4	204	0	36	Russia	Anzhero-Sudzhensk	3	Android
ads							

Посты и топики

```
posts_info = pd.read_sql("""SELECT * FROM post_text_df""", con=engine)
```

```
posts_info.head()
```

	post_id	text
topic		
0	1	UK economy facing major risks\n\nThe UK manufa...
business		
1	2	Aids and climate top Davos agenda\n\nClimate c...
business		
2	3	Asian quake hits European shares\n\nShares in ...
business		
3	4	India power shares jump on debut\n\nShares in ...
business		
4	5	Lacroix label bought by US firm\n\nLuxury good...
business		

Попробуем забрать, скажем, 5 миллионов

```
feed_data = pd.read_sql(
    """SELECT * FROM feed_data WHERE action = 'view' LIMIT 5000000""",
    con=engine
)
```

```
feed_data.head()
```

	timestamp	user_id	post_id	action	target
0	2021-10-11 20:45:10	55365	1650	view	1
1	2021-10-11 20:45:41	55365	956	view	0
2	2021-10-11 20:47:36	55365	1875	view	1
3	2021-10-11 20:50:15	55365	1039	view	0
4	2021-10-11 20:52:11	55365	2571	view	0

Проверяем распределение классов

```
class_distribution = (
    feed_data["target"].value_counts(normalize=True) * 100
) # В процентах
```

Создаем график

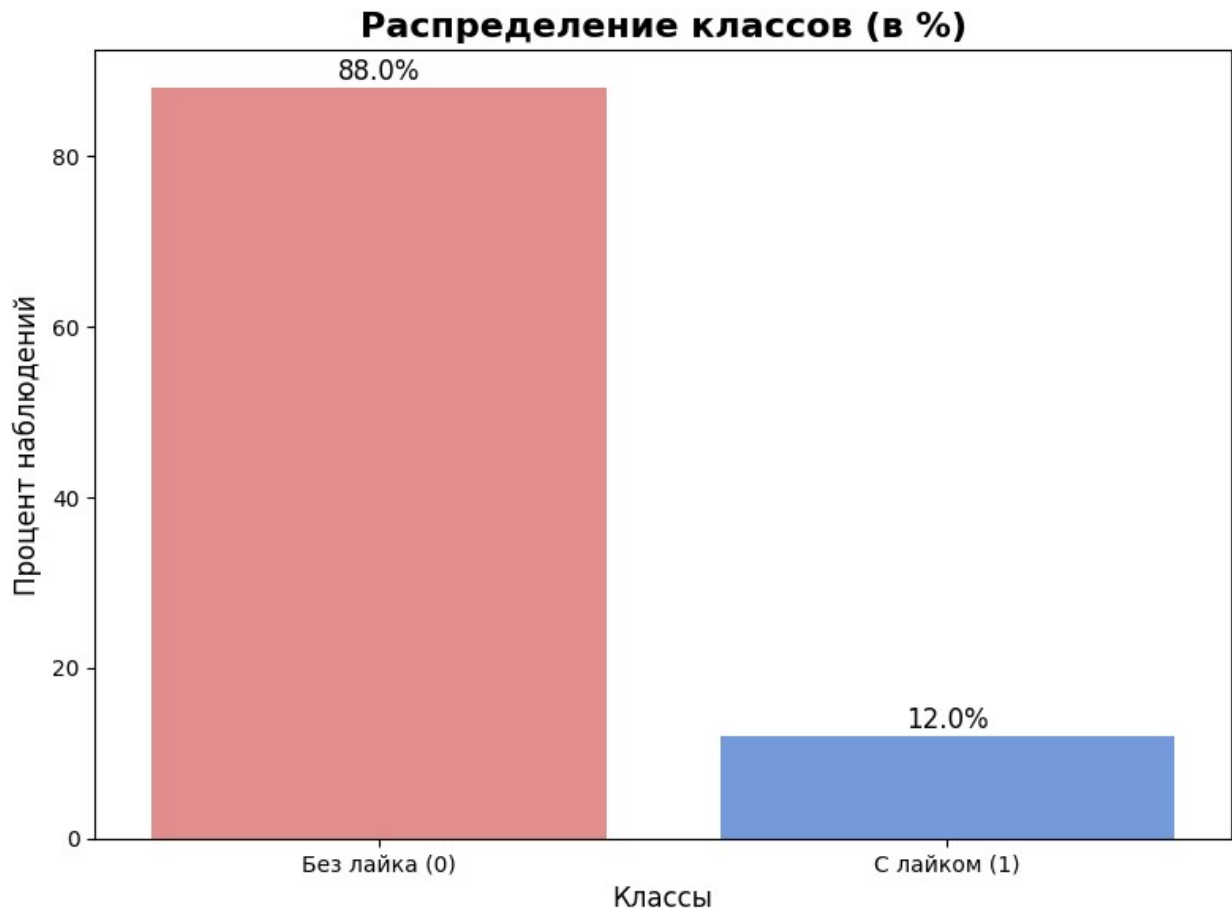
```
plt.figure(figsize=(8, 6))
sns.barplot(
    x=class_distribution.index,
    y=class_distribution.values,
    palette=["lightcoral", "cornflowerblue"], # Цвета для классов
)
```

Добавляем подписи и заголовок

```
plt.title("Распределение классов (в %)", fontsize=16,
fontweight="bold")
plt.xlabel("Классы", fontsize=12)
plt.ylabel("Процент наблюдений", fontsize=12)
plt.xticks(ticks=[0, 1], labels=["Без лайка (0)", "С лайком (1)"],
fontsize=10)
```

```
# Добавляем процентные значения над столбцами
for i, value in enumerate(class_distribution):
    plt.text(i, value + 1, f"{value:.1f}%", ha="center", fontsize=12,
color="black")

# Отображаем график
plt.tight_layout()
plt.show()
```



Анализ дисбаланса классов

На графике явно виден значительный дисбаланс между классами, что вполне объяснимо с точки зрения поведения пользователей. В реальности пользователи чаще просматривают контент, чем ставят лайки — это естественная особенность взаимодействия с платформой. Такое поведение приводит к тому, что положительный класс (лайки) оказывается значительно менее представленным по сравнению с отрицательным классом (просмотры без лайков).

Этот дисбаланс необходимо учитывать при разработке модели, чтобы избежать перекоса в сторону большинства и обеспечить качественное предсказание лайков.

Модель может научиться всегда предсказывать "не лайк" и получить искусственно высокую ассурасу. Требуется специальная обработка для корректного обучения:

1. Метрики: F1-score, Precision-Recall AUC вместо accuracy
 2. Методы: class weighting, oversampling/undersampling
 3. Архитектура: ансамблирование или threshold moving
-

Feature Engineering для таблицы `post_text_df`

□ План генерации признаков

1. Базовые текстовые метрики

Генерируемые фичи:

- `text_length` - длина текста в символах
- `word_count` - количество слов в тексте
- `digit_count` - количество цифр в тексте

Цель:

Выявить зависимость между структурой текста и пользовательской вовлеченностью.

2. TF-IDF векторизация с предобработкой

Этапы:

1. Предобработка текста с помощью `sraCy`:
 - Лемматизация
 - Удаление стоп-слов
 - Удаление пунктуации/чисел
2. Построение TF-IDF матрицы (10000 наиболее значимых терминов)

Особенности:

- Используется русскоязычная модель `ru_core_news_sm`
 - Минимальная длина токена - 5 символа
-

3. Агрегатные статистики TF-IDF

Генерируемые фичи:

- `tfidf_mean` - среднее значение TF-IDF по терминам
- `tfidf_sum` - сумма значений TF-IDF
- `tfidf_max` - максимальное значение TF-IDF

Гипотеза:

Агрегатные характеристики помогут:

- Оценить общую информативность поста (**mean**)
- Измерить суммарный "вес" ключевых слов (**sum**)
- Выявить доминирующий термин (**max**)

PS: В конце поймем, что особо ничего полезного эти фичи не принесли :(

4. Снижение размерности

Метод: TruncatedSVD (300 компонент)

Преимущества:

- Стараемся сохранить как можно больше дисперсии данных
 - Уменьшает шумовые компоненты
 - Оптимизирует вычислительные затраты для последующей кластеризации
-

5. Кластеризация текстов

Параметры:

- Метод: K-Means
- Количество кластеров: 25
- Дополнительные фичи: расстояния до центроидов всех кластеров

Зачем нужны расстояния:

- Показывают степень принадлежности к кластерам
 - Характеризуют положение поста в признаковом пространстве
 - Позволяют выявлять аномальные тексты
-

6. Метрики популярности

Источник данных: таблица **feed_data**

Рассчитываемые показатели:

- **views** - общее количество просмотров
- **likes** - количество лайков
- **ctr** - коэффициент конверсии (лайки/просмотры)

Аналитическая ценность:

- Возможность связать текстовые характеристики с пользовательским engagement
 - Выявление "идеального" профиля поста
-

```

# Загрузка модели spaCy
nlp = spacy.load("en_core_web_sm")

### Функция для предобработки текста
def spacy_preprocess(text: str) -> Tuple[str, int]:
    """
    Очищает текст и унифицирует шрифт:
    - Убирает пунктуацию
    - Группирует числа в один признак (количество чисел)
    - Приводит текст к нижнему регистру
    - Лемматизирует слова
    - Удаляет стоп-слова и короткие слова
    - Унифицирует шрифт (например, убирает стильные шрифты)
    """
    # 1. Унификация шрифта
    text = unicodedata.normalize("NFKC", text)

    # 2. Удаляем HTML-теги и эмодзи
    text = re.sub(r"<[^>]+>", " ", text) # HTML-теги
    text = re.sub(r"[\U00010000-\U0010ffff]", " ", text) # Эмодзи

    # 3. Убираем пунктуацию и лишние пробелы
    text = re.sub(r"^[^\w\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()

    # 4. Заменяем числа на <NUMBER>
    numbers_count = len(re.findall(r"\d+", text))
    text = re.sub(r"\d+", "<NUMBER>", text)

    # 5. Приводим к нижнему регистру
    text = text.lower()

    # 6. Лемматизация и фильтрация
    doc = nlp(text)
    tokens = [
        token.lemma_
        for token in doc
        if token.is_alpha and not token.is_stop and len(token) > 2
    ]

    return " ".join(tokens), numbers_count

# Применяем spaCy очистку
posts_info["numbers_count"] = posts_info["text"].apply(lambda x:
    spacy_preprocess(x)[1])
posts_info["cleaned_text"] = posts_info["text"].apply(lambda x:
    spacy_preprocess(x)[0])

```

```

# Функция для подсчета количества слов
def count_words(text):
    if isinstance(text, str): # Проверяем, что текст не NaN
        words = re.findall(
            r"\b\w+\b", text
        ) # Ищем только слова (без знаков препинания)
        return len(words)
    return 0

# Добавляем новые столбцы
posts_info["text_length"] = posts_info["text"].apply(len) # Длина
# текста
posts_info["word_count"] = posts_info["text"].apply(count_words) #
# Количество слов

posts_info.head()

```

	post_id	text
topic \		
0	1	UK economy facing major risks\n\nThe UK manufa...
business		
1	2	Aids and climate top Davos agenda\n\nClimate c...
business		
2	3	Asian quake hits European shares\n\nShares in ...
business		
3	4	India power shares jump on debut\n\nShares in ...
business		
4	5	Lacroix label bought by US firm\n\nLuxury good...
business		

	numbers_count	cleaned_text \
0	13	economy face major risk manufacturing sector c...
1	20	aid climate davos agenda climate change fight ...
2	47	asian quake hit european share share europe le...
3	6	india power share jump debut share india large...
4	3	lacroix label buy firm luxury good group lvmh ...

	text_length	word_count
0	1967	327
1	2701	449
2	3408	571
3	1026	178
4	889	155

```

# Настройка стиля графиков
sns.set(style="whitegrid")

# Создаем фигуру с двумя подграфиками
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

```



```

# График распределения длины текста (text_length)
sns.histplot(
    data=posts_info,
    x="text_length",
    bins=50,
    kde=True,
    color="cornflowerblue",
    ax=axes[0],
)
axes[0].set_title("Распределение длины текста", fontsize=14,
fontweight="bold")
axes[0].set_xlabel("Длина текста (символы)", fontsize=12)
axes[0].set_ylabel("Частота", fontsize=12)

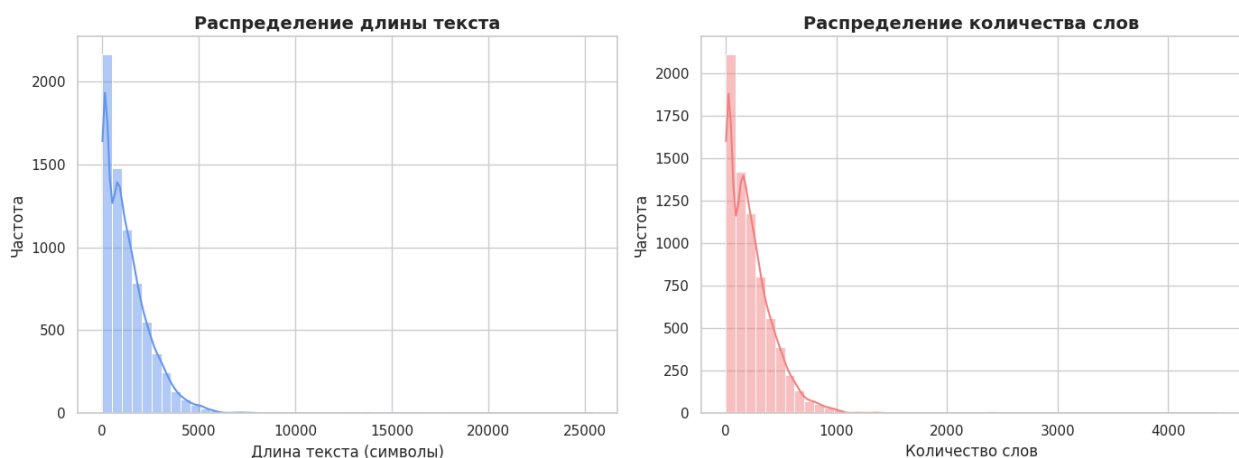
# График распределения количества слов (word_count)
sns.histplot(
    data=posts_info, x="word_count", bins=50, kde=True,
color="lightcoral", ax=axes[1]
)
axes[1].set_title("Распределение количества слов", fontsize=14,
fontweight="bold")
axes[1].set_xlabel("Количество слов", fontsize=12)
axes[1].set_ylabel("Частота", fontsize=12)

# Общий заголовок
fig.suptitle("Анализ распределения фичей текста", fontsize=18,
fontweight="bold")

# Отображаем график
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```

Анализ распределения фичей текста



0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000										
1	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000										
2	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.051226										
3	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000										
4	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000										

	post_id
0	1
1	2
2	3
3	4
4	5

[5 rows x 15026 columns]

Генерим фичи на основе TfIdf (возьмем сначала простую агрегацию)

```
posts_info["TotalTfIdf"] = (
    tfidf_spacy_df.drop(columns=["post_id", "topic"], axis=1)
    .sum(axis=1)
    .reset_index()[0]
)
posts_info["MaxTfIdf"] = (
    tfidf_spacy_df.drop(columns=["post_id", "topic"], axis=1)
    .max(axis=1)
    .reset_index()[0]
)
posts_info["MeanTfIdf"] = (
    tfidf_spacy_df.drop(columns=["post_id", "topic"], axis=1)
    .mean(axis=1)
    .reset_index()[0]
)
```

posts_info.head()

	post_id	text
topic \		
0	1	UK economy facing major risks\n\nThe UK manufa...
business		
1	2	Aids and climate top Davos agenda\n\nClimate c...
business		
2	3	Asian quake hits European shares\n\nShares in ...
business		
3	4	India power shares jump on debut\n\nShares in ...
business		
4	5	Lacroix label bought by US firm\n\nLuxury good...

business

	numbers_count	cleaned_text \
0	13	economy face major risk manufacturing sector c...
1	20	aid climate davos agenda climate change fight ...
2	47	asian quake hit european share share europe le...
3	6	india power share jump debut share india large...
4	3	lacroix label buy firm luxury good group lvmh ...

	text_length	word_count	TotalTfIdf	MaxTfIdf	MeanTfIdf
0	1967	327	8.134297	0.511790	0.000541
1	2701	449	11.356141	0.254945	0.000756
2	3408	571	10.911138	0.410319	0.000726
3	1026	178	6.754817	0.372043	0.000450
4	889	155	5.946105	0.526337	0.000396

Снизим размерность TF-IDF матрицы

```
centered = tfidf_spacy_matrix - tfidf_spacy_matrix.mean()
```

TruncatedSVD

```
svd = TruncatedSVD(n_components=300, random_state=42)
```

```
svd_decomp = svd.fit_transform(centered)
```

Кластеризация по семантике

```
kmeans = KMeans(n_clusters=25, random_state=42)
```

```
cluster = kmeans.fit_predict(svd_decomp).reshape(-1, 1)
```

```
posts_info["TextCluster"] = cluster
```

Генерация названий столбцов динамически

```
num_clusters = 25 # Количество кластеров
```

```
dists_columns = [  
    f"DistanceTo{cluster}thCluster" for cluster in range(1,  
    num_clusters + 1)  
]
```

Создание DataFrame с автоматически сгенерированными названиями столбцов

```
dists_df = pd.DataFrame(data=kmeans.transform(svd_decomp),  
    columns=dists_columns)
```

Проверка первых строк

```
dists_df.head()
```

	DistanceTo1thCluster	DistanceTo2thCluster	DistanceTo3thCluster	\
0	0.620717	0.639040	0.710284	
1	0.609376	0.632340	0.684434	
2	0.725162	0.744982	0.766279	
3	0.714635	0.730692	0.802663	
4	0.531889	0.549947	0.657950	

	DistanceTo4thCluster	DistanceTo5thCluster	DistanceTo6thCluster	\
--	----------------------	----------------------	----------------------	---

0	0.592390	0.703258	0.632899
1	0.589731	0.675975	0.621455
2	0.701997	0.740468	0.729460
3	0.705589	0.798022	0.725473
4	0.520446	0.659838	0.542882
	DistanceTo7thCluster	DistanceTo8thCluster	DistanceTo9thCluster \
0	0.942256	0.635237	0.436462
1	0.916474	0.558346	0.516970
2	1.024672	0.754167	0.509004
3	1.013979	0.732404	0.621708
4	0.898131	0.574709	0.528084
	DistanceTo10thCluster ... DistanceTo16thCluster		
DistanceTo17thCluster \			
0	0.761016 ... 0.828193		
0.757815			
1	0.739384 ... 0.820065		
0.728541			
2	0.814646 ... 0.901114		
0.803224			
3	0.843584 ... 0.859086		
0.837872			
4	0.704511 ... 0.762391		
0.708491			
	DistanceTo18thCluster	DistanceTo19thCluster	DistanceTo20thCluster
\			
0	0.671149	0.611784	0.581796
1	0.661025	0.587971	0.557299
2	0.772441	0.668490	0.658415
3	0.749451	0.716179	0.683872
4	0.591135	0.554863	0.506563
	DistanceTo21thCluster	DistanceTo22thCluster	DistanceTo23thCluster
\			
0	0.584506	0.587526	0.540042
1	0.577983	0.579828	0.512105
2	0.700094	0.685362	0.642903
3	0.682313	0.680886	0.633085
4	0.484730	0.524628	0.448073

	DistanceTo24thCluster	DistanceTo25thCluster
0	0.624294	0.702854
1	0.646764	0.703910
2	0.629109	0.773713
3	0.789145	0.804049
4	0.686375	0.635861

[5 rows x 25 columns]

```
posts_info = pd.concat((posts_info, dists_df), axis=1).drop(
    columns=["cleaned_text"], axis=1
)
```

```
post_statistics = pd.read_sql(
    """
    WITH prep AS (
        SELECT
            timestamp,
            user_id,
            post_id,
            action,
            target,
            MIN(timestamp) FILTER (WHERE target = 1) OVER (PARTITION
BY user_id, post_id) AS min_timestamp,
            CASE
                WHEN timestamp > MIN(timestamp) FILTER (WHERE target =
1) OVER (PARTITION BY user_id, post_id)
                THEN 1
                ELSE 0
            END AS check
        FROM feed_data
        WHERE action <> 'like'
    )
    SELECT
        post_id,
        COUNT(*) FILTER (WHERE target = 0) AS total_views_post,
        COUNT(*) FILTER (WHERE target = 1) AS total_likes_post,
        COUNT(*) FILTER (WHERE target = 1) * 1.0 / NULLIF(COUNT(*), 0)
    AS post_ctr
    FROM prep
    WHERE "check" = 0
    GROUP BY post_id
    """,
    con=engine,
)
```

Заполняем медианами посты без статистики

```
views_median = post_statistics["total_views_post"].median()
likes_median = post_statistics["total_likes_post"].median()
```

```
post_statistics["total_views_post"].fillna(views_median, inplace=True)
post_statistics["total_likes_post"].fillna(likes_median, inplace=True)
```

Пересчитываем CTR

```
post_statistics["post_ctr"] = (
    post_statistics["total_likes_post"] /
    post_statistics["total_views_post"]
)
```

```
post_statistics.head()
```

	post_id	total_views_post	total_likes_post	post_ctr
0	1	7414	1057	0.142568
1	2	6851	636	0.092833
2	3	7263	1116	0.153656
3	4	6964	1165	0.167289
4	5	7400	1141	0.154189

Соединяем нашу таблицу со статистическими признаками

```
posts_info = posts_info.join(
    post_statistics.set_index("post_id"), on="post_id", how="left"
)
```

```
posts_info.head()
```

	post_id	text
topic \		
0	1	UK economy facing major risks\n\nThe UK manufa...
business		
1	2	Aids and climate top Davos agenda\n\nClimate c...
business		
2	3	Asian quake hits European shares\n\nShares in ...
business		
3	4	India power shares jump on debut\n\nShares in ...
business		
4	5	Lacroix label bought by US firm\n\nLuxury good...
business		

	numbers_count	text_length	word_count	TotalTfIdf	MaxTfIdf
MeanTfIdf \					
0	13	1967	327	8.134297	0.511790
0.000541					
1	20	2701	449	11.356141	0.254945
0.000756					
2	47	3408	571	10.911138	0.410319
0.000726					
3	6	1026	178	6.754817	0.372043
0.000450					
4	3	889	155	5.946105	0.526337

0.000396

	TextCluster	...	DistanceTo19thCluster	DistanceTo20thCluster	\
0	8	...	0.611784	0.581796	
1	22	...	0.587971	0.557299	
2	8	...	0.668490	0.658415	
3	8	...	0.716179	0.683872	
4	22	...	0.554863	0.506563	

	DistanceTo21thCluster	DistanceTo22thCluster	DistanceTo23thCluster	\
0	0.584506	0.587526	0.540042	
1	0.577983	0.579828	0.512105	
2	0.700094	0.685362	0.642903	
3	0.682313	0.680886	0.633085	
4	0.484730	0.524628	0.448073	

	DistanceTo24thCluster	DistanceTo25thCluster	total_views_post	\
0	0.624294	0.702854	7414	
1	0.646764	0.703910	6851	
2	0.629109	0.773713	7263	
3	0.789145	0.804049	6964	
4	0.686375	0.635861	7400	

	total_likes_post	post_ctr
0	1057	0.142568
1	636	0.092833
2	1116	0.153656
3	1165	0.167289
4	1141	0.154189

[5 rows x 38 columns]

Загружаем таблица со всеми признаками постов в базу данных

```
posts_info.to_sql(  
    "posts_info_features_ruslan_prashchurovich",  
    con=engine,  
    index=False,  
    if_exists="replace",  
)
```

143

Функция для пакетной загрузки данных из SQL

```
def batch_load_sql(query: str, engine) -> pd.DataFrame:
```



```

CHUNKSIZE = 200000
conn = engine.connect().execution_options(stream_results=True)
chunks = []
for chunk_dataframe in pd.read_sql(query, conn,
chunksize=CHUNKSIZE):
    chunks.append(chunk_dataframe)
conn.close()
return pd.concat(chunks, ignore_index=True)

# Функция для загрузки признаков из базы данных
def load_posts_features(engine) -> pd.DataFrame:
    query = "SELECT * FROM posts_info_features_ruslan_prashchurovich"
    return batch_load_sql(query, engine)

```

```

# Проверка загрузки данных
post_features = load_posts_features(engine)
post_features.head()

```

	post_id	text
topic \		
0	1	UK economy facing major risks\n\nThe UK manufa...
business		
1	2	Aids and climate top Davos agenda\n\nClimate c...
business		
2	3	Asian quake hits European shares\n\nShares in ...
business		
3	4	India power shares jump on debut\n\nShares in ...
business		
4	5	Lacroix label bought by US firm\n\nLuxury good...
business		

	numbers_count	text_length	word_count	TotalTfIdf	MaxTfIdf
MeanTfIdf \					
0	13	1967	327	8.134297	0.511790
0.000541					
1	20	2701	449	11.356141	0.254945
0.000756					
2	47	3408	571	10.911138	0.410319
0.000726					
3	6	1026	178	6.754817	0.372043
0.000450					
4	3	889	155	5.946105	0.526337
0.000396					

	TextCluster	...	DistanceTo19thCluster	DistanceTo20thCluster	\
0	8	...	0.611784	0.581796	
1	22	...	0.587971	0.557299	
2	8	...	0.668490	0.658415	

3	8	...	0.716179	0.683872
4	22	...	0.554863	0.506563

	DistanceTo21thCluster	DistanceTo22thCluster	DistanceTo23thCluster
\			
0	0.584506	0.587526	0.540042
1	0.577983	0.579828	0.512105
2	0.700094	0.685362	0.642903
3	0.682313	0.680886	0.633085
4	0.484730	0.524628	0.448073

	DistanceTo24thCluster	DistanceTo25thCluster	total_views_post	\
0	0.624294	0.702854	7414	
1	0.646764	0.703910	6851	
2	0.629109	0.773713	7263	
3	0.789145	0.804049	6964	
4	0.686375	0.635861	7400	

	total_likes_post	post_ctr
0	1057	0.142568
1	636	0.092833
2	1116	0.153656
3	1165	0.167289
4	1141	0.154189

[5 rows x 38 columns]

Почистим переменные

```
del svd
del KMeans
del spacy_preprocess

del tfidf_spacy_matrix
del tfidf_spacy_df
del centered
del svd_decomp
del kmeans
del dists_columns
del dists_df
```

```
gc.collect()
```

728

Feature Engineering для таблицы `user_data`

□ План генерации признаков пользователей

1. Возрастные группы

Генерируемые фичи:

- `age_group` (категориальный признак)

Группировка:

- `child`: 0-18 лет
- `young`: 18-35 лет
- `middle_aged`: 35-50 лет
- `senior`: 50+ лет

Гипотеза:

Разные возрастные группы демонстрируют различное поведение при взаимодействии с контентом. Например:

- Молодежь более активна с коротким форматом
 - Старшая возрастная группа предпочитает более сложный контент
-

2. Показатели активности пользователя

Источник данных: таблицы `feed_data` + `post_text_df`

Генерируемые фичи:

- `user_total_views` - общее количество просмотров
- `user_total_likes` - общее количество лайков
- `user_ctr` - отношение лайков к просмотрам (лайки/просмотры)
- `favorite_topic` - самая частая тема лайкнутых постов
- `mean_length_view` - средняя длина просмотренных текстов
- `mean_length_number_view` - среднее количество цифр в просмотренных текстах
- `mean_length_like` - средняя длина лайкнутых текстов
- `mean_length_number_like` - среднее количество цифр в лайкнутых текстах

Гипотезы:

Признак	Гипотеза	Ожидаемый эффект
<code>user_ctr</code>	Пользователи с высоким CTR более избирательны	Поможет выделить "качественную" аудиторию
<code>mean_length_view</code>	Пользователи имеют предпочтения по длине контента	Позволит персонализировать рекомендации по объему текста
<code>mean_length</code>	Лайкают посты определенной	Выявит оптимальную длину

Признак	Гипотеза	Ожидаемый эффект
<code>_like</code>	длины	контента для вовлечения
<code>mean_length</code> <code>_number_view</code>	Цифры в тексте привлекают внимание	Поможет определить интерес к аналитическому контенту
<code>favorite_to</code> <code>pic</code>	У пользователей есть тематические предпочтения	Улучшит тематическую персонализацию

Определяем функцию для категоризации возраста (идея: разные категории возраста по-разному взаимодействуют с постами)

```
def categorize_age(age):
    if age < 18:
        return "child"
    elif 18 <= age < 35:
        return "young"
    elif 35 <= age < 50:
        return "middle_aged"
    else:
        return "senior"
```

Применяем функцию к столбцу age

```
user_info["age_category"] = user_info["age"].apply(categorize_age)
user_info = user_info.drop(columns=["age"], axis=1)
user_info.head()
```

	user_id	gender	country	city	exp_group	os
source \						
0	200	1	Russia	Degtyarsk	3	Android
ads						
1	201	0	Russia	Abakan	0	Android
ads						
2	202	1	Russia	Smolensk	4	Android
ads						
3	203	0	Russia	Moscow	1	iOS
ads						
4	204	0	Russia	Anzhero-Sudzhensk	3	Android
ads						

	age_category
0	young
1	middle_aged
2	child
3	young
4	middle_aged

Объединение данных

```
merged_data = feed_data.merge(
    user_info, on="user_id", how="left"
).merge(# Добавляем информацию о пользователях
```

```

    posts_info, on="post_id", how="left"
) # Добавляем информацию о постах

# Проверяем, что данные объединились корректно
merged_data.head()

```

	timestamp	user_id	post_id	action	target	gender	country
0	2021-10-11 20:45:10	55365	1650	view	1	1	Turkey
1	2021-10-11 20:45:41	55365	956	view	0	1	Turkey
2	2021-10-11 20:47:36	55365	1875	view	1	1	Turkey
3	2021-10-11 20:50:15	55365	1039	view	0	1	Turkey
4	2021-10-11 20:52:11	55365	2571	view	0	1	Turkey

	city	exp_group	os	source	age_category
0	Istanbul	0	Android	ads	middle_aged
1	Istanbul	0	Android	ads	middle_aged
2	Istanbul	0	Android	ads	middle_aged
3	Istanbul	0	Android	ads	middle_aged
4	Istanbul	0	Android	ads	middle_aged

	text	topic
0	Klinsmann issues Lehmann warning\n\nGermany co...	sport
1	Clarke to press on with ID cards\n\nNew Home S...	politics
2	Robben and Cole earn Chelsea win\n\nCheslea sa...	sport
3	Tsunami debt deal to be announced\n\nChancello...	politics
4	Health rights activists are calling for massiv...	covid

Анализ предпочтений по возрастным категориям

Теперь сгруппируем данные по age_category и topic, чтобы посчитать, сколько раз пользователи каждой возрастной категории взаимодействовали с постами на разные темы.

```

# Группировка данных
age_topic_preferences = (
    merged_data.groupby(["age_category", "topic"])[
        "action"
    ] # Группируем по возрасту и теме
    .count() # Считаем количество действий
    .reset_index()
    .rename(columns={"action": "count"}) # Переименовываем столбец
    для удобства
)

```

```
# Добавляем нормализованные значения (доля взаимодействий)
total_actions_by_age = age_topic_preferences.groupby("age_category")
["count"].transform(
    "sum"
)
age_topic_preferences["normalized_count"] = (
    age_topic_preferences["count"] / total_actions_by_age
)
```

```
# Проверяем результат
```

```
age_topic_preferences.head()
```

	age_category	topic	count	normalized_count
0	child	business	32782	0.045862
1	child	covid	178952	0.250352
2	child	entertainment	27465	0.038423
3	child	movie	290588	0.406530
4	child	politics	63801	0.089257

```
# Создаем график
```

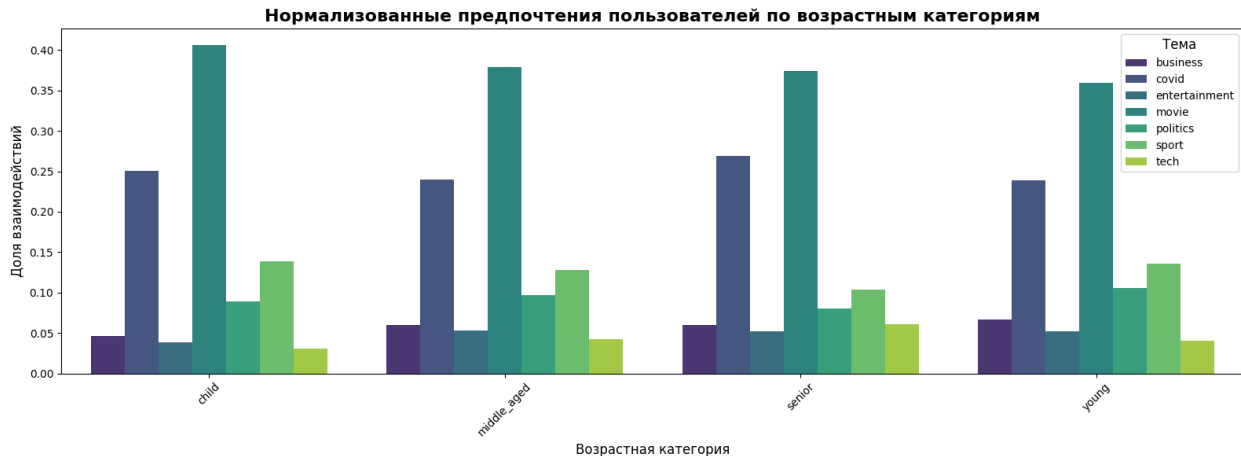
```
plt.figure(figsize=(16, 6))
sns.barplot(
    data=age_topic_preferences,
    x="age_category",
    y="normalized_count",
    hue="topic",
    palette="viridis",
)
```

```
# Добавляем заголовок и подписи
```

```
plt.title(
    "Нормализованные предпочтения пользователей по возрастным категориям",
    fontsize=16,
    fontweight="bold",
)
plt.xlabel("Возрастная категория", fontsize=12)
plt.ylabel("Доля взаимодействий", fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.legend(title="Тема", title_fontsize=12, fontsize=10)
```

```
# Отображаем график
```

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



Несмотря на первоначальную гипотезу о том, что пользователи разных возрастных категорий могут демонстрировать значительные различия в интересах, проведенный анализ показал интересный результат:

В целом, доля каждого топика внутри каждой возрастной категории оказалась практически идентичной.

Это означает, что независимо от возраста пользователи взаимодействуют с темами контента схожим образом. Например, популярные темы, такие как `movie`, `politics` или `tech`, занимают примерно одинаковую долю взаимодействий в группах "18–24", "25–34" и других возрастных категориях.

Такая однородность может быть связана с тем, что платформа предлагает универсальный контент, который привлекателен для широкой аудитории, либо с тем, что интересы пользователей менее вариативны, чем мы предполагали.

Однако мы все равно оставим эту фику для дальнейшего анализа.

Хотя различия между возрастными категориями минимальны, даже небольшие отклонения могут оказаться полезными при построении более точных рекомендательных моделей. В процессе машинного обучения такие фики могут выявить скрытые паттерны, которые не всегда очевидны при поверхностном анализе.

Итог

- **Основной вывод:** Доля взаимодействий с темами практически одинакова для всех возрастных категорий.
- **Практическое решение:** Несмотря на это, фича `age_category` остается потенциально полезной для моделирования, так как даже незначительные различия могут улучшить качество рекомендаций.

Следующим шагом будет интеграция этой фики в общий пайплайн и проверка её влияния на метрики модели. □

Поскольку таблица очень большая, посчитаем все метрики прямо в БД

```
users_statistics = pd.read_sql(
    """
    WITH prep AS (
        SELECT
            timestamp,
            user_id,
            post_id,
            action,
            target,
            MIN(timestamp) FILTER (WHERE target = 1) OVER (PARTITION
BY user_id, post_id) AS min_timestamp,
            CASE
                WHEN timestamp > MIN(timestamp) FILTER (WHERE target =
1) OVER (PARTITION BY user_id, post_id)
                THEN 1
                ELSE 0
            END AS check
        FROM feed_data
        WHERE action <> 'like'
    )
    SELECT
        user_id,
        COUNT(*) FILTER (WHERE target = 0) AS user_total_views,
        COUNT(*) FILTER (WHERE target = 1) AS user_total_likes,
        COUNT(*) FILTER (WHERE target = 1) * 1.0 / NULLIF(COUNT(*), 0)
AS user_ctr
    FROM prep
    WHERE "check" = 0
    GROUP BY user_id
    """
    ,
    con=engine,
)
users_statistics.head()
```

	user_id	user_total_views	user_total_likes	user_ctr
0	200	315	43	0.120112
1	201	632	58	0.084058
2	202	541	87	0.138535
3	203	258	61	0.191223
4	204	115	23	0.166667

Найдем для каждого пользователя его любимый топик по следующей логике:

- посчитаем для каждого пользователя для каждого поста кол-во просмотров, лайков, ctr

- взвесим все три показателя

```
user_favorite_topic = pd.read_sql(
```



```

""
WITH prep AS (
    SELECT
        timestamp,
        user_id,
        post_id,
        action,
        target,
        MIN(timestamp) FILTER (WHERE target = 1) OVER (PARTITION
BY user_id, post_id) AS min_timestamp,
        CASE
            WHEN timestamp > MIN(timestamp) FILTER (WHERE target =
1) OVER (PARTITION BY user_id, post_id)
            THEN 1
            ELSE 0
        END AS check
    FROM feed_data
    WHERE action <> 'like'
),
grouped AS (
    SELECT
        user_id,
        topic,
        COUNT(*) FILTER (WHERE target = 0) AS user_total_views,
        COUNT(*) FILTER (WHERE target = 1) AS user_total_likes,
        COUNT(*) FILTER (WHERE target = 1) * 1.0 /
NULLIF(COUNT(*), 0) AS user_ctr
    FROM prep
    INNER JOIN post_text_df ON prep.post_id = post_text_df.post_id
    WHERE "check" = 0
    GROUP BY user_id, topic
),
normalized AS (
    SELECT
        user_id,
        topic,
        (user_total_likes - MIN(user_total_likes) OVER ())
        /
NULLIF((MAX(user_total_likes) OVER () - MIN(user_total_likes) OVER
()), 0) AS normalized_likes,
        (user_total_views - MIN(user_total_views) OVER ())
        /
NULLIF((MAX(user_total_views) OVER () - MIN(user_total_views) OVER
()), 0) AS normalized_views,
        (user_ctr - MIN(user_ctr) OVER ()) / NULLIF((MAX(user_ctr)
OVER () - MIN(user_ctr) OVER ()), 0) AS normalized_ctr
    FROM grouped
),
combined_scores AS (

```

```

        SELECT
            user_id,
            topic,
            0.4 * normalized_likes + 0.3 * normalized_views + 0.3 *
normalized_ctr AS combined_score
        FROM normalized
    ),
    ranked_topics AS (
        SELECT
            user_id,
            topic,
            combined_score,
            ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
combined_score DESC) AS rank
        FROM combined_scores
    )
    SELECT
        user_id,
        topic AS favorite_topic,
        combined_score
    FROM ranked_topics
    WHERE rank = 1;
    """
    con=engine,
)

```

Вывод результата

```
user_favorite_topic.head()
```

	user_id	favorite_topic
0	200	entertainment
1	201	tech
2	202	politics
3	203	sport
4	204	tech

1. Распределение популярности тем: (видим, что спорт, политика и развлечения составляют ТОП-3 самых популярных тем постов)

```

topic_distribution = (
    user_favorite_topic["favorite_topic"].value_counts(normalize=True)
* 100
)

```

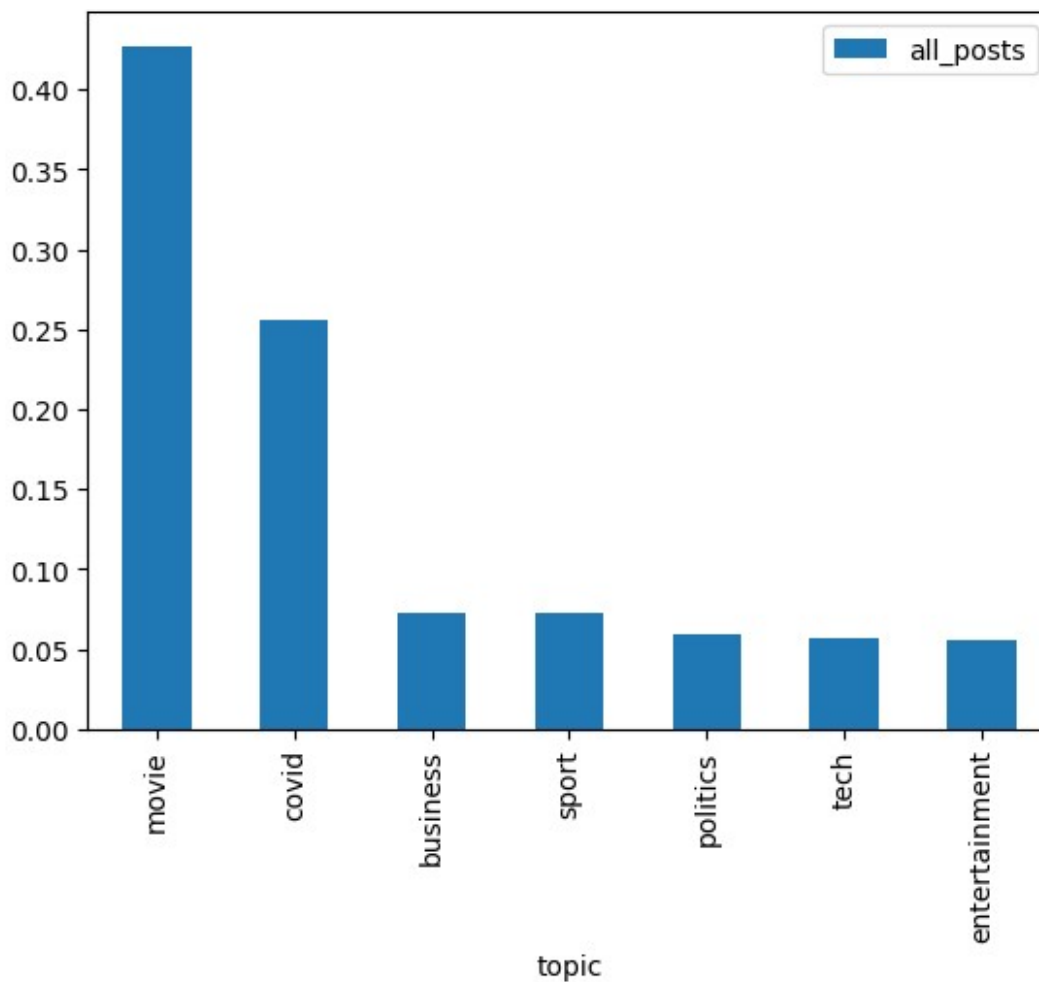
```
topic_distribution
```

favorite_topic	
sport	22.077755
politics	17.549095
entertainment	17.332190

```
business      13.452406
movie         12.068870
tech          11.666922
covid         5.852762
Name: proportion, dtype: float64
```

```
# Доля тем во всех постах
```

```
all_topics = posts_info["topic"].value_counts(normalize=True)
pd.DataFrame({"all_posts": all_topics}).plot.bar()
```



Анализ распределения тем в постах и предпочтений пользователей

На основе анализа данных можно сделать важные наблюдения о различиях между общим распределением тем в постах и предпочтениями пользователей (их любимыми темами):

1. Доминирование тем в постах

- В общей массе постов наибольшую долю занимают темы **movie** и **covid**, которые вместе составляют почти **70%** всего контента.
- Это указывает на то, что данные темы являются наиболее популярными или часто публикуемыми на платформе.

2. Предпочтения пользователей

- Любимые темы пользователей распределены иначе: здесь лидируют такие категории, как **sport**, **politics** и **entertainment**, каждая из которых занимает около **17–22%** предпочтений.
- Темы **movie** и **covid**, несмотря на их доминирование в общем пуле постов, занимают значительно меньшую долю в предпочтениях пользователей (**12% и 5.8% соответственно**).

3. Ключевой инсайт

- Наблюдается явное **несоответствие** между тем, что публикуется на платформе, и тем, что предпочитают пользователи.
- Например, тема **movie** составляет **42.7%** всех постов, но только **12%** пользователей считают её своей любимой темой. Аналогично, **covid** занимает **25.6%** постов, но лишь **5.8%** пользователей выбирают её как приоритетную.

4. Возможные рекомендации

- Для улучшения пользовательского опыта стоит пересмотреть стратегию создания контента: увеличить долю тем, которые действительно популярны среди пользователей (например, **sport**, **politics**, **entertainment**), и сбалансировать количество постов по менее востребованным темам (**movie**, **covid**).
 - Это может повысить вовлеченность пользователей и сделать платформу более релевантной их интересам.
-

Итог:

Несоответствие между распределением тем в постах и предпочтениями пользователей подчеркивает необходимость перераспределения усилий на создание контента, который будет лучше соответствовать интересам аудитории. □

```
user_info = user_info.join(
    users_statistics.set_index("user_id"), on="user_id", how="left"
)
```

```

user_info = user_info.join(
user_favorite_topic.set_index("user_id").drop(columns=["combined_score
"], axis=1),
    on="user_id",
    how="left",
)

# Также найдем любимый час для лайка
user_favorite_hour = pd.read_sql(
    """
    WITH prep AS (
        SELECT
            timestamp,
            user_id,
            post_id,
            action,
            target,
            MIN(timestamp) FILTER (WHERE target = 1) OVER (PARTITION
BY user_id, post_id) AS min_timestamp,
            CASE
                WHEN timestamp > MIN(timestamp) FILTER (WHERE target =
1) OVER (PARTITION BY user_id, post_id)
                THEN 1
                ELSE 0
            END AS check
        FROM feed_data
        WHERE action <> 'like'
    ),
    grouped AS (
        SELECT
            user_id,
            EXTRACT(HOUR FROM timestamp) AS hour_of_day,
            COUNT(*) FILTER (WHERE target = 0) AS total_views,
            COUNT(*) FILTER (WHERE target = 1) AS total_likes,
            COUNT(*) FILTER (WHERE target = 1) * 1.0 /
NULLIF(COUNT(*), 0) AS ctr
        FROM prep
        WHERE "check" = 0
        GROUP BY user_id, EXTRACT(HOUR FROM timestamp)
    ),
    normalized AS (
        SELECT
            user_id,
            hour_of_day,
            (total_likes - MIN(total_likes) OVER ()) /
NULLIF((MAX(total_likes) OVER () - MIN(total_likes) OVER ()), 0) AS
normalized_likes,
            (total_views - MIN(total_views) OVER ()) /

```

```

NULLIF((MAX(total_views) OVER () - MIN(total_views) OVER ()), 0) AS
normalized_views,
        (ctr - MIN(ctr) OVER ()) / NULLIF((MAX(ctr) OVER () -
MIN(ctr) OVER ()), 0) AS normalized_ctr
    FROM grouped
),
combined_scores AS (
    SELECT
        user_id,
        hour_of_day,
        0.35 * normalized_likes + 0.15 * normalized_views + 0.5 *
normalized_ctr AS combined_score
    FROM normalized
),
ranked_hours AS (
    SELECT
        user_id,
        hour_of_day,
        combined_score,
        ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
combined_score DESC) AS rank
    FROM combined_scores
)
SELECT
    user_id,
    hour_of_day AS favorite_hour,
    combined_score
FROM ranked_hours
WHERE rank = 1;
"""
con=engine,
)
user_favorite_hour.head()

```

	user_id	favorite_hour
0	200	8
1	201	22
2	202	22
3	203	16
4	204	15

Анализ распределения любимых часов

```

favorite_hour_distribution = (
    user_favorite_hour["favorite_hour"].value_counts().sort_index()
)

```

Создаем график

```

plt.figure(figsize=(12, 6))
sns.barplot(
    x=favorite_hour_distribution.index,

```

```

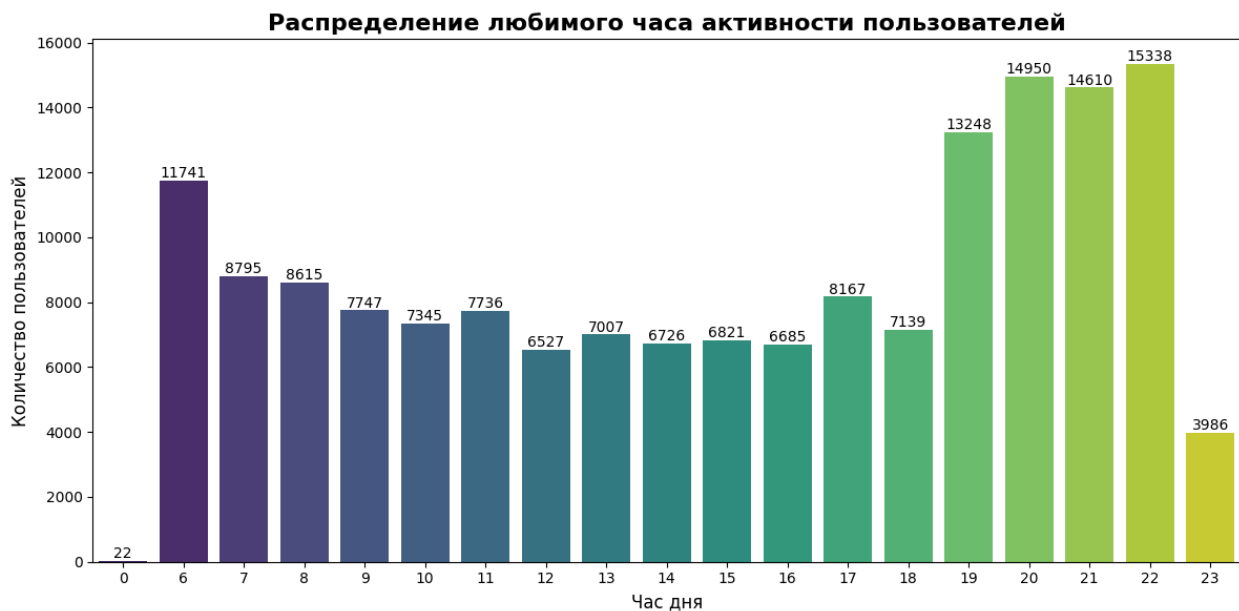
    y=favorite_hour_distribution.values,
    palette="viridis",
)

# Добавляем заголовок и подписи
plt.title(
    "Распределение любимого часа активности пользователей",
    fontsize=16,
    fontweight="bold",
)
plt.xlabel("Час дня", fontsize=12)
plt.ylabel("Количество пользователей", fontsize=12)

# Добавляем значения над столбцами
for i, value in enumerate(favorite_hour_distribution):
    plt.text(i, value + 90, f"{value}", ha="center", fontsize=10,
color="black")

# Отображаем график
plt.tight_layout()
plt.show()

```



Анализ любимого часа активности пользователей

На основе распределения любимых часов активности пользователей можно выделить несколько ключевых паттернов поведения:

1. Пик активности приходится на вечернее время

- Наибольшее количество пользователей предпочитают быть активными в промежутке с **19:00 до 22:00**. Особенно выделяются часы **20:00 (14,950 пользователей)** и **21:00 (14,610 пользователей)**, а также **22:00 (15,338 пользователей)**.
 - Это типично для платформ, где пользователи взаимодействуют в свободное время после работы или учебы.
2. **Утренний подъем активности**
 - Начиная с **6:00 утра**, наблюдается резкий рост активности, достигающий максимума в **9:00 (7,747 пользователей)**. Это может быть связано с началом рабочего дня или утренними рутинами пользователей.
 3. **Снижение активности в ночные часы**
 - Ночные часы, особенно **0:00–5:00**, демонстрируют минимальную активность. Например, в полночь (**0:00**) активны всего **22 пользователя**, а к **3:00–5:00** активность практически отсутствует. Это соответствует естественному ритму жизни большинства людей.
 4. **Дневной спад**
 - В дневное время (**12:00–16:00**) активность остается относительно стабильной, но ниже вечерних пиков. Возможно, это связано с занятостью пользователей в течение рабочего дня.
-

Ключевые инсайты

- **Вечерний прайм-тайм:** Платформа может использовать временные интервалы с **19:00 до 22:00** для запуска маркетинговых кампаний, рассылок или публикации контента, чтобы максимизировать охват и вовлеченность.
 - **Утренний потенциал:** Утренние часы (**6:00–9:00**) также представляют собой важный период для взаимодействия с пользователями, особенно если целевая аудитория включает работающих людей.
 - **Ночные часы:** Низкая активность в ночные часы указывает на то, что этот период менее перспективен для взаимодействия с аудиторией.
-

Итог:

Распределение любимого часа активности пользователей демонстрирует четкие паттерны поведения, которые можно использовать для оптимизации взаимодействия с аудиторией.

□

Также попробуем посчитать среднюю длину текста, которые пользователи любят смотреть и лайкать

```
user_post_length = pd.read_sql(  
    """  
    WITH post_stats AS (  
        SELECT  
            post_id,  
            LENGTH(text) AS text_length,  
    """
```



```

        (LENGTH(text) - LENGTH(REGEXP_REPLACE(text, '[0-9]', '',
'g')))) AS number_count
        FROM post_text_df
    ),
    user_actions AS (
        SELECT
            user_id,
            post_id,
            action,
            target
        FROM feed_data
        WHERE action <> 'like'
    ),
    aggregated_stats AS (
        SELECT
            ua.user_id,
            AVG(ps.text_length) FILTER (WHERE ps.post_id IS NOT NULL)
AS mean_length_view,
            AVG(ps.number_count) FILTER (WHERE ps.post_id IS NOT NULL)
AS mean_length_number_view,
            AVG(ps.text_length) FILTER (WHERE ua.target = 1) AS
mean_length_like,
            AVG(ps.number_count) FILTER (WHERE ua.target = 1) AS
mean_length_number_like
        FROM user_actions ua
        LEFT JOIN post_stats ps ON ua.post_id = ps.post_id
        GROUP BY ua.user_id
    )
    SELECT
        user_id,
        mean_length_view,
        mean_length_number_view,
        mean_length_like,
        mean_length_number_like
    FROM aggregated_stats;
""",
con=engine,
)

user_info = user_info.join(
    user_post_length.set_index("user_id"), on="user_id", how="left"
)

# Почистим переменные

del user_post_length
del favorite_hour_distribution
del user_favorite_hour

del all_topics

```

```

del topic_distribution
del user_favorite_topic
del users_statistics
del age_topic_preferences
del merged_data

gc.collect()

# Загружаем таблица со всеми признаками юзеров в базу данных
user_info.to_sql(
    "users_info_features_ruslan_prashchurovich",
    con=engine,
    index=False,
    if_exists="replace",
)

205

# Функция для загрузки признаков юзеров из базы данных
def load_users_features(engine) -> pd.DataFrame:
    query = "SELECT * FROM users_info_features_ruslan_prashchurovich"
    return batch_load_sql(query, engine)

# Проверка загрузки данных
users_features = load_users_features(engine)
users_features.head()

```

	user_id	gender	country	city	exp_group	os
source \						
0	200	1	Russia	Degtyarsk	3	Android
ads						
1	201	0	Russia	Abakan	0	Android
ads						
2	202	1	Russia	Smolensk	4	Android
ads						
3	203	0	Russia	Moscow	1	iOS
ads						
4	204	0	Russia	Anzhero-Sudzhensk	3	Android
ads						

	age_category	user_total_views	user_total_likes	user_ctr
favorite_topic \				
0	young	315	43	0.120112
entertainment				
1	middle_aged	632	58	0.084058
tech				
2	child	541	87	0.138535
politics				
3	young	258	61	0.191223

```
sport
4 middle_aged      115      23  0.166667
tech
```

```
    favorite_hour  mean_length_view  mean_length_number_view
mean_length_like \
0      8      1798.600000      8.133333
1439.400000
1      22      1636.866667      3.800000
1984.266667
2      22      449.266667      3.133333
1095.600000
3      16      1053.800000      2.933333
1898.333333
4      15      1441.266667      4.333333
1104.066667
```

```
    mean_length_number_like
0      5.200000
1      3.666667
2      3.600000
3      4.800000
4      4.600000
```