

Подготовка среды выполнения, нужно выполнить один раз и не запускать при повторном выполнении.

```
!git clone https://github.com/priorlabs/tabPFN-extensions.git
```

```
→ Cloning into 'tabPFN-extensions'...
remote: Enumerating objects: 1923, done.
remote: Counting objects: 100% (229/229), done.
remote: Compressing objects: 100% (114/114), done.
remote: Total 1923 (delta 141), reused 129 (delta 110), pack-reused 1694 (from 2)
Receiving objects: 100% (1923/1923), 748.78 KiB | 6.45 MiB/s, done.
Resolving deltas: 100% (1162/1162), done.
```

```
pip install -e tabPFN-extensions
```

```
→ Downloading nvidia_cuda_cupti_cu12_12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB) 13.8/13.8 MB 68.0 MB/s eta 0:00:00
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB) 24.6/24.6 MB 60.0 MB/s eta 0:00:00
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB) 883.7/883.7 kB 53.2 MB/s eta 0:00:00
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB) 664.8/664.8 MB 2.4 MB/s eta 0:00:00
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB) 211.5/211.5 MB 10.8 MB/s eta 0:00:00
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB) 56.3/56.3 MB 33.6 MB/s eta 0:00:00
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB) 127.9/127.9 MB 10.0 MB/s eta 0:00:00
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB) 207.5/207.5 MB 3.7 MB/s eta 0:00:00
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB) 21.1/21.1 MB 53.5 MB/s eta 0:00:00
Building wheels for collected packages: tabPFN-extensions
  Building editable for tabPFN-extensions (pyproject.toml) ... done
  Created wheel for tabPFN-extensions: filename=tabPFN_extensions-0.1.0-0.editable-py3-none-any.whl size=11667 sha256=d6f9132a83ff4473debda28f51
  Stored in directory: /tmp/pip-ephem-wheel-cache-d03wyk23/wheels/8f/b7/2c/ec18b502697068b39e1503426a50b729bdac046bed4810759d
Successfully built tabPFN-extensions
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, n
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    .....
```

```
import os
os.kill(os.getpid(), 9)
```

## Исследование датасета с данными о продлении/отказе от продления полисов ОСАГО

```
'запуск кода при перезапуске среды с этой ячейки Ctrl+F10'
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
%matplotlib inline
import nltk
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

Загружаем датасет

```
df=pd.read_csv('Data.txt', sep=';')
#чтобы правильно открыть разделим с помощью сер где знаки=;
```

```
df.shape
```

```
→ (96605, 30)
```

```
df.head(10)
```

	DATA_TYPE	POLICY_ID	POLICY_BEGIN_MONTH	POLICY_END_MONTH	POLICY_IS_RENEWED	POLICY_SALES_CHANNEL	POLICY_SALES_CHANNEL_GROUP	POLICY_BRANCH
0	TRAIN	1	1	1	1	39		1 Москва
1	TRAIN	2	1	1	1	50		5 Москва
2	TRAIN	3	1	1	1	52		6 Москва
3	TRAIN	4	1	1	1	50		5 Москва
4	TRAIN	5	1	1	0	52		6 Санкт-Петербург
5	TRAIN	6	2	1	1	2		4 Санкт-Петербург
6	TRAIN	7	1	1	1	52		6 Москва
7	TRAIN	8	2	2	1	10		1 Санкт-Петербург
8	TRAIN	9	1	1	0	53		6 Москва
9	TEST	10	2	2	0	53		6 Санкт-Петербург

10 rows × 30 columns

```
count_values = df['DATA_TYPE'].value_counts()
print(count_values)
```

```
→ DATA_TYPE
TRAIN    77407
TEST     19198
Name: count, dtype: int64
```

Так как в тестовом наборе все значения целевого признака искусственно обнулены, удалим эту часть из датасета, использование ее для обучения моделей будет некорректным

```
df = df.loc[df['DATA_TYPE'] != 'TEST ']
```

```
df=df.drop(['DATA_TYPE'], axis=1)
```

```
df.duplicated().sum()
```

```
→ np.int64(0)
```

Дубликаты в датасете отсутствуют

```
df.info()
```

```
→ <class 'pandas.core.frame.DataFrame'>
Index: 77407 entries, 0 to 96604
Data columns (total 29 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   POLICY_ID        77407 non-null   int64  
 1   POLICY_BEGIN_MONTH 77407 non-null   int64  
 2   POLICY_END_MONTH  77407 non-null   int64  
 3   POLICY_IS_RENEWED 77407 non-null   int64  
 4   POLICY_SALES_CHANNEL 77407 non-null   int64  
 5   POLICY_SALES_CHANNEL_GROUP 77407 non-null   int64  
 6   POLICY_BRANCH      77407 non-null   object 
 7   POLICY_MIN_AGE     77407 non-null   int64  
 8   POLICY_MIN_DRIVING_EXPERIENCE 77407 non-null   int64  
 9   VEHICLE_MAKE       77407 non-null   object 
 10  VEHICLE_MODEL      77407 non-null   object 
 11  VEHICLE_ENGINE_POWER 77407 non-null   float64 
 12  VEHICLE_IN_CREDIT   77407 non-null   int64  
 13  VEHICLE_SUM_INSURED 77407 non-null   float64 
 14  POLICY_INTERMEDIARY 77407 non-null   object 
 15  INSURER_GENDER      77407 non-null   object 
 16  POLICY_CLM_N        77407 non-null   object 
 17  POLICY_CLM_GLT_N    77407 non-null   object 
 18  POLICY_PRV_CLM_N    77407 non-null   object 
```

```

19 POLICY_PRV_CLM_GLT_N    77407 non-null object
20 CLIENT_HAS_DAGO          77407 non-null int64
21 CLIENT_HAS_OSAGO         77407 non-null int64
22 POLICY_COURT_SIGN        77407 non-null int64
23 CLAIM_AVG_ACC_ST_PRD     77407 non-null float64
24 POLICY_HAS_COMPLAINTS   77407 non-null int64
25 POLICY_YEARS_RENEWED_N   77407 non-null object
26 POLICY_DEDUCT_VALUE      77407 non-null float64
27 CLIENT_REGISTRATION_REGION 77407 non-null object
28 POLICY_PRICE_CHANGE       77407 non-null float64
dtypes: float64(5), int64(13), object(11)
memory usage: 17.7+ MB

```

#### ОПИСАНИЕ:

Nº; Наименование поля; Описание поля; Описание значений

0; **POLICY\_ID**; ID полиса;

1; **POLICY\_BEGIN\_MONTH**; Месяц начала действия полиса;

2; **POLICY\_END\_MONTH**; Месяц окончания действия полиса;

3; **POLICY\_IS\_RENEWED**; Факт пролонгации полиса. Прогнозируемый параметр; Для тестовой выборки параметр обнулён

4; **POLICY\_SALES\_CHANNEL**; Канал продаж полиса;

5; **POLICY\_SALES\_CHANNEL\_GROUP**; Группа каналов продаж (группировка для колонки - канал продаж полиса) ;

6; **POLICY\_BRANCH**; Филиал продажи полиса;

7; **POLICY\_MIN\_AGE**; Минимальный возраст лиц допущенных к управлению по полису;

8; **POLICY\_MIN\_DRIVING\_EXPERIENCE**; Минимальный стаж вождения лиц допущенных к управлению по полису;

9; **VEHICLE\_MAKE**; Марка ТС;

10; **VEHICLE\_MODEL**; Модель ТС;

11; **VEHICLE\_ENGINE\_POWER**; Мощность двигателя ТС;Мощность в лошадиных силах

12; **VEHICLE\_IN\_CREDIT**; ТС куплено в кредит;

13; **VEHICLE\_SUM\_INSURED**; Страховая сумма по полису (оценочная стоимость ТС - лимит возмещения);

14; **POLICY\_INTERMEDIARY**; Посредник по полису;Идентификатор посредника

15; **INSURER\_GENDER**; Пол страхователя; "M - мужчина F - женщина"

16; **POLICY\_CLM\_N**; Кол-во убытков, всего по данному полису; "Цифра - количество убытков. Для клиентов с одним убытком:

Окончание S - small, сумма убытка меньше 60% премии по полису Окончение L - large, сумма убытка больше 60% премии по полису"

17; **POLICY\_CLM\_GLT\_N**; Кол-во убытков, где клиент виновен в ДТП по данному полису;

18; **POLICY\_PRV\_CLM\_N**; Кол-во убытков, всего по предыдущему (если есть) полису (клиент пролонгируется второй и более раз);

19; **POLICY\_PRV\_CLM\_GLT\_N**; Кол-во убытков, где клиент виновен в ДТП по предыдущему полису (клиент пролонгируется второй и более раз);

20; **CLIENT\_HAS\_DAGO**; У клиента также есть полис ДАГО (ДАГО - расширение для страхования ответственности);

21; **CLIENT\_HAS\_OSAGO**; У клиента также есть полис ОСАГО;

22; **POLICY\_COURT\_SIGN**; По полису был суд;

23; **CLAIM\_AVG\_ACC\_ST\_PRD**; Среднее время от страхового случая до заявления убытков по данному полису;

24; **POLICY\_HAS\_COMPLAINTS**; По полису были жалобы (в том числе при урегулировании убытков);

25; **POLICY\_YEARS\_RENEWED\_N**; Количество лет пролонгации полиса;

26; **POLICY\_DEDUCT\_VALUE**; Сумма франшизы по полису;

27; **CLIENT\_REGISTRATION\_REGION**; Регион регистрации страхователя; Регион, где клиент зарегистрирован по паспорту

28; **POLICY\_PRICE\_CHANGE**; "Как поменялась премия при пролонгации

(""цена предложенная при пролонгации"" "-" "цена пролонгируемого полиса""") / ""цена пролонгируемого полиса""", Изменения в %, если > 0, то значит, что цена выросла

#### **Некоторые признаки являются категориальными. Исследуем их.**

Для исследования категориальных признаков разобьем исходный датасет на два - с объектами, по которым наблюдалось продление полиса (**POLICY\_IS\_RENEWED = 1**), и с объектами, у которых полис не был продлен (**POLICY\_IS\_RENEWED = 0**)

```

df0 = df.loc[df['POLICY_IS_RENEWED'] == 0]
df1 = df.loc[df['POLICY_IS_RENEWED'] == 1]

```

```
df0.info()
```

```

→ <class 'pandas.core.frame.DataFrame'>
Index: 29031 entries, 4 to 96598
Data columns (total 29 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   POLICY_ID        29031 non-null  int64  
 1   POLICY_BEGIN_MONTH 29031 non-null  int64  
 2   POLICY_END_MONTH   29031 non-null  int64  
 3   POLICY_IS_RENEWED  29031 non-null  int64  
 4   POLICY_SALES_CHANNEL 29031 non-null  int64  
 5   POLICY_SALES_CHANNEL_GROUP 29031 non-null  int64  
 6   POLICY_BRANCH       29031 non-null  object  
 7   POLICY_MIN_AGE     29031 non-null  int64  
 8   POLICY_MIN_DRIVING_EXPERIENCE 29031 non-null  int64  
 9   VEHICLE_MAKE        29031 non-null  object  
 10  VEHICLE_MODEL       29031 non-null  object  
 11  VEHICLE_ENGINE_POWER 29031 non-null  float64 
 12  VEHICLE_IN_CREDIT    29031 non-null  int64  
 13  VEHICLE_SUM_INSURED 29031 non-null  float64 
 14  POLICY_INTERMEDIARY 29031 non-null  object  
 15  INSURER_GENDER      29031 non-null  object  
 16  POLICY_CLM_N        29031 non-null  object  
 17  POLICY_CLM_GLT_N    29031 non-null  object  
 18  POLICY_PRV_CLM_N    29031 non-null  object  
 19  POLICY_PRV_CLM_GLT_N 29031 non-null  object  
 20  CLIENT_HAS_DAGO     29031 non-null  int64  
 21  CLIENT_HAS_OSAGO    29031 non-null  int64  
 22  POLICY_COURT_SIGN   29031 non-null  int64  
 23  CLAIM_AVG_ACC_ST_PRD 29031 non-null  float64 
 24  POLICY_HAS_COMPLAINTS 29031 non-null  int64  
 25  POLICY_YEARS_RENEWED_N 29031 non-null  object  
 26  POLICY_DEDUCT_VALUE  29031 non-null  float64 
 27  CLIENT_REGISTRATION_REGION 29031 non-null  object  
 28  POLICY_PRICE_CHANGE  29031 non-null  float64 
dtypes: float64(5), int64(13), object(11)
memory usage: 6.6+ MB

```

df1.info()

```

→ <class 'pandas.core.frame.DataFrame'>
Index: 48376 entries, 0 to 96604
Data columns (total 29 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   POLICY_ID        48376 non-null  int64  
 1   POLICY_BEGIN_MONTH 48376 non-null  int64  
 2   POLICY_END_MONTH   48376 non-null  int64  
 3   POLICY_IS_RENEWED  48376 non-null  int64  
 4   POLICY_SALES_CHANNEL 48376 non-null  int64  
 5   POLICY_SALES_CHANNEL_GROUP 48376 non-null  int64  
 6   POLICY_BRANCH       48376 non-null  object  
 7   POLICY_MIN_AGE     48376 non-null  int64  
 8   POLICY_MIN_DRIVING_EXPERIENCE 48376 non-null  int64  
 9   VEHICLE_MAKE        48376 non-null  object  
 10  VEHICLE_MODEL       48376 non-null  object  
 11  VEHICLE_ENGINE_POWER 48376 non-null  float64 
 12  VEHICLE_IN_CREDIT    48376 non-null  int64  
 13  VEHICLE_SUM_INSURED 48376 non-null  float64 
 14  POLICY_INTERMEDIARY 48376 non-null  object  
 15  INSURER_GENDER      48376 non-null  object  
 16  POLICY_CLM_N        48376 non-null  object  
 17  POLICY_CLM_GLT_N    48376 non-null  object  
 18  POLICY_PRV_CLM_N    48376 non-null  object  
 19  POLICY_PRV_CLM_GLT_N 48376 non-null  object  
 20  CLIENT_HAS_DAGO     48376 non-null  int64  
 21  CLIENT_HAS_OSAGO    48376 non-null  int64  
 22  POLICY_COURT_SIGN   48376 non-null  int64  
 23  CLAIM_AVG_ACC_ST_PRD 48376 non-null  float64 
 24  POLICY_HAS_COMPLAINTS 48376 non-null  int64  
 25  POLICY_YEARS_RENEWED_N 48376 non-null  object  
 26  POLICY_DEDUCT_VALUE  48376 non-null  float64 
 27  CLIENT_REGISTRATION_REGION 48376 non-null  object  
 28  POLICY_PRICE_CHANGE  48376 non-null  float64 
dtypes: float64(5), int64(13), object(11)
memory usage: 11.1+ MB

```

Количество объектов в каждом датасете отличается - датасет с объектами, продлившими страховку, в 1,6 раза больше, чем датасет с отказавшимися продлевать

POLICY\_BRANCH

```

count_values = df['POLICY_BRANCH'].value_counts()
print(count_values)

```

```

→ POLICY_BRANCH
Москва          40675
Санкт-Петербург 36732
Name: count, dtype: int64

```

POLICY\_BRANCH принимает 2 возможных значения. Кодируем с помощью Label Encoder

```
from sklearn.preprocessing import LabelEncoder
l = LabelEncoder()

df['POLICY_BRANCH'] = l.fit_transform(df['POLICY_BRANCH'])

count_values = df['POLICY_BRANCH'].value_counts()
print(count_values)
```

```
→ POLICY_BRANCH
0    40675
1    36732
Name: count, dtype: int64
```

## VEHICLE\_MAKE

```
count_values = df['VEHICLE_MAKE'].value_counts()
print(count_values)
```

```
→ VEHICLE_MAKE
Kia          8803
Hyundai      7347
Toyota        7102
Renault       7029
Ford          6496
...
Rolls-Royce   1
Krone         1
Hafei          1
<Пусто>       1
Jcb            1
Name: count, Length: 80, dtype: int64
```

VEHICLE\_MAKE имеет 80 возможных значений

```
feature='VEHICLE_MAKE'
```

Меняем значения исследуемого признака с текущих на Different по всему датасету в случае, когда такое значение встречается реже 0,01% случаев (от всего датасета)

```
total_count = len(df)

# Находим количество уникальных значений и их частоту
name_counts = df[feature].value_counts()

# Определяем значения, которые составляют менее 0,01% от общего объема датасета
threshold = total_count * 0.0001
names_to_replace = name_counts[name_counts < threshold].index

# Заменяем значения в столбце name на 'Different' для тех, которые попадают под условие
df[feature] = df[feature].replace(names_to_replace, 'Different')
```

Заменим также неинформативные значения

```
df[feature] = df[feature].replace(['другая марка (Иностранного производства)', 'другая марка (Отечественного производства)'], 'Different')
```

```
count_values = df[feature].value_counts()
print(count_values)
```

```
→ VEHICLE_MAKE
Kia          8803
Hyundai      7347
Toyota        7102
Renault       7029
Ford          6496
Mitsubishi    5752
Nissan         5227
Volkswagen    4185
BMW           3785
Skoda          3258
Audi           1745
Honda          1440
Opel           1435
Volvo          1397
Land Rover     1343
Suzuki          1294
Chevrolet       1066
Mercedes-Benz  1007
Mazda           986
Subaru          899
Ssang Yong     742
Lada            643
ВАЗ             491
УАЗ             397
Peugeot         383
Jeep            376
```

```
Infiniti      303
Citroen       269
Great Wall    267
Porsche        256
Lifan          246
Lexus           222
Fiat            195
Seat             189
Chery           141
Different       111
Geely            109
Daewoo           99
Cadillac         66
Jaguar            57
Mini              51
Dodge              41
Vortex             33
ГАЗ               29
Chrysler          28
Acura              18
Tagaz              13
Brilliance        10
Datsun              9
Hummer              9
Alfa Romeo         8
Name: count, dtype: int64
```

Повторяем разбиение на субдатасеты по целевому признаку

```
df0 = df.loc[df['POLICY_IS_RENEWED'] == 0]
df1 = df.loc[df['POLICY_IS_RENEWED'] == 1]
```

Для того, чтобы сравнивать объекты в обоих датасетах, необходимо унифицировать их содержание. Оставим в датасетах только строки (наименования признака), которые присутствуют в обоих датасетах одновременно. Уникальные наименования (присутствуют только в одном датасете) - объединим и отнесем в категорию other. После этого количество наименований в обоих датасетах должно сравняться.

```
# Находим общие наименования признаков по двум датасетам
common_features = set(df1[feature]).intersection(set(df0[feature]))

# Фильтруем df1 и df0 по общим наименованиям признаков
df1_common = df1[df1[feature].isin(common_features)]
df0_common = df0[df0[feature].isin(common_features)]

# Убираем строки с уникальными наименованиями признаков и суммируем их
other_df1 = df1[~df1[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df0 = df0[~df0[feature].isin(common_features)].groupby(feature, as_index=False).sum()

# Объединяем уникальные строки в 'other'
other_df1[feature] = 'other'
other_df0[feature] = 'other'

# Объединяем df1 и df2 с общими признаками
result1 = pd.concat([df1_common, other_df1]).reset_index(drop=True)
result0 = pd.concat([df0_common, other_df0]).reset_index(drop=True)

result1[feature].value_counts()
```



count

## VEHICLE\_MAKE

Kia	5535
Hyundai	4588
Renault	4513
Toyota	4415
Ford	4293
Nissan	3427
Mitsubishi	3119
Volkswagen	2726
BMW	2156
Skoda	2109
Audi	1129
Volvo	948
Honda	919
Opel	912
Land Rover	866
Suzuki	828
Mazda	687
Chevrolet	662
Mercedes-Benz	651
Subaru	552
Ssang Yong	441
Lada	352
BA3	271
Jeep	243
Peugeot	236
YA3	224
Citroen	171
Great Wall	154
Porsche	150
Lexus	146
Infiniti	143
Seat	129
Lifan	127
Fiat	117
Chery	76
Geely	51
Daewoo	48
Cadillac	39
Jaguar	38
Different	36
Mini	33
Dodge	26
Chrysler	17
Vortex	17
Acura	16
TA3	12
Tagaz	9
Hummer	6
Brilliance	6
Alfa Romeo	5
Datsun	2

**dtype:** int64

```
result0[feature].value_counts()
```



count

## VEHICLE\_MAKE

Kia	3268
Hyundai	2759
Toyota	2687
Mitsubishi	2633
Renault	2516
Ford	2203
Nissan	1800
BMW	1629
Volkswagen	1459
Skoda	1149
Audi	616
Opel	523
Honda	521
Land Rover	477
Suzuki	466
Volvo	449
Chevrolet	404
Mercedes-Benz	356
Subaru	347
Ssang Yong	301
Mazda	299
Lada	291
BA3	220
YA3	173
Infiniti	160
Peugeot	147
Jeep	133
Lifan	119
Great Wall	113
Porsche	106
Citroen	98
Fiat	78
Lexus	76
Different	75
Chery	65
Seat	60
Geely	58
Daewoo	51
Cadillac	27
Jaguar	19
Mini	18
FA3	17
Vortex	16
Dodge	15
Chrysler	11
Datsun	7
Brilliance	4
Tagaz	4
Hummer	3
Alfa Romeo	3
Acura	2

**dtype:** int64

Видим, что в обоих датасетах количество уникальных наименований по исседуемому признаку сравнялось

Сортируем по именам в категории, в алфавитном порядке

```
df_serv0 = result0[feature].value_counts().reset_index(name='count').sort_values(feature)
```

```
df_serv0
```



VEHICLE\_MAKE count



50	Acura	2
49	Alfa Romeo	3
10	Audi	616
7	BMW	1629
46	Brilliance	4
38	Cadillac	27
34	Chery	65
16	Chevrolet	404
44	Chrysler	11
30	Citroen	98
37	Daewoo	51
45	Datsun	7
33	Different	75
43	Dodge	15
31	Fiat	78
5	Ford	2203
36	Geely	58
28	Great Wall	113
12	Honda	521
48	Hummer	3
1	Hyundai	2759
24	Infiniti	160
39	Jaguar	19
26	Jeep	133
0	Kia	3268
21	Lada	291
13	Land Rover	477
32	Lexus	76
27	Lifan	119
20	Mazda	299
17	Mercedes-Benz	356
40	Mini	18
3	Mitsubishi	2633
6	Nissan	1800
11	Opel	523
25	Peugeot	147
29	Porsche	106
4	Renault	2516
35	Seat	60
9	Skoda	1149
19	Ssang Yong	301
18	Subaru	347
14	Suzuki	466
47	Tagaz	4
2	Toyota	2687
8	Volkswagen	1459
15	Volvo	449
42	Vortex	16
22	BA3	220
41	ГАЗ	17
23	УАЗ	173

Далее:

 Посмотреть рекомендованные графики

 New interactive sheet

```
df_serv1 = result1[feature].value_counts().reset_index(name='count').sort_values(feature)
```

```
df_serv1
```



VEHICLE\_MAKE count



44	Acura	16
49	Alfa Romeo	5
10	Audi	1129
8	BMW	2156
48	Brilliance	6
37	Cadillac	39
34	Chery	76
17	Chevrolet	662
42	Chrysler	17
26	Citroen	171
36	Daewoo	48
50	Datsun	2
39	Different	36
41	Dodge	26
33	Fiat	117
4	Ford	4293
35	Geely	51
27	Great Wall	154
12	Honda	919
47	Hummer	6
1	Hyundai	4588
30	Infiniti	143
38	Jaguar	38
23	Jeep	243
0	Kia	5535
21	Lada	352
14	Land Rover	866
29	Lexus	146
32	Lifan	127
16	Mazda	687
18	Mercedes-Benz	651
40	Mini	33
6	Mitsubishi	3119
5	Nissan	3427
13	Opel	912
24	Peugeot	236
28	Porsche	150
2	Renault	4513
31	Seat	129
9	Skoda	2109
20	Ssang Yong	441
19	Subaru	552
15	Suzuki	828
46	Tagaz	9
3	Toyota	4415
7	Volkswagen	2726
11	Volvo	948
43	Vortex	17
22	BA3	271
45	ГАЗ	12
25	УАЗ	224



Далее: [Посмотреть рекомендованные графики](#)

New interactive sheet

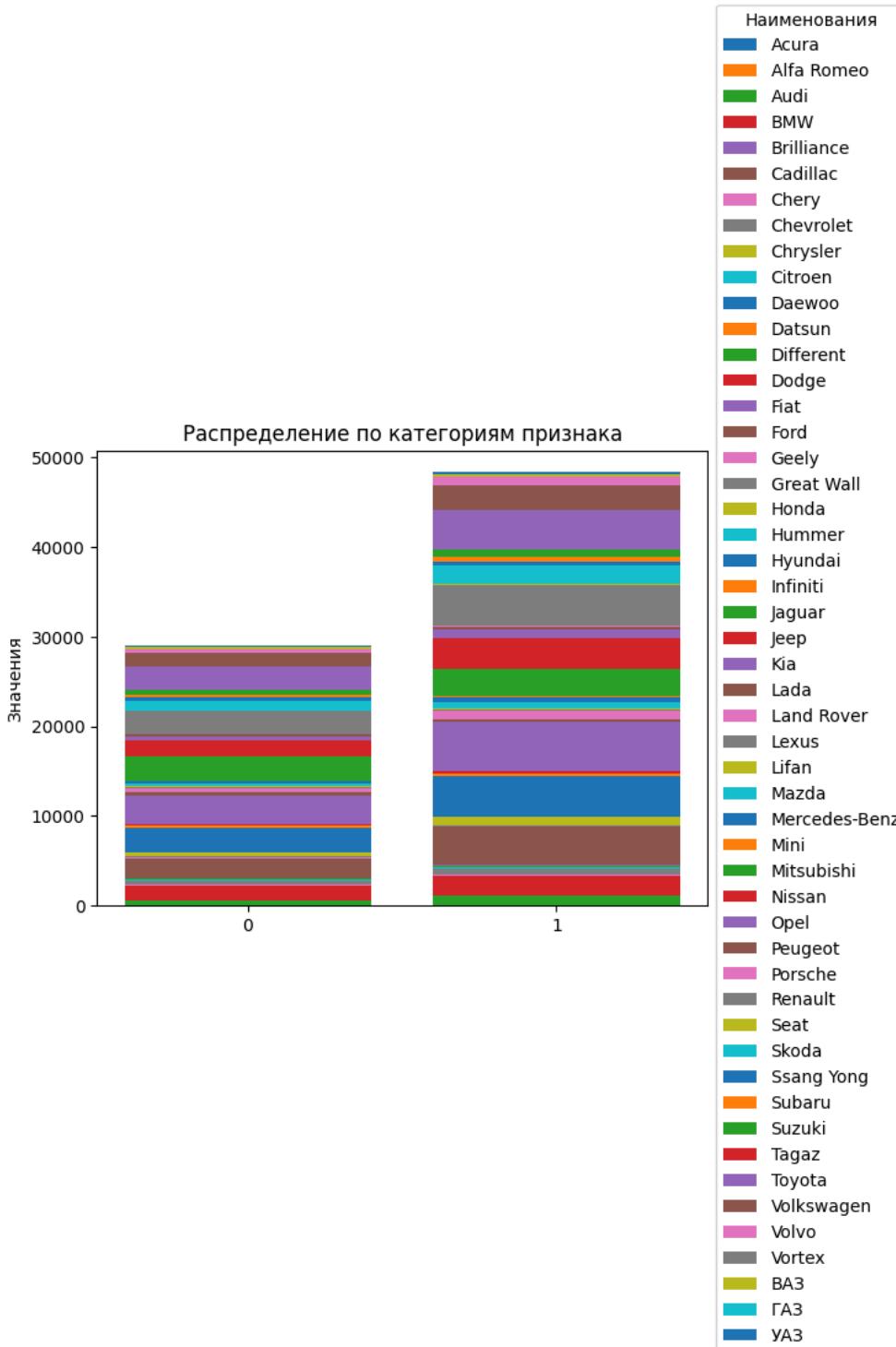
```
categories = np.array(['0', '1'])
subcategories = df_serv0[feature].tolist()
values = np.array((df_serv0['count'].tolist(), df_serv1['count'].tolist()))
fig, ax = plt.subplots()

bottom_values = np.zeros(len(categories))

for i, subcategory in enumerate(subcategories):
    ax.bar(categories, values[:, i], bottom=bottom_values, label=subcategory)
    bottom_values += values[:, i]

ax.set_ylabel('Значения')
ax.set_title('Распределение по категориям признака')
ax.legend(title='Наименования', loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```



Визуально различий в распределении по категориям исследуемого признака в обоих датасетах не наблюдаем, объекты распределены примерно одинаково. Предполагаем, что исследуемый признак VEHICLE\_MAKE не оказывает существенного влияния на целевой признак (продление/отказ от продления полиса страхования POLICY\_IS\_RENEWED).

Однако, совсем отбрасывать признак из датасета не будем - применим кодирование категорий признака на основе корреляции с целевым признаком.

Создадим в служебном датасете с подсчетом уникальных значений по исследуемому признаку дополнительный столбец rank, который будет определяться как количество присутствий этого значения в субдатасете с целевым значением POLICY\_IS\_RENEWED = 1, деленное на количество присутствий этого значения в субдатасете POLICY\_IS\_RENEWED = 0, увеличенное в 1000 раз (чтобы не повторялись). Иными словами, мы будем подсказывать модели, насколько чаще конкретное значение категориальной переменной встречаются в субдатасете с продленным полисом КАСКО по сравнению с субдатасетом с непродленным.

```
df_serv0['rank']=(1000*df_serv1['count']/df_serv0['count']).round().astype(int)
```

```
df_serv0
```



VEHICLE\_MAKE count rank



50	Acura	2	1000
49	Alfa Romeo	3	1667
10	Audi	616	1833
7	BMW	1629	1673
46	Brilliance	4	2250
38	Cadillac	27	1407
34	Chery	65	1169
16	Chevrolet	404	1700
44	Chrysler	11	1455
30	Citroen	98	1459
37	Daewoo	51	765
45	Datsun	7	1714
33	Different	75	1560
43	Dodge	15	1133
31	Fiat	78	1654
5	Ford	2203	1556
36	Geely	58	828
28	Great Wall	113	1327
12	Honda	521	1764
48	Hummer	3	2000
1	Hyundai	2759	1663
24	Infiniti	160	1475
39	Jaguar	19	1895
26	Jeep	133	1286
0	Kia	3268	1694
21	Lada	291	1210
13	Land Rover	477	1912
32	Lexus	76	1671
27	Lifan	119	1294
20	Mazda	299	1475
17	Mercedes-Benz	356	1860
40	Mini	18	1833
3	Mitsubishi	2633	1677
6	Nissan	1800	1733
11	Opel	523	1813
25	Peugeot	147	1524
29	Porsche	106	1377
4	Renault	2516	1706
35	Seat	60	850
9	Skoda	1149	1836
19	Ssang Yong	301	1834
18	Subaru	347	1876
14	Suzuki	466	1858
47	Tagaz	4	1500
2	Toyota	2687	1680
8	Volkswagen	1459	1478
15	Volvo	449	1844
42	Vortex	16	1062
22	BA3	220	1232
41	ГАЗ	17	1529
23	УАЗ	173	1405



Перекодируем значения категориальной переменной исходя из полученных значений rank в исходном датасете

```
with pd.option_context("future.no_silent_downcasting", True):
    df[feature]=df[feature].replace(dict(zip(df_serv0[feature], df_serv0['rank'])))
df[feature]=df[feature].round().astype(int)
```

```
count_values = df[feature].value_counts().to_frame()
print(count_values.sort_values(feature).to_string(max_rows=15))
```

	count
VEHICLE_MAKE	
765	99
828	109
850	189
1000	18
1062	33
1133	41
1169	141
...	...
1858	1294
1860	1007
1876	899
1895	57
1912	1343
2000	9
2250	10

Видим, что соотношение продленных/не продленных полисов варьирует по отдельным значениям исследуемого признака от 76,5% до 225%

#### VEHICLE\_MODEL

```
count_values = df['VEHICLE_MODEL'].value_counts()
print(count_values)
```

	VEHICLE_MODEL
RAV4	3367
Sportage	2961
Duster	2908
Focus	2557
ix35	2394
...	
RCZ	1
Tager	1
2194 Kalina Cross	1
Coupe	1
Vita	1
Name: count, Length: 525, dtype: int64	

VEHICLE\_MAKE имеет 525 возможных значения

```
feature='VEHICLE_MODEL'
```

Меняем значения исследуемого признака с текущих на Different по всему датасету в случае, когда такое значение встречается реже 0,05% случаев (от всего датасета)

```
total_count = len(df)

# Находим количество уникальных значений и их частоту
name_counts = df[feature].value_counts()

# Определяем значения, которые составляют менее 0,05% от общего объема датасета
threshold = total_count * 0.0005
names_to_replace = name_counts[name_counts < threshold].index

# Заменяем значения в столбце name на 'Different' для тех, которые попадают под условие
df[feature] = df[feature].replace(names_to_replace, 'Different')
```

Заменим также неинформативные значения

```
df[feature] = df[feature].replace('другая модель (Прицепы к грузовым автомобилям)', 'Different')
```

```
count_values = df[feature].value_counts()
print(count_values)
```

	VEHICLE_MODEL
RAV4	3367
Different	3293

```
Sportage    2961
Duster      2908
Focus       2557
                ...
2114        39
Roomster    39
Rexton      39
3008        39
A8          39
Name: count, Length: 191, dtype: int64
```

Повторяем разбиение на субдатасеты по целевому признаку

```
df0 = df.loc[df['POLICY_IS_RENEWED'] == 0]
df1 = df.loc[df['POLICY_IS_RENEWED'] == 1]
```

Для того, чтобы сравнивать объекты в обоих датасетах, необходимо унифицировать их содержание. Оставим в датасетах только строки (наименования признака), которые присутствуют в обоих датасетах одновременно. Уникальные наименования (присутствуют только в одном датасете) - объединим и отнесем в категорию other. После этого количество наименований в обоих датасетах должно сравняться.

```
# Находим общие наименования признаков по двум датасетам
common_features = set(df1[feature]).intersection(set(df0[feature]))
```

```
# Фильтруем df1 и df0 по общим наименованиям признаков
df1_common = df1[df1[feature].isin(common_features)]
df0_common = df0[df0[feature].isin(common_features)]
```

```
# Убираем строки с уникальными наименованиями признаков и суммируем их
other_df1 = df1[~df1[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df0 = df0[~df0[feature].isin(common_features)].groupby(feature, as_index=False).sum()
```

```
# Объединяем уникальные строки в 'other'
other_df1[feature] = 'other'
other_df0[feature] = 'other'
```

```
# Объединяем df1 и df2 с общими признаками
result1 = pd.concat([df1_common, other_df1]).reset_index(drop=True)
result0 = pd.concat([df0_common, other_df0]).reset_index(drop=True)
```

Сортируем по именам в категории, в алфавитном порядке

```
df_serv0 = result0[feature].value_counts().reset_index(name='count').sort_values(feature)
df_serv1 = result1[feature].value_counts().reset_index(name='count').sort_values(feature)
```

```
df_serv0.shape
```

```
→ (191, 2)
```

```
df_serv1.shape
```

```
→ (191, 2)
```

```
categories = np.array(['0', '1'])
subcategories = df_serv0[feature].tolist()
values = np.array([df_serv0['count'].tolist(), df_serv1['count'].tolist()])
fig, ax = plt.subplots()

bottom_values = np.zeros(len(categories))

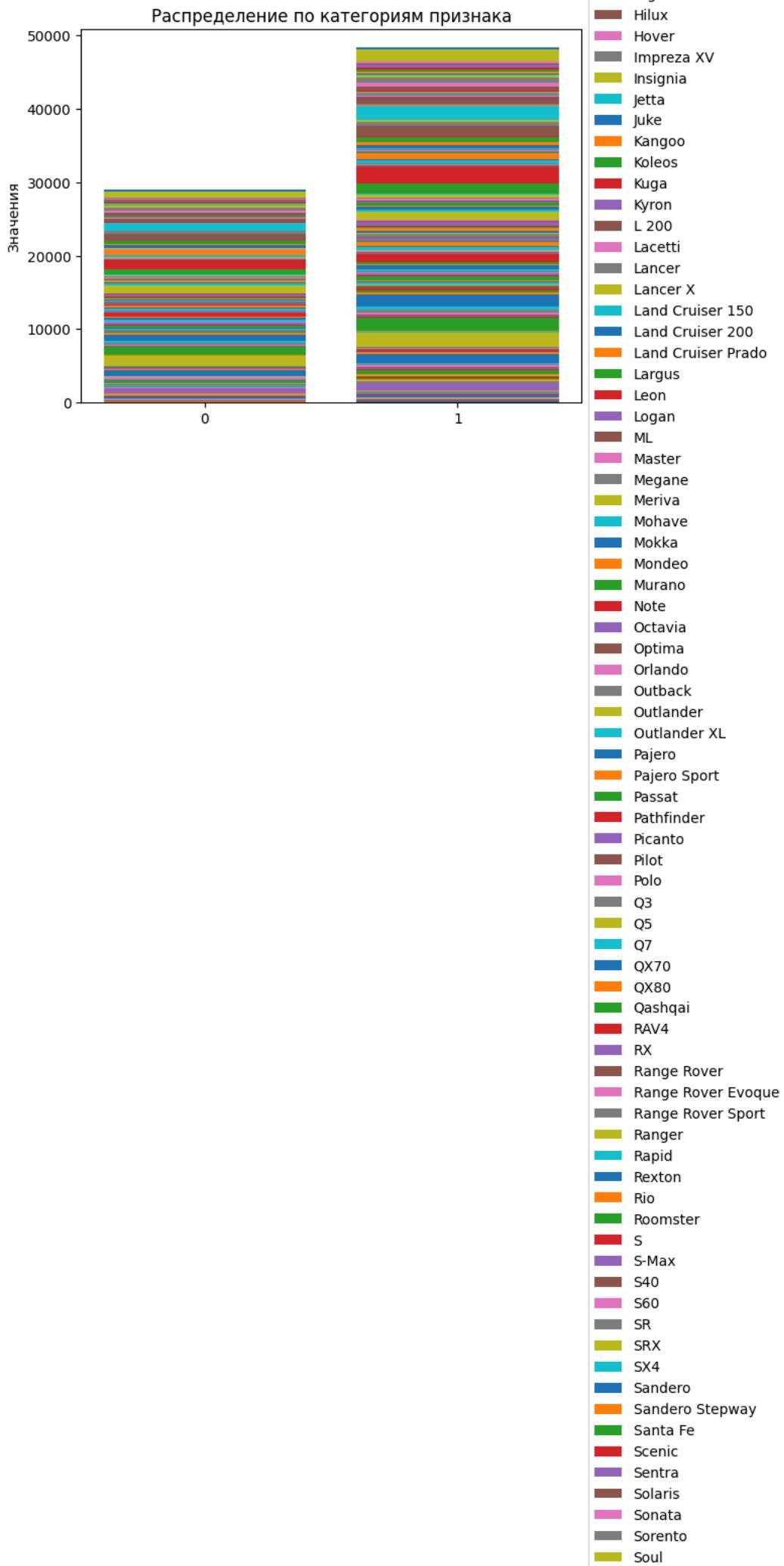
for i, subcategory in enumerate(subcategories):
    ax.bar(categories, values[:, i], bottom=bottom_values, label=subcategory)
    bottom_values += values[:, i]

ax.set_ylabel('Значения')
ax.set_title('Распределение по категориям признака')
ax.legend(title='Наименования', loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```

Наименования

- 1-series
- 1117 Kalina
- 1119 Kalina
- 2114
- 2121
- 3
- 3-series
- 3-series GT
- 3008
- 308
- 3163 Patriot
- 4-series
- 4007
- 5-series
- 6
- 7-series
- A1
- A3
- A4
- A4 Allroad
- A5
- A6
- A6 Allroad
- A8
- ASX
- Accent
- Accord
- Actyon
- Almera
- Altea
- Amarok
- Antara
- Astra
- Auris
- Avensis
- Aveo
- B
- Berlingo
- C
- C-Crosser
- C-Max
- C4
- CR-V
- CX-5
- CX-7
- Caddy
- Camry
- Captiva
- Caravelle
- Cayenne
- Ceed
- Cerato
- Cherokee
- Civic
- Compass
- Corolla
- Corsa
- Cruze
- Different
- Discovery
- Doblo
- Ducato
- Duster
- E
- EcoSport
- Elantra
- Explorer
- Fabia
- Fiesta
- Fluence
- Focus
- Forester
- Freelander
- Fusion
- GL
- GLK
- Galaxy
- Gentra
- Getz



Sportage
Superb
Swift
Tahoe
Teana
Tiggo (T11)
Tiguan
Tiida
Touareg
Touran
Trailblazer
Transit
Tucson
Venga
Venza
Verso
X-Trail
X1
X3
X5
X6
X60
XC60
XC70
XC90
Yeti
Zafira
i30
i40
ix35
ix55
Другая модель (Легковое ТС)

Визуально различий в распределении по категориям исследуемого признака в обоих датасетах не наблюдаем, объекты распределены примерно одинаково. Предполагаем, что исследуемый признак VEHICLE\_MODEL не оказывает существенного влияния на целевой признак (продление/отказ от продления полиса страхования POLICY\_IS\_RENEWED).

Однако, совсем отбрасывать признак из датасета не будем - применим кодирование категорий признака на основе корреляции с целевым признаком.

Создадим в служебном датасете с подсчетом уникальных значений по исследуемому признаку дополнительный столбец rank, который будет определяться как количество присутствий этого значения в субдатасете с целевым значением POLICY\_IS\_RENEWED = 1, деленное на количество присутствий этого значения в субдатасете POLICY\_IS\_RENEWED = 0, увеличенное в 1000 раз (чтобы не повторялись). Иными словами, мы будем подсказывать модели, насколько чаще конкретное значение категориальной переменной встречаются в субдатасете с продленным полисом КАСКО по сравнению с субдатасетом с непродленным.

```
df_serv0['rank']=(1000*df_serv1['count']/df_serv0['count']).round().astype(int)
```

Перекодируем значения категориальной переменной исходя из полученных значений rank в исходном датасете

```
with pd.option_context("future.no_silent_downcasting", True):
    df[feature]=df[feature].replace(dict(zip(df_serv0[feature], df_serv0['rank'])))
```

```
df[feature]=df[feature].round().astype(int)
```

```
count_values = df[feature].value_counts().to_frame()
print(count_values.sort_values(feature).to_string(max_rows=15))
```

```
VEHICLE_MODEL count
1464          85
1481          82
1498         959
1500         426
1507         691
1520         153
1524        6660
...
1858         283
1871         184
1899        1669
1901         546
1941         579
2000         39
2013        1630
```

Видим, что соотношение продленных/не продленных полисов варьирует по отдельным значениям исследуемого признака от 146% до 201%, однако количество объектов со значением 146% очень мало

## POLICY\_INTERMEDIARY

```
count_values = df['POLICY_INTERMEDIARY'].value_counts()
print(count_values)
```

```
POLICY_INTERMEDIARY
N          16448
1096       9026
910        2955
1252       1840
326        1084
...
123         1
1325        1
367         1
150         1
733         1
Name: count, Length: 1333, dtype: int64
```

POLICY\_INTERMEDIARY имеет 1406 возможных значений

```
feature='POLICY_INTERMEDIARY'
```

Меняем значения исследуемого признака с текущих на Different по всему датасету в случае, когда такое значение встречается реже 0,05% случаев (от всего датасета)

```
total_count = len(df)

# Находим количество уникальных значений и их частоту
name_counts = df[feature].value_counts()

# Определяем значения, которые составляют менее 0,05% от общего объема датасета
threshold = total_count * 0.0005
names_to_replace = name_counts[name_counts < threshold].index
```

```
# Заменяем значения в столбце name на 'Different' для тех, которые попадают под условие
df[feature] = df[feature].replace(names_to_replace, 'Different')
```

```
count_values = df[feature].value_counts()
print(count_values)
```

```
→ POLICY_INTERMEDIARY
N           16448
1096        9026
Different    7093
910         2955
1252        1840
...
881          39
202          39
1290         39
319          39
500          39
Name: count, Length: 209, dtype: int64
```

```
df0 = df.loc[df['POLICY_IS_RENEWED'] == 0]
df1 = df.loc[df['POLICY_IS_RENEWED'] == 1]
common_features = set(df1[feature]).intersection(set(df0[feature]))
df1_common = df1[df1[feature].isin(common_features)]
df0_common = df0[df0[feature].isin(common_features)]
other_df1 = df1[~df1[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df0 = df0[~df0[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df1[feature] = 'other'
other_df0[feature] = 'other'
result1 = pd.concat([df1_common, other_df1]).reset_index(drop=True)
result0 = pd.concat([df0_common, other_df0]).reset_index(drop=True)
df_serv0 = result0[feature].value_counts().reset_index(name='count').sort_values(feature)
df_serv1 = result1[feature].value_counts().reset_index(name='count').sort_values(feature)
```

```
categories = np.array(['0', '1'])
subcategories = df_serv0[feature].tolist()
values = np.array([df_serv0['count'].tolist(), df_serv1['count'].tolist()])
fig, ax = plt.subplots()

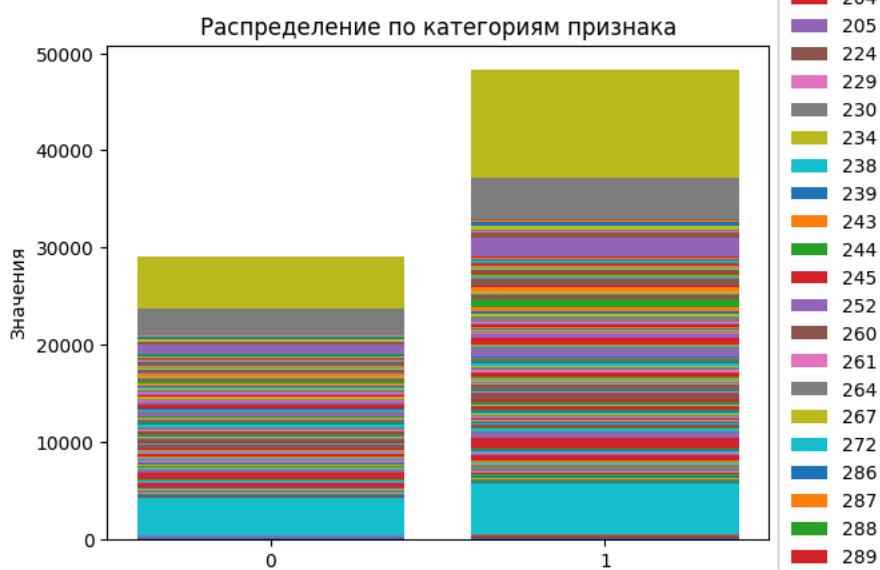
bottom_values = np.zeros(len(categories))

for i, subcategory in enumerate(subcategories):
    ax.bar(categories, values[:, i], bottom=bottom_values, label=subcategory)
    bottom_values += values[:, i]

ax.set_ylabel('Значения')
ax.set_title('Распределение по категориям признака')
ax.legend(title='Наименования', loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```

Наименования	
1020	
1022	
1034	
1037	
1051	
1058	
1063	
1070	
1092	
1096	
110	
1131	
1142	
1144	
1151	
1156	
1157	
1158	
1167	
117	
1173	
1186	
1188	
1190	
1191	
1193	
1199	
1200	
1209	
1211	
1214	
1219	
1220	
1221	
1223	
1224	
1227	
1234	
1238	
1241	
1244	
1245	
1250	
1252	
1253	
1257	
1258	
126	
1274	
1276	
1278	
1286	
1287	
129	
1290	
1292	
1299	
1308	
1315	
1316	
1319	
1323	
1326	
1330	
1333	
1338	
1346	
1355	
1357	
1362	
1366	
1369	
1370	
1372	
139	
1397	
1401	
1406	
143	



143  
149  
153  
157  
160  
164  
167  
171  
174  
181  
186  
187  
190  
2  
202  
204  
205  
224  
229  
230  
234  
238  
239  
243  
244  
245  
252  
260  
261  
264  
267  
272  
286  
287  
288  
289  
292  
293  
294  
296  
297  
3  
319  
320  
321  
322  
326  
328  
33  
330  
335  
338  
339  
34  
340  
343  
344  
348  
352  
360  
363  
376  
381  
396  
399  
400  
406  
412  
437  
454  
461  
469  
481  
485  
493  
494  
500  
509  
511  
515  
516

517
519
528
533
55
56
565
57
580
583
585
586
6
60
616
622
641
645
648
66
67
674
682
686
7
702
703
704
709
710
782
826
849
881
89
910
917
924
94
946
947
948
951
952
96
965
967
998
Different
N

Визуально различий в распределении по категориям исследуемого признака в обоих датасетах не наблюдаем, объекты распределены примерно одинаково. Предполагаем, что исследуемый признак POLICY\_INTERMEDIARY не оказывает существенного влияния на целевой признак (продление/отказ от продления полиса страхования POLICY\_IS\_RENEWED).

Однако, совсем отбрасывать признак из датасета не будем - применим кодирование категорий признака на основе корреляции с целевым признаком.

Создадим в служебном датасете с подсчетом уникальных значений по исследуемому признаку дополнительный столбец rank, который будет определяться как количество присутствий этого значения в субдатасете с целевым значением POLICY\_IS\_RENEWED = 1, деленное на количество присутствий этого значения в субдатасете POLICY\_IS\_RENEWED = 0, увеличенное в 1000 раз (чтобы не повторялись). Иными словами, мы будем подсказывать модели, насколько чаще конкретное значение категориальной переменной встречаются в субдатасете с продленным полисом КАСКО по сравнению с субдатасетом с непродленным.

```
df_serv0['rank']=(1000*df_serv1['count']/df_serv0['count']).round().astype(int)
```

```
with pd.option_context("future.no_silent_downcasting", True):
    df[feature]=df[feature].replace(dict(zip(df_serv0[feature], df_serv0['rank'])))
```

```
df[feature]=df[feature].round().astype(int)
count_values = df[feature].value_counts().to_frame()
print(count_values.sort_values(feature).to_string(max_rows=15))
```

```
→ count
POLICY_INTERMEDIARY
1329      443
1336      264
1343      160
1347      302
1354      205
1357      127
1361      274
...
1955     1054
1968     1032
1982     1002
2000      189
2125      89
2133    16448
2400      39
```

Видим, что соотношение продленных/не продленных полисов варьирует по отдельным значениям исследуемого признака от 133% до 240%, однако количество объектов со значением 240% очень мало. По данным также можно выяснить, какой агент работает лучше остальных (наивысший коэффициент продления полисов)

## INSURER\_GENDER

```
count_values = df['INSURER_GENDER'].value_counts()
print(count_values)
```

```
→ INSURER_GENDER
M      48923
F      28484
Name: count, dtype: int64
```

Всего 2 значения, можно кодировать по шкале, применяем Label Encoder

```
df['INSURER_GENDER'] = le.fit_transform(df['INSURER_GENDER'])
```

```
count_values = df['INSURER_GENDER'].value_counts()
print(count_values)
```

```
→ INSURER_GENDER
1      48923
0      28484
Name: count, dtype: int64
```

## POLICY\_CLM\_N

```
count_values = df['POLICY_CLM_N'].value_counts()
print(count_values)
```

```
→ POLICY_CLM_N
0      54481
1S     8073
1L     8000
2      4972
3      1377
4+     458
n/d     46
Name: count, dtype: int64
```

Видим, что мода признака = 0. Заменяем n/d на моду

```
df['POLICY_CLM_N'] = df['POLICY_CLM_N'].replace('n/d', '0')
```

```
count_values = df['POLICY_CLM_N'].value_counts()  
print(count_values)
```

```
→ POLICY_CLM_N  
0      54527  
1S     8073  
1L     8000  
2      4972  
3      1377  
4+     458  
Name: count, dtype: int64
```

```
df['POLICY_CLM_N'] = df['POLICY_CLM_N'].replace('1S', '1a')  
df['POLICY_CLM_N'] = df['POLICY_CLM_N'].replace('1L', '1b')
```

Всего 6 значений, можно кодировать по шкале, применяем Label Encoder

```
df['POLICY_CLM_N'] = l.fit_transform(df['POLICY_CLM_N'])
```

```
count_values = df['POLICY_CLM_N'].value_counts()  
print(count_values)
```

```
→ POLICY_CLM_N  
0      54527  
1      8073  
2      8000  
3      4972  
4      1377  
5      458  
Name: count, dtype: int64
```

## POLICY\_CLM\_GLT\_N

```
count_values = df['POLICY_CLM_GLT_N'].value_counts()  
print(count_values)
```

```
→ POLICY_CLM_GLT_N  
0      57885  
1S     8016  
1L     6223  
2      3941  
3      961  
4+     335  
n/d     46  
Name: count, dtype: int64
```

```
df['POLICY_CLM_GLT_N'] = df['POLICY_CLM_GLT_N'].replace('1S', '1a')  
df['POLICY_CLM_GLT_N'] = df['POLICY_CLM_GLT_N'].replace('1L', '1b')
```

Видим, что мода признака = 0. Заменяем n/d на моду

```
df['POLICY_CLM_GLT_N'] = df['POLICY_CLM_GLT_N'].replace('n/d', '0')
```

Всего 6 значений, можно кодировать по шкале, применяем Label Encoder

```
df['POLICY_CLM_GLT_N'] = l.fit_transform(df['POLICY_CLM_GLT_N'])
```

```
count_values = df['POLICY_CLM_GLT_N'].value_counts()  
print(count_values)
```

```
→ POLICY_CLM_GLT_N  
0      57931  
1      8016  
2      6223  
3      3941  
4      961  
5      335  
Name: count, dtype: int64
```

## POLICY\_PRV\_CLM\_N

```
count_values = df['POLICY_PRV_CLM_N'].value_counts()  
print(count_values)
```

```
→ POLICY_PRV_CLM_N  
0      35747  
N      26680  
1S     5817  
1L     5403
```

```
2      3012
3      641
4+     107
Name: count, dtype: int64
```

N - отсутствие значения (нет убытков по предыдущему полису) - меняем на моду 0

```
df['POLICY_PRV_CLM_N'] = df['POLICY_PRV_CLM_N'].replace('1S', '1a')
df['POLICY_PRV_CLM_N'] = df['POLICY_PRV_CLM_N'].replace('1L', '1b')
```

```
df['POLICY_PRV_CLM_N'] = df['POLICY_PRV_CLM_N'].replace('N', '0')
```

Всего 6 значений, можно кодировать по шкале, применяем Label Encoder\*

```
df['POLICY_PRV_CLM_N'] = l.fit_transform(df['POLICY_PRV_CLM_N'])
```

```
count_values = df['POLICY_PRV_CLM_N'].value_counts()
print(count_values)
```

```
→ POLICY_PRV_CLM_N
0      62427
1      5817
2      5403
3      3012
4      641
5      107
Name: count, dtype: int64
```

POLICY\_PRV\_CLM\_GLT\_N

```
count_values = df['POLICY_PRV_CLM_GLT_N'].value_counts()
print(count_values)
```

```
→ POLICY_PRV_CLM_GLT_N
0      38588
N      26680
1S     5879
1L     3850
2      2073
3      285
4+     52
Name: count, dtype: int64
```

```
df['POLICY_PRV_CLM_GLT_N'] = df['POLICY_PRV_CLM_GLT_N'].replace('1S', '1a')
df['POLICY_PRV_CLM_GLT_N'] = df['POLICY_PRV_CLM_GLT_N'].replace('1L', '1b')
```

N - отсутствие значения (нет убытков по предыдущему полису) - меняем на моду 0

```
df['POLICY_PRV_CLM_GLT_N'] = df['POLICY_PRV_CLM_GLT_N'].replace('N', '0')
```

```
df['POLICY_PRV_CLM_GLT_N'] = l.fit_transform(df['POLICY_PRV_CLM_GLT_N'])
```

```
count_values = df['POLICY_PRV_CLM_GLT_N'].value_counts()
print(count_values)
```

```
→ POLICY_PRV_CLM_GLT_N
0      65268
1      5879
2      3850
3      2073
4      285
5      52
Name: count, dtype: int64
```

Всего 7 значений, можно кодировать по шкале, применяем Label Encoder

POLICY\_YEARS\_RENEWED\_N

```
count_values = df['POLICY_YEARS_RENEWED_N'].value_counts()
print(count_values)
```

```
→ POLICY_YEARS_RENEWED_N
0      26634
1      20261
2      12230
3      9261
4      4943
5      1806
6      1393
7      667
8       90
```

```
9      62
N      46
10     14
Name: count, dtype: int64
```

Видим, что мода признака = 0. Заменяем N на моду

```
df['POLICY_YEARS_RENEWED_N'] = df['POLICY_YEARS_RENEWED_N'].replace('N', '0')
```

```
count_values = df['POLICY_YEARS_RENEWED_N'].value_counts()
print(count_values)
```

```
→ POLICY_YEARS_RENEWED_N
0      26680
1      20261
2      12230
3      9261
4      4943
5      1806
6      1393
7      667
8      90
9      62
10     14
Name: count, dtype: int64
```

```
df['POLICY_YEARS_RENEWED_N'] = df['POLICY_YEARS_RENEWED_N'].replace('10', '99')
```

Всего 11 значений, можно кодировать по шкале, применяем Label Encoder

```
df['POLICY_YEARS_RENEWED_N'] = l.fit_transform(df['POLICY_YEARS_RENEWED_N'])
```

```
count_values = df['POLICY_YEARS_RENEWED_N'].value_counts()
print(count_values)
```

```
→ POLICY_YEARS_RENEWED_N
0      26680
1      20261
2      12230
3      9261
4      4943
5      1806
6      1393
7      667
8      90
9      62
10     14
Name: count, dtype: int64
```

## CLIENT\_REGISTRATION\_REGION

```
count_values = df['CLIENT_REGISTRATION_REGION'].value_counts()
print(count_values)
```

```
→ CLIENT_REGISTRATION_REGION
Санкт-Петербург          31124
Москва                   27621
Московская                10306
Ленинградская              4660
N                         898
...
Северная Осетия - Алания    2
Магаданская                  2
Ненецкий                     2
Чукотский                     1
Камчатский                     1
Name: count, Length: 83, dtype: int64
```

```
feature='CLIENT_REGISTRATION_REGION'
```

Меняем значения исследуемого признака с текущих на Different по всему датасету в случае, когда такое значение встречается реже 0,05% случаев (от всего датасета)

```
total_count = len(df)

# Находим количество уникальных значений и их частоту
name_counts = df[feature].value_counts()

# Определяем значения, которые составляют менее 0,05% от общего объема датасета
threshold = total_count * 0.0005
names_to_replace = name_counts[name_counts < threshold].index

# Заменяем значения в столбце name на 'Different' для тех, которые попадают под условие
df[feature] = df[feature].replace(names_to_replace, 'Different')
```

```

df[feature] = df[feature].replace('N', 'Different')

count_values = df[feature].value_counts()
print(count_values)

→ CLIENT_REGISTRATION_REGION
Санкт-Петербург      31124
Москва              27621
Московская          10306
Ленинградская       4660
Different            1821
Калужская           212
Тульская             179
Тверская             158
Владимирская        134
Смоленская          110
Новгородская         105
Псковская            95
Нижегородская        83
Саратовская          81
Самарская            75
Рязанская            74
Мурманская           72
Ивановская           70
Волгоградская        64
Брянская             63
Карелия              62
Тамбовская           60
Ростовская            47
Мордовия              46
Белгородская          45
Свердловская          40
Name: count, dtype: int64

df0 = df.loc[df['POLICY_IS_RENEWED'] == 0]
df1 = df.loc[df['POLICY_IS_RENEWED'] == 1]
common_features = set(df1[feature]).intersection(set(df0[feature]))
df1_common = df1[df1[feature].isin(common_features)]
df0_common = df0[df0[feature].isin(common_features)]
other_df1 = df1[~df1[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df0 = df0[~df0[feature].isin(common_features)].groupby(feature, as_index=False).sum()
other_df1[feature] = 'other'
other_df0[feature] = 'other'
result1 = pd.concat([df1_common, other_df1]).reset_index(drop=True)
result0 = pd.concat([df0_common, other_df0]).reset_index(drop=True)
df_serv0 = result0[feature].value_counts().reset_index(name='count').sort_values(feature)
df_serv1 = result1[feature].value_counts().reset_index(name='count').sort_values(feature)

categories = np.array(['0', '1'])
subcategories = df_serv0[feature].tolist()
values = np.array([df_serv0['count'].tolist(), df_serv1['count'].tolist()])
fig, ax = plt.subplots()

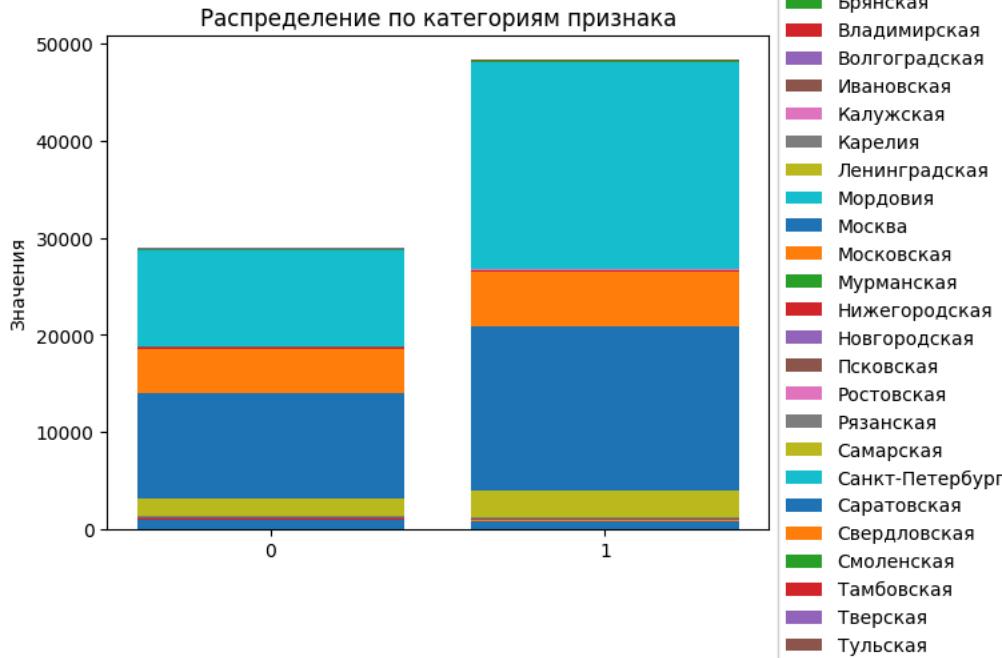
bottom_values = np.zeros(len(categories))

for i, subcategory in enumerate(subcategories):
    ax.bar(categories, values[:, i], bottom=bottom_values, label=subcategory)
    bottom_values += values[:, i]

ax.set_ylabel('Значения')
ax.set_title('Распределение по категориям признака')
ax.legend(title='Наименования', loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()

```



Визуально есть некоторые различия в распределении по категориям исследуемого признака в обоих датасетах.

Применим кодирование категорий признака на основе корреляции с целевым признаком.

Создадим в служебном датасете с подсчетом уникальных значений по исследуемому признаку дополнительный столбец rank, который будет определяться как количество присутствий этого значения в субдатасете с целевым значением POLICY\_IS\_RENEWED = 1, деленное на количество присутствий этого значения в субдатасете POLICY\_IS\_RENEWED = 0, увеличенное в 1000 раз (чтобы не повторялись). Иными словами, мы будем подсказывать модели, насколько чаще конкретное значение категориальной переменной встречаются в субдатасете с продленным полисом КАСКО по сравнению с субдатасетом с непродленным.

```
df_serv0['rank']=(1000*df_serv1['count']/df_serv0['count']).round().astype(int)

with pd.option_context("future.no_silent_downcasting", True):
    df[feature]=df[feature].replace(dict(zip(df_serv0[feature], df_serv0['rank'])))

df[feature]=df[feature].round().astype(int)
count_values = df[feature].value_counts().to_frame()
print(count_values.sort_values(feature).to_string(max_rows=15))
```

	count
CLIENT_REGISTRATION_REGION	
662	158
833	110
871	212
895	40
907	105
909	1821
914	63
...	...
1050	47
1051	81
1214	179
1233	10306
1535	4660
1705	31124
1968	27621

Видим, что соотношение продленных/не продленных полисов варьирует по отдельным значениям исследуемого признака от 66% до 197%.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 77407 entries, 0 to 96604
Data columns (total 29 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   POLICY_ID       77407 non-null   int64  
 1   POLICY_BEGIN_MONTH 77407 non-null   int64  
 2   POLICY_END_MONTH 77407 non-null   int64  
 3   POLICY_IS_RENEWED 77407 non-null   int64  
 4   POLICY_SALES_CHANNEL 77407 non-null   int64
```

```

5  POLICY_SALES_CHANNEL_GROUP    77407 non-null  int64
6  POLICY_BRANCH                 77407 non-null  int64
7  POLICY_MIN_AGE                77407 non-null  int64
8  POLICY_MIN_DRIVING_EXPERIENCE 77407 non-null  int64
9  VEHICLE_MAKE                  77407 non-null  int64
10 VEHICLE_MODEL                 77407 non-null  int64
11 VEHICLE_ENGINE_POWER          77407 non-null  float64
12 VEHICLE_IN_CREDIT             77407 non-null  int64
13 VEHICLE_SUM_INSURED           77407 non-null  float64
14 POLICY_INTERMEDIARY           77407 non-null  int64
15 INSURER_GENDER                77407 non-null  int64
16 POLICY_CLM_N                  77407 non-null  int64
17 POLICY_CLM_GLT_N              77407 non-null  int64
18 POLICY_PRV_CLM_N              77407 non-null  int64
19 POLICY_PRV_CLM_GLT_N          77407 non-null  int64
20 CLIENT_HAS_DAGO               77407 non-null  int64
21 CLIENT_HAS_OSAGO              77407 non-null  int64
22 POLICY_COURT_SIGN             77407 non-null  int64
23 CLAIM_AVG_ACC_ST_PRD          77407 non-null  float64
24 POLICY_HAS_COMPLAINTS         77407 non-null  int64
25 POLICY_YEARS_RENEWED_N         77407 non-null  int64
26 POLICY_DEDUCT_VALUE            77407 non-null  float64
27 CLIENT_REGISTRATION_REGION    77407 non-null  int64
28 POLICY_PRICE_CHANGE            77407 non-null  float64
dtypes: float64(5), int64(24)
memory usage: 17.7 MB

```

df.head(10)

	POLICY_ID	POLICY_BEGIN_MONTH	POLICY_END_MONTH	POLICY_IS_RENEWED	POLICY_SALES_CHANNEL	POLICY_SALES_CHANNEL_GROUP	POLICY_BRANCH	POLICY_MIN_AGE
0	1	1	1	1	39	1	1	0
1	2	1	1	1	50	5	5	0
2	3	1	1	1	52	6	6	0
3	4	1	1	1	50	5	5	0
4	5	1	1	0	52	6	6	1
5	6	2	1	1	2	4	4	1
6	7	1	1	1	52	6	6	0
7	8	2	2	1	10	1	1	1
8	9	1	1	0	53	6	6	0
10	11	1	1	0	50	5	5	1

10 rows × 29 columns

```
df = df.drop('POLICY_ID', axis=1) #удаляем незначащий столбец ID
```

```
df.describe().T.style.background_gradient(cmap='YlOrRd')
```

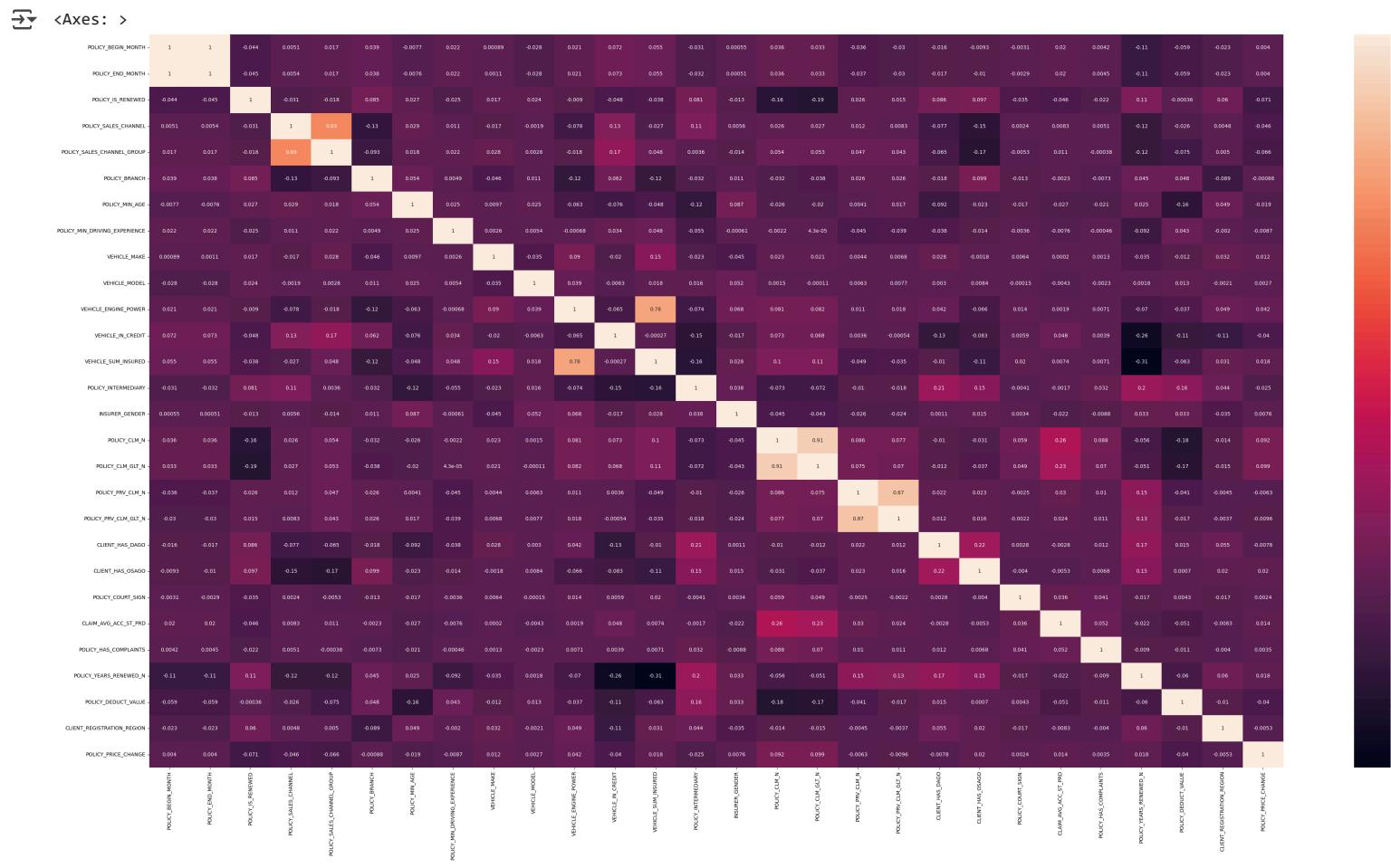
		count	mean	std	min	25%	50%	75%	m
POLICY_BEGIN_MONTH	77407.000000	6.888033	3.396889	1.000000	4.000000	7.000000	10.000000	12.0000	
POLICY_END_MONTH	77407.000000	6.866046	3.400070	1.000000	4.000000	7.000000	10.000000	12.0000	
POLICY_IS_RENEWED	77407.000000	0.624956	0.484137	0.000000	0.000000	1.000000	1.000000	1.0000	
POLICY_SALES_CHANNEL	77407.000000	47.354064	14.036828	1.000000	50.000000	52.000000	53.000000	63.0000	
POLICY_SALES_CHANNEL_GROUP	77407.000000	5.123916	1.585383	1.000000	5.000000	6.000000	6.000000	8.0000	
POLICY_BRANCH	77407.000000	0.474531	0.499354	0.000000	0.000000	0.000000	1.000000	1.0000	
POLICY_MIN_AGE	77407.000000	42.627359	10.715267	18.000000	34.000000	41.000000	50.000000	86.0000	
POLICY_MIN_DRIVING_EXPERIENCE	77407.000000	35.122017	200.828668	0.000000	8.000000	14.000000	19.000000	2015.0000	
VEHICLE_MAKE	77407.000000	1669.598938	141.778605	765.000000	1663.000000	1680.000000	1733.000000	2250.0000	
VEHICLE_MODEL	77407.000000	1669.894816	112.082830	1464.000000	1576.000000	1670.000000	1716.000000	2013.0000	
VEHICLE_ENGINE_POWER	77407.000000	154.274499	54.257499	0.000000	123.000000	146.000000	171.000000	2000.0000	
VEHICLE_IN_CREDIT	77407.000000	0.320410	0.466637	0.000000	0.000000	0.000000	1.000000	1.0000	
VEHICLE_SUM_INSURED	77407.000000	980058.319535	687620.305638	0.000000	558246.000000	807500.000000	1164990.000000	9449000.0000	
POLICY_INTERMEDIARY	77407.000000	1695.273812	276.993010	1329.000000	1479.000000	1607.000000	1955.000000	2400.0000	
INSURER_GENDER	77407.000000	0.632023	0.482258	0.000000	0.000000	1.000000	1.000000	1.0000	
POLICY_CLM_N	77407.000000	0.604429	1.078427	0.000000	0.000000	0.000000	1.000000	5.0000	
POLICY_CLM_GLT_N	77407.000000	0.488380	0.975538	0.000000	0.000000	0.000000	1.000000	5.0000	
POLICY_PRV_CLM_N	77407.000000	0.371517	0.856496	0.000000	0.000000	0.000000	0.000000	5.0000	
POLICY_PRV_CLM_GLT_N	77407.000000	0.273851	0.718776	0.000000	0.000000	0.000000	0.000000	5.0000	
CLIENT_HAS_DAGO	77407.000000	0.275970	0.447005	0.000000	0.000000	0.000000	1.000000	1.0000	
CLIENT_HAS_OSAGO	77407.000000	0.550041	0.497493	0.000000	0.000000	1.000000	1.000000	1.0000	
POLICY_COURT_SIGN	77407.000000	0.000982	0.031319	0.000000	0.000000	0.000000	0.000000	1.0000	
CLAIM_AVG_ACC_ST_PRD	77407.000000	3.903052	17.933455	0.000000	0.000000	0.000000	0.000000	0.000000	737.0000
POLICY_HAS_COMPLAINTS	77407.000000	0.007299	0.085123	0.000000	0.000000	0.000000	0.000000	0.000000	1.0000
POLICY_YEARS_RENEWED_N	77407.000000	1.495356	1.602149	0.000000	0.000000	1.000000	2.000000	10.0000	
POLICY_DEDUCT_VALUE	77407.000000	5965.020556	10397.744838	0.000000	0.000000	0.000000	10000.000000	120873.0000	
CLIENT_REGISTRATION_REGION	77407.000000	1688.815262	291.301151	662.000000	1705.000000	1705.000000	1968.000000	1968.0000	
POLICY_PRICE_CHANGE	77407.000000	-0.033503	0.837380	-1.000000	-0.170000	0.000000	0.050000	60.0200	

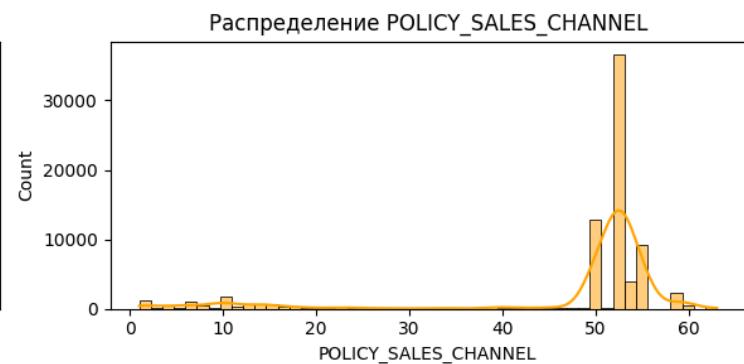
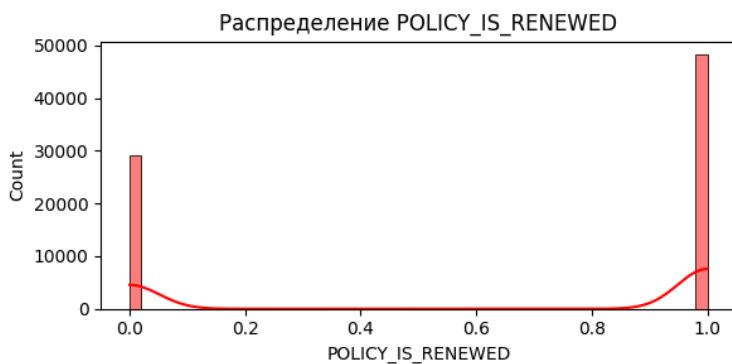
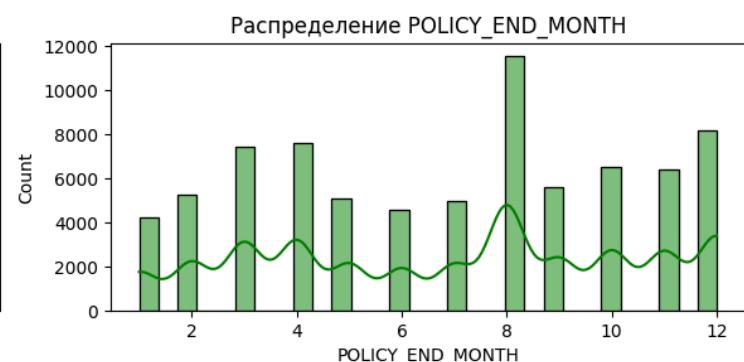
```
df.duplicated().sum()
```

```
np.int64(4)
```

Дубликаты появились, но до удаления столбца POLICE\_ID их не было, это просто объекты с одинаковыми значениями по всем признакам

```
correlation_matrix = df.corr()
plt.figure(figsize= (50, 26))
sns.heatmap(correlation_matrix, annot = True)
```





```
plt.figure(figsize=(12, 6))

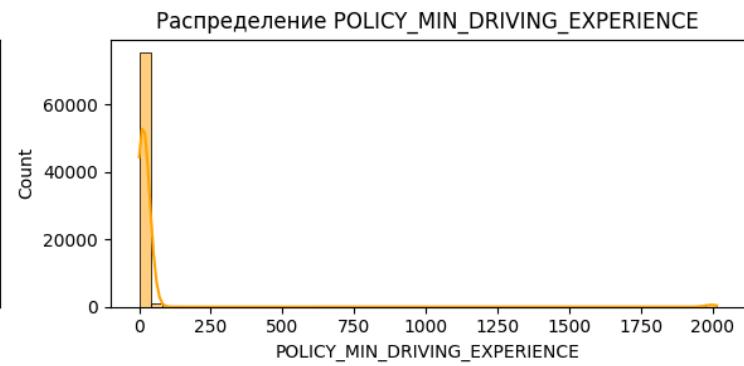
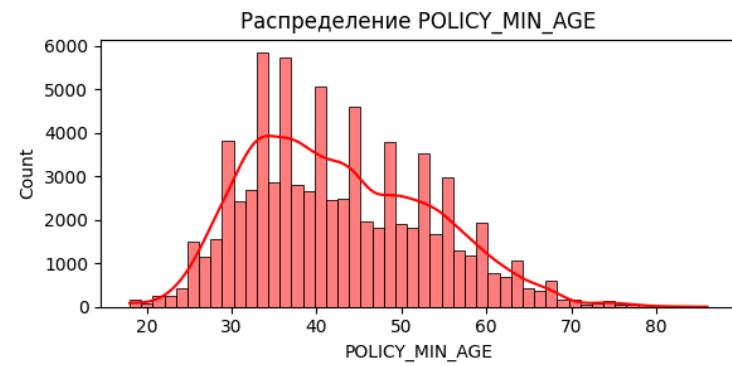
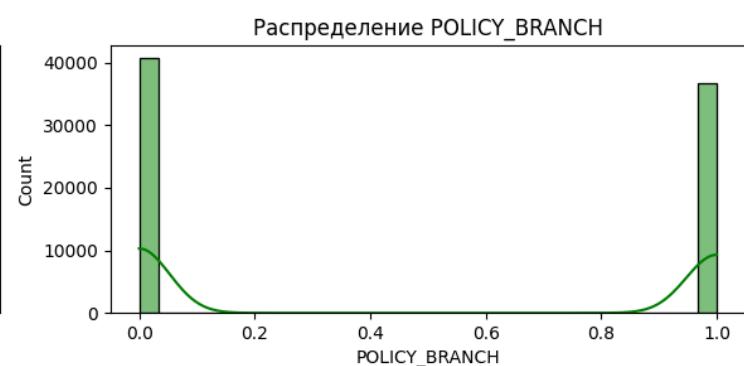
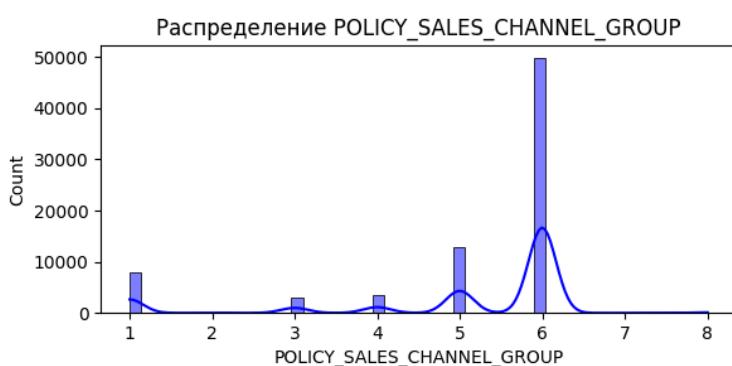
plt.subplot(2, 2, 1)
sns.histplot(df['POLICY_SALES_CHANNEL_GROUP'], bins=50, kde=True, color='blue')
plt.title('Распределение POLICY_SALES_CHANNEL_GROUP')

plt.subplot(2, 2, 2)
sns.histplot(df['POLICY_BRANCH'], bins=30, kde=True, color='green')
plt.title('Распределение POLICY_BRANCH')

plt.subplot(2, 2, 3)
sns.histplot(df['POLICY_MIN_AGE'], bins=50, kde=True, color='red')
plt.title('Распределение POLICY_MIN_AGE')

plt.subplot(2, 2, 4)
sns.histplot(df['POLICY_MIN_DRIVING_EXPERIENCE'], bins=50, kde=True, color='orange')
plt.title('Распределение POLICY_MIN_DRIVING_EXPERIENCE')

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



Подозреваем наличие выбросов у признака POLICY\_MIN\_DRIVING\_EXPERIENCE - на правой границе диапазона рост частоты объектов

```

plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
sns.histplot(df['VEHICLE_MAKE'], bins=50, kde=True, color='blue')
plt.title('Распределение VEHICLE_MAKE')

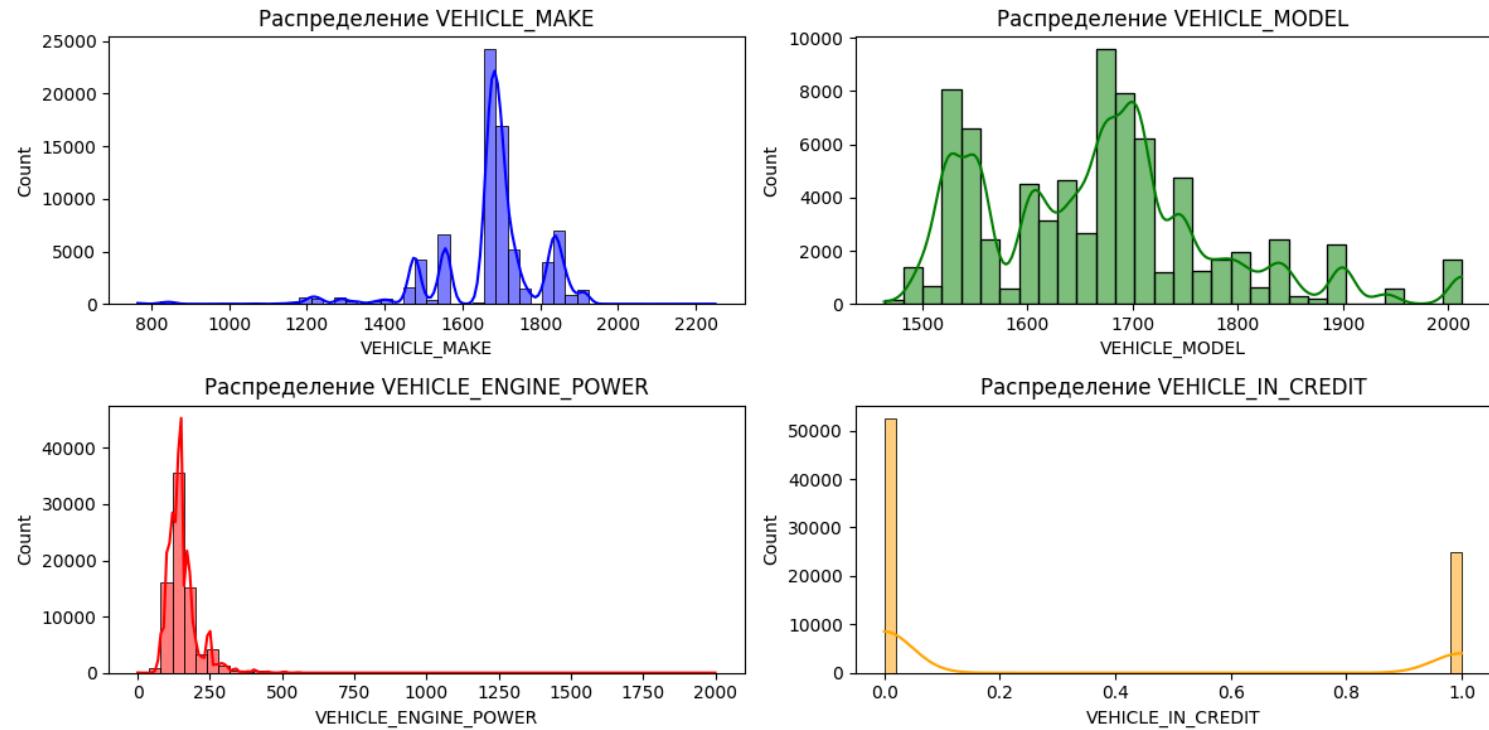
plt.subplot(2, 2, 2)
sns.histplot(df['VEHICLE_MODEL'], bins=30, kde=True, color='green')
plt.title('Распределение VEHICLE_MODEL')

plt.subplot(2, 2, 3)
sns.histplot(df['VEHICLE_ENGINE_POWER'], bins=50, kde=True, color='red')
plt.title('Распределение VEHICLE_ENGINE_POWER')

plt.subplot(2, 2, 4)
sns.histplot(df['VEHICLE_IN_CREDIT'], bins=50, kde=True, color='orange')
plt.title('Распределение VEHICLE_IN_CREDIT')

plt.tight_layout()
plt.show()

```



```

plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
sns.histplot(df['VEHICLE_SUM_INSURED'], bins=50, kde=True, color='blue')
plt.title('Распределение VEHICLE_SUM_INSURED')

plt.subplot(2, 2, 2)
sns.histplot(df['POLICY_INTERMEDIARY'], bins=30, kde=True, color='green')
plt.title('Распределение POLICY_INTERMEDIARY')

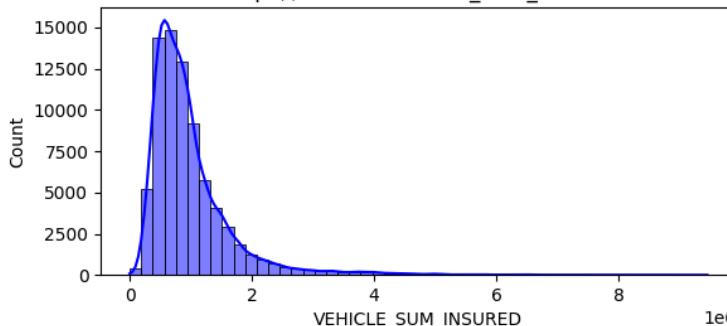
plt.subplot(2, 2, 3)
sns.histplot(df['INSURER_GENDER'], bins=50, kde=True, color='red')
plt.title('Распределение INSURER_GENDER')

plt.subplot(2, 2, 4)
sns.histplot(df['POLICY_CLM_N'], bins=50, kde=True, color='orange')
plt.title('Распределение POLICY_CLM_N')

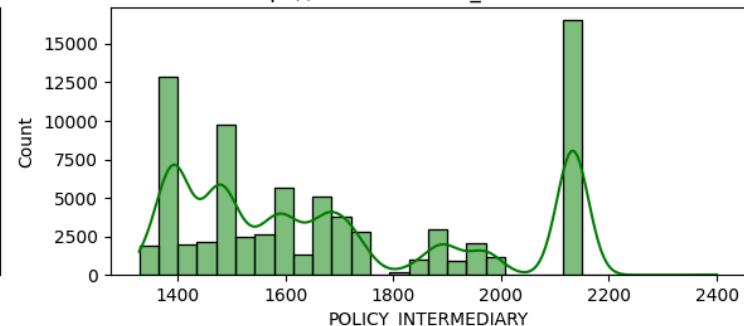
plt.tight_layout()
plt.show()

```

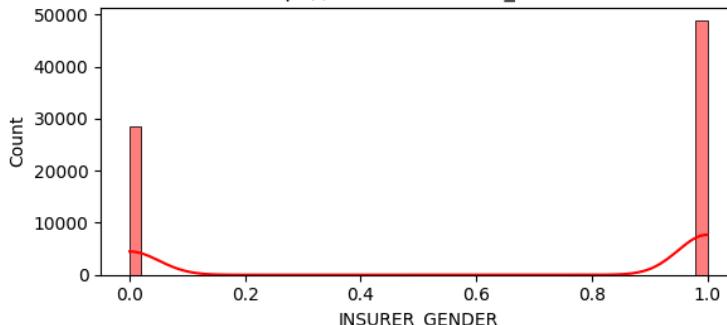
Распределение VEHICLE\_SUM\_INSURED



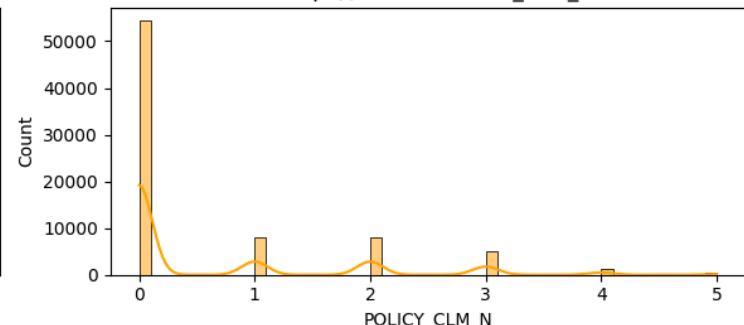
Распределение POLICY\_INTERMEDIARY



Распределение INSURER\_GENDER



Распределение POLICY\_CLM\_N



```
plt.figure(figsize=(12, 6))
```

```
plt.subplot(2, 2, 1)
sns.histplot(df['POLICY_CLM_GLT_N'], bins=50, kde=True, color='blue')
plt.title('Распределение POLICY_CLM_GLT_N')
```

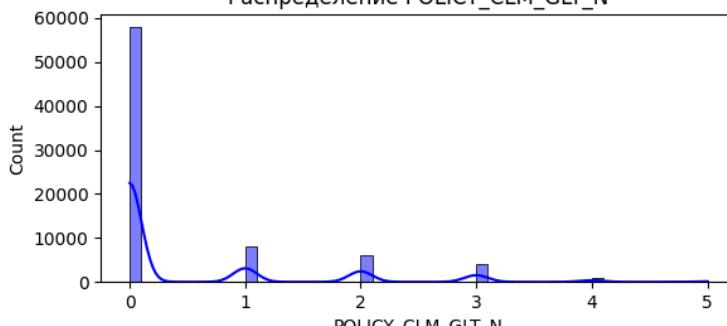
```
plt.subplot(2, 2, 2)
sns.histplot(df['POLICY_PRV_CLM_N'], bins=30, kde=True, color='green')
plt.title('Распределение POLICY_PRV_CLM_N')
```

```
plt.subplot(2, 2, 3)
sns.histplot(df['POLICY_PRV_CLM_GLT_N'], bins=50, kde=True, color='red')
plt.title('Распределение POLICY_PRV_CLM_GLT_N')
```

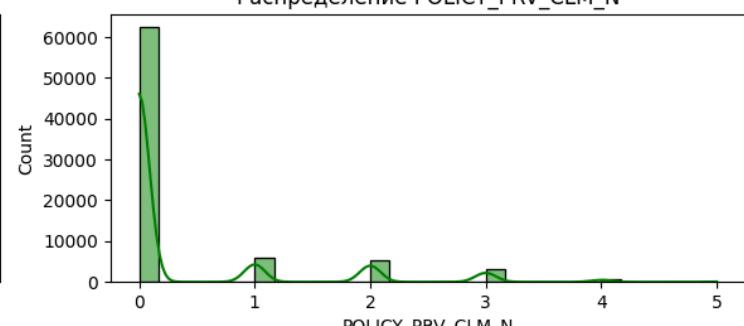
```
plt.subplot(2, 2, 4)
sns.histplot(df['CLIENT_HAS_DAGO'], bins=50, kde=True, color='orange')
plt.title('Распределение CLIENT_HAS_DAGO')
```

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

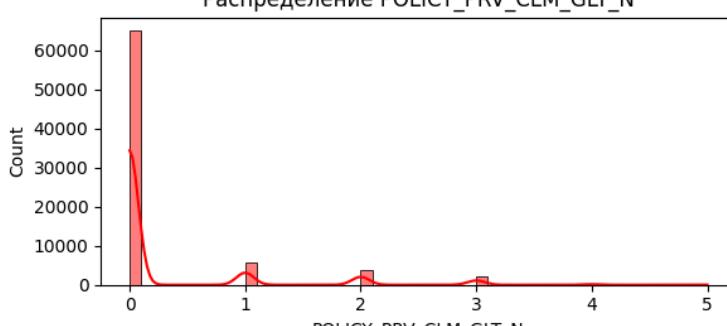
Распределение POLICY\_CLM\_GLT\_N



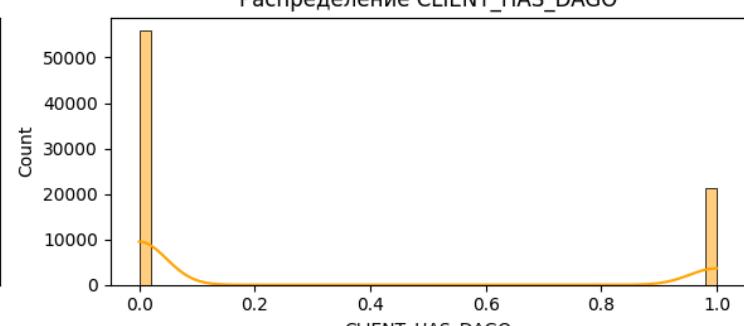
Распределение POLICY\_PRV\_CLM\_N



Распределение POLICY\_PRV\_CLM\_GLT\_N



Распределение CLIENT\_HAS\_DAGO



```
plt.figure(figsize=(12, 6))
```

```
plt.subplot(2, 2, 1)
```

```

sns.histplot(df['CLIENT_HAS_OSAGO'], bins=50, kde=True, color='blue')
plt.title('Распределение CLIENT_HAS_OSAGO')

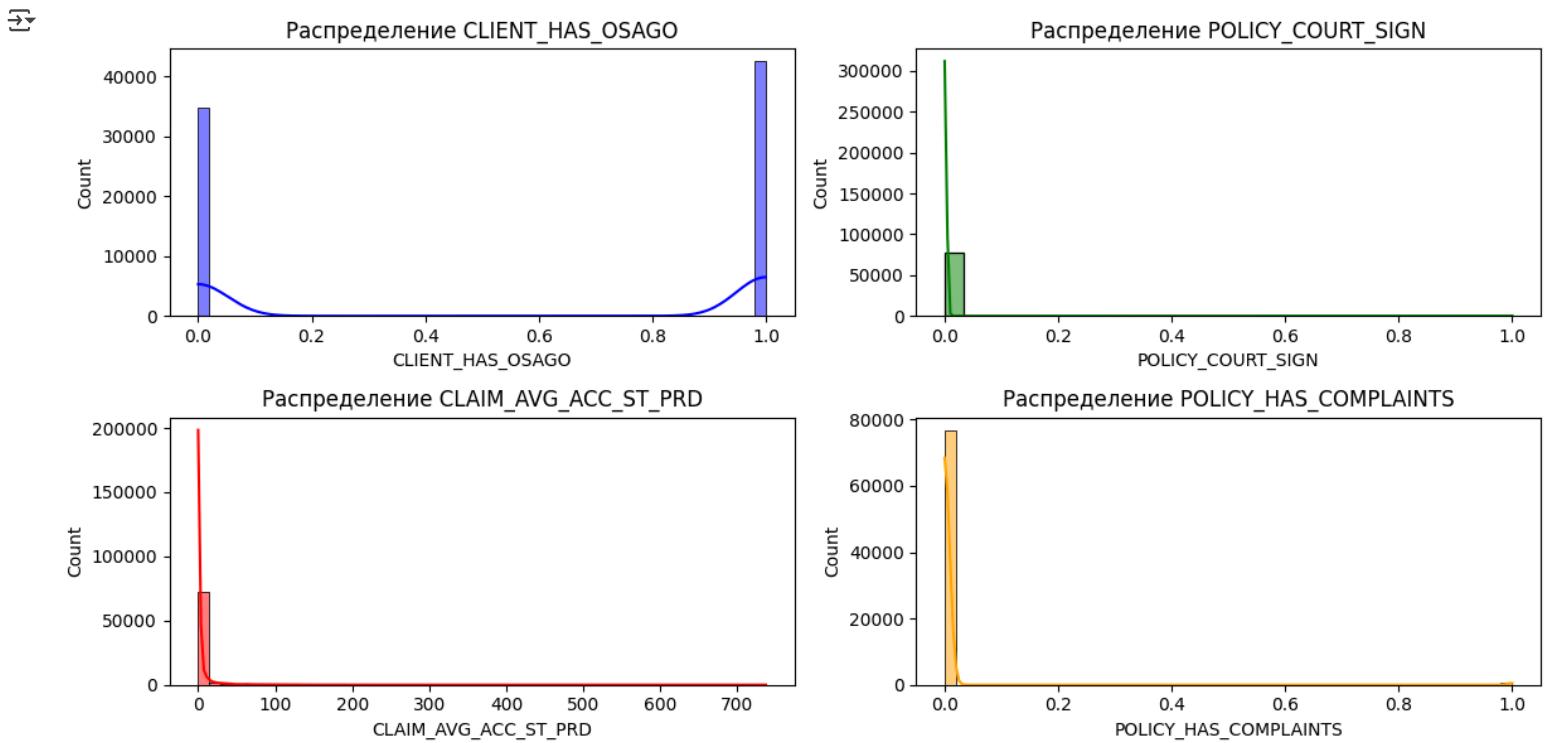
plt.subplot(2, 2, 2)
sns.histplot(df['POLICY_COURT_SIGN'], bins=30, kde=True, color='green')
plt.title('Распределение POLICY_COURT_SIGN')

plt.subplot(2, 2, 3)
sns.histplot(df['CLAIM_AVG_ACC_ST_PRD'], bins=50, kde=True, color='red')
plt.title('Распределение CLAIM_AVG_ACC_ST_PRD')

plt.subplot(2, 2, 4)
sns.histplot(df['POLICY_HAS_COMPLAINTS'], bins=50, kde=True, color='orange')
plt.title('Распределение POLICY_HAS_COMPLAINTS')

plt.tight_layout()
plt.show()

```



```

plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
sns.histplot(df['POLICY_YEARS_RENEWED_N'], bins=50, kde=True, color='blue')
plt.title('Распределение POLICY_YEARS_RENEWED_N')

plt.subplot(2, 2, 2)
sns.histplot(df['POLICY_DEDUCT_VALUE'], bins=30, kde=True, color='green')
plt.title('Распределение POLICY_DEDUCT_VALUE')

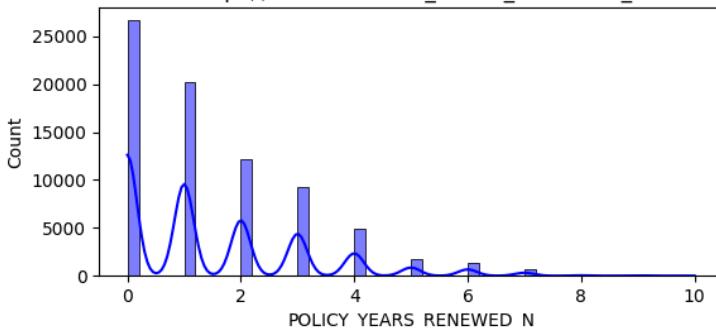
plt.subplot(2, 2, 3)
sns.histplot(df['CLIENT_REGISTRATION_REGION'], bins=50, kde=True, color='red')
plt.title('Распределение CLIENT_REGISTRATION_REGION')

plt.subplot(2, 2, 4)
sns.histplot(df['POLICY_PRICE_CHANGE'], bins=50, kde=True, color='orange')
plt.title('Распределение POLICY_PRICE_CHANGE')

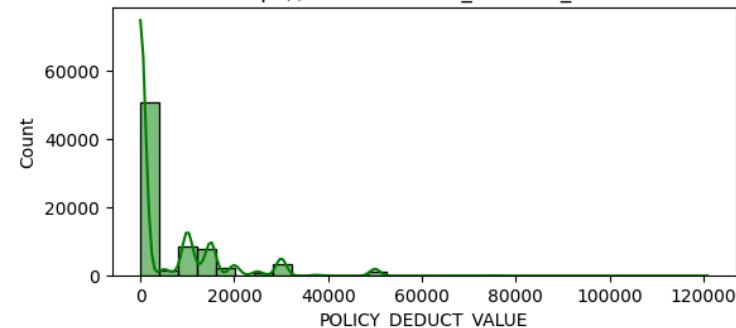
plt.tight_layout()
plt.show()

```

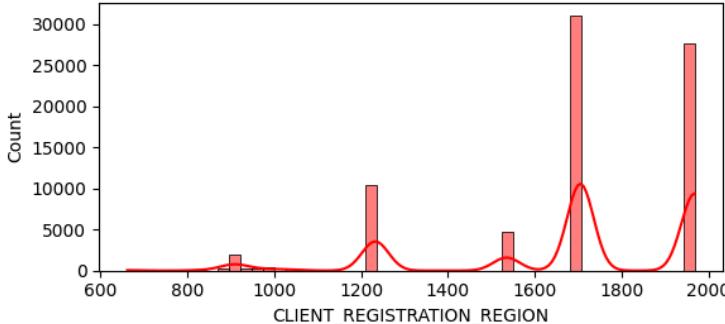
### Распределение POLICY\_YEARS\_RENEWED\_N



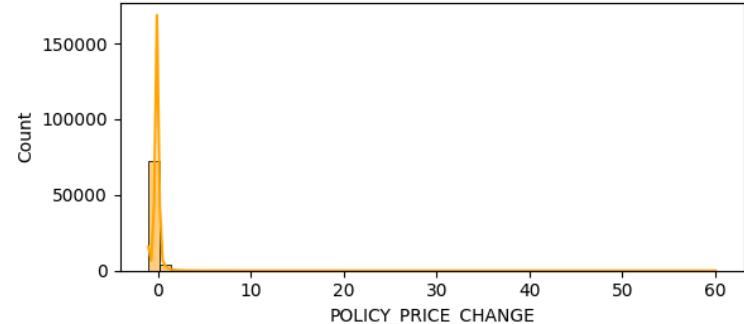
### Распределение POLICY\_DEDUCT\_VALUE



### Распределение CLIENT\_REGISTRATION\_REGION



### Распределение POLICY\_PRICE\_CHANGE



Проверяем гипотезу о выбросах в признаке POLICY\_MIN\_DRIVING\_EXPERIENCE

```
count_values = df['POLICY_MIN_DRIVING_EXPERIENCE'].value_counts().to_frame()
print(count_values.sort_values('POLICY_MIN_DRIVING_EXPERIENCE').to_string(max_rows=20))
```

POLICY_MIN_DRIVING_EXPERIENCE	count
0	2570
1	1450
2	1559
3	1590
4	1757
5	2237
6	3017
7	3242
8	3067
9	3361
...	...
2006	28
2007	21
2008	24
2009	9
2010	4
2011	7
2012	7
2013	3
2014	4
2015	4

Отвергаем гипотезу о наличии выбросов - объектов со сходими максимальными значениями (2006-2015) довольно много, количество уменьшается ближе к самой правой границе диапазона значений

Исследуем целевой класс POLICY\_IS\_RENEWED на сбалансированность

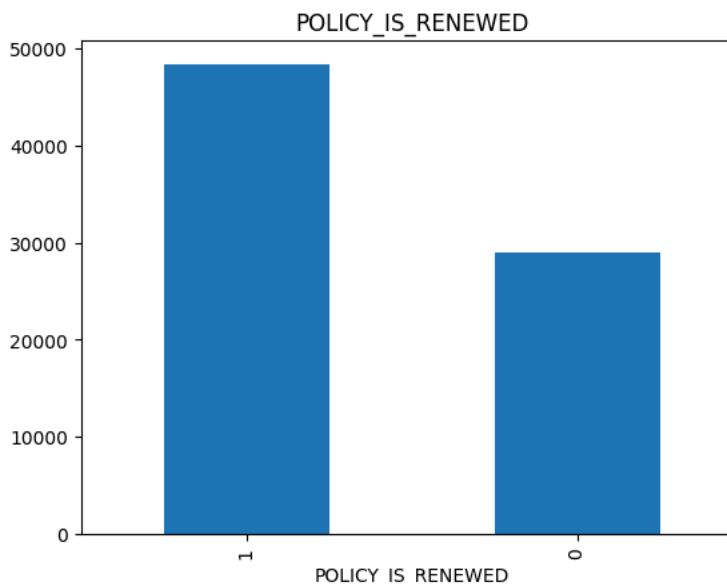
```
df['POLICY_IS_RENEWED'].value_counts(sort = False)
```

POLICY_IS_RENEWED	count
1	48376
0	29031

dtype: int64

```
df['POLICY_IS_RENEWED'].value_counts().plot(kind='bar');
plt.title('POLICY_IS_RENEWED')
```

```
→ Text(0.5, 1.0, 'POLICY_IS_RENEWED')
```



Наблюдается небольшой дисбаланс классов

Создаем матрицу объект-признак X и вектор с целевой переменной (POLICY\_IS\_RENEWED) y. Стратифицированная выборка тестового и обучающего датасета

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn import metrics
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score

X = df.drop("POLICY_IS_RENEWED", axis=1)
y = df["POLICY_IS_RENEWED"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Масштабируем признаки

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Применим различные модели классификации. Обратим внимание, что в нашем случае будет разный вес ошибок - ошибочно классифицированные как "непродляющие" страховку клиенты, которые впоследствии все-таки продлили ее - это лучше, чем ошибочно предсказанное продление полиса страхования, которое в реальности не состоялось. Будем отражать в настройках моделей дисбаланс классов.

## 1. Бустинговая модель классификации CatBoost

```
pip install catboost
```

```
→ Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
99.2/99.2 MB 26.3 MB/s eta 0:00:00
Installing collected packages: catboost
```

Successfully installed catboost-1.2.8

```
from catboost import CatBoostClassifier
```

Модель **CatBoost** "из коробки", сбалансированная по классам

```
model0 = CatBoostClassifier(auto_class_weights='Balanced')
model0.fit(X_train, y_train)
y_pred0 = model0.predict(X_test)
```

847: learn: 0.5265192 total: 6.46s remaining: 1.16s  
848: learn: 0.5264646 total: 6.47s remaining: 1.15s  
849: learn: 0.5263983 total: 6.48s remaining: 1.14s  
850: learn: 0.5263412 total: 6.49s remaining: 1.14s  
851: learn: 0.5262972 total: 6.5s remaining: 1.13s  
852: learn: 0.5262376 total: 6.51s remaining: 1.12s  
853: learn: 0.5261814 total: 6.52s remaining: 1.11s  
854: learn: 0.5261292 total: 6.52s remaining: 1.11s  
855: learn: 0.5260835 total: 6.53s remaining: 1.1s  
856: learn: 0.5260553 total: 6.54s remaining: 1.09s  
857: learn: 0.5259977 total: 6.54s remaining: 1.08s  
858: learn: 0.5259180 total: 6.55s remaining: 1.07s  
859: learn: 0.5258381 total: 6.56s remaining: 1.07s  
860: learn: 0.5257363 total: 6.56s remaining: 1.06s  
861: learn: 0.5256719 total: 6.57s remaining: 1.05s  
862: learn: 0.5256316 total: 6.58s remaining: 1.04s  
863: learn: 0.5255591 total: 6.58s remaining: 1.04s  
864: learn: 0.5254784 total: 6.59s remaining: 1.03s  
865: learn: 0.5254298 total: 6.6s remaining: 1.02s  
866: learn: 0.5253837 total: 6.6s remaining: 1.01s  
867: learn: 0.5253580 total: 6.61s remaining: 1s  
868: learn: 0.5252688 total: 6.62s remaining: 997ms  
869: learn: 0.5252095 total: 6.62s remaining: 990ms  
870: learn: 0.5251675 total: 6.63s remaining: 982ms  
871: learn: 0.5250678 total: 6.63s remaining: 974ms  
872: learn: 0.5249811 total: 6.64s remaining: 966ms  
873: learn: 0.5249125 total: 6.65s remaining: 958ms  
874: learn: 0.5248325 total: 6.65s remaining: 951ms  
875: learn: 0.5247583 total: 6.66s remaining: 943ms  
876: learn: 0.5246972 total: 6.67s remaining: 935ms  
877: learn: 0.5246336 total: 6.67s remaining: 927ms  
878: learn: 0.5246134 total: 6.68s remaining: 920ms  
879: learn: 0.5245558 total: 6.69s remaining: 912ms  
880: learn: 0.5244954 total: 6.69s remaining: 904ms  
881: learn: 0.5244204 total: 6.7s remaining: 897ms  
882: learn: 0.5243739 total: 6.71s remaining: 889ms  
883: learn: 0.5243223 total: 6.71s remaining: 881ms  
884: learn: 0.5242602 total: 6.72s remaining: 873ms  
885: learn: 0.5241886 total: 6.73s remaining: 866ms  
886: learn: 0.5240925 total: 6.73s remaining: 858ms  
887: learn: 0.5240659 total: 6.74s remaining: 850ms  
888: learn: 0.5239789 total: 6.75s remaining: 843ms  
889: learn: 0.5239151 total: 6.76s remaining: 836ms  
890: learn: 0.5238543 total: 6.77s remaining: 828ms  
891: learn: 0.5238046 total: 6.78s remaining: 821ms  
892: learn: 0.5237310 total: 6.79s remaining: 813ms  
893: learn: 0.5236879 total: 6.8s remaining: 806ms  
894: learn: 0.5236368 total: 6.81s remaining: 799ms  
895: learn: 0.5235777 total: 6.82s remaining: 792ms  
896: learn: 0.5235289 total: 6.83s remaining: 784ms  
897: learn: 0.5234772 total: 6.83s remaining: 776ms  
898: learn: 0.5234074 total: 6.84s remaining: 769ms  
899: learn: 0.5233468 total: 6.85s remaining: 761ms  
900: learn: 0.5233088 total: 6.86s remaining: 753ms  
901: learn: 0.5232627 total: 6.86s remaining: 746ms  
902: learn: 0.5231822 total: 6.87s remaining: 738ms  
903: learn: 0.5231531 total: 6.88s remaining: 730ms  
904: learn: 0.5230740 total: 6.88s remaining: 723ms  
905: learn: 0.5230091 total: 6.89s remaining: 715ms

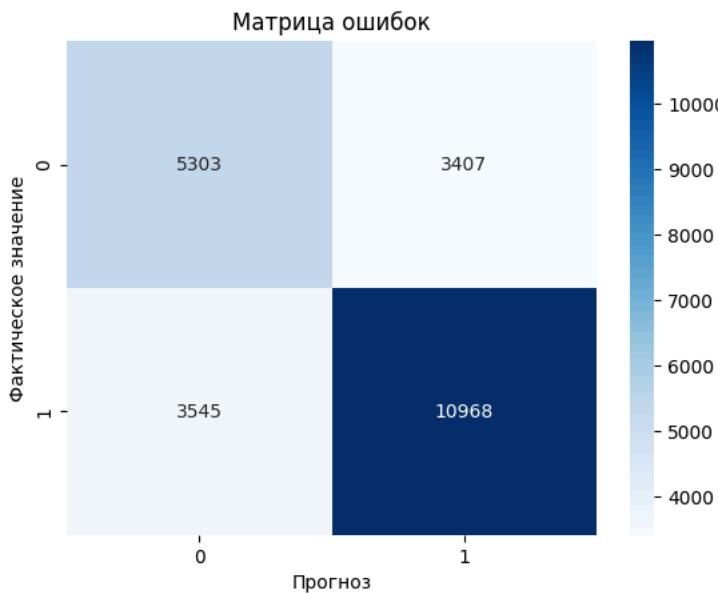
```
print(classification_report(y_test, y_pred0))
```

	precision	recall	f1-score	support
0	0.60	0.61	0.60	8710
1	0.76	0.76	0.76	14513
accuracy			0.70	23223
macro avg	0.68	0.68	0.68	23223
weighted avg	0.70	0.70	0.70	23223

```
confusion_mat = confusion_matrix(y_test, y_pred0)
print("Точность:", accuracy_score(y_test, y_pred0))
print("Точность:", precision_score(y_test, y_pred0, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred0, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred0, average = 'binary'))
```

→ Точность: 0.7006416053050855  
Точность: 0.7629913043478261  
Полнота: 0.7557362364776408  
F1-мера: 0.759346441428967

```
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();
```



Модель **CatBoost** после тюнинга и подбора гиперпараметров

```
model1 = CatBoostClassifier(iterations=4000, auto_class_weights='SqrtBalanced')
model1.fit(X_train, y_train)
y_pred1 = model1.predict(X_test)
```

```
3599: learn: 0.509473      total: 25.2s  remaining: 2.8s
3600: learn: 0.5094613     total: 25.2s  remaining: 2.79s
3601: learn: 0.5094421     total: 25.2s  remaining: 2.78s
```

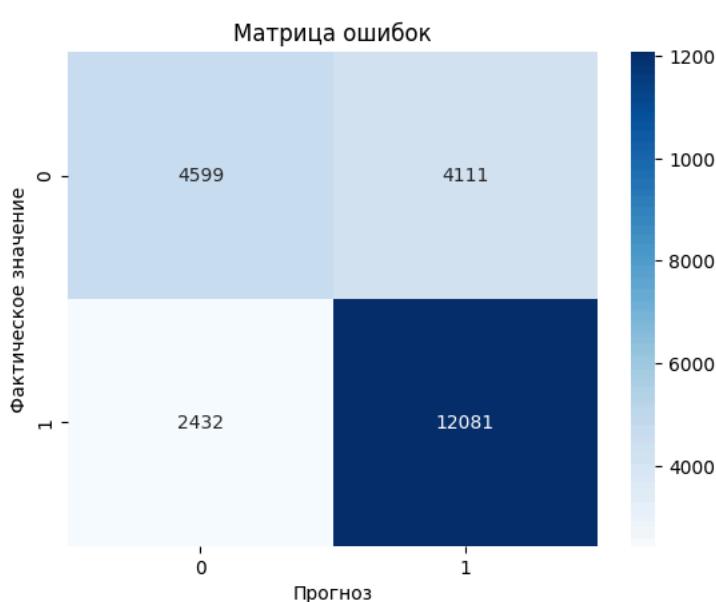
```
print(classification_report(y_test, y_pred1))
```

	precision	recall	f1-score	support
0	0.65	0.53	0.58	8710
1	0.75	0.83	0.79	14513
accuracy			0.72	23223
macro avg	0.70	0.68	0.69	23223
weighted avg	0.71	0.72	0.71	23223

```
confusion_mat = confusion_matrix(y_test, y_pred1)
print("Точность:", accuracy_score(y_test, y_pred1))
print("Точность:", precision_score(y_test, y_pred1, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred1, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred1, average = 'binary'))
```

```
→ Точность: 0.7182534556258882
Точность: 0.7461091897233202
Полнота: 0.8324261007372701
F1-мера: 0.7869076697606253
```

```
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();
```



## Бэггинг: RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

sample_weights = [0.15 if y == 0 else 1 for y in y_train]

model2 = RandomForestClassifier(n_estimators=1000)
model2.fit(X_train, y_train, sample_weight=sample_weights)
y_pred2 = model2.predict(X_test)
print(classification_report(y_test, y_pred2))
```

	precision	recall	f1-score	support
0	0.65	0.49	0.56	8710
1	0.73	0.84	0.78	14513
accuracy			0.71	23223
macro avg	0.69	0.67	0.67	23223
weighted avg	0.70	0.71	0.70	23223

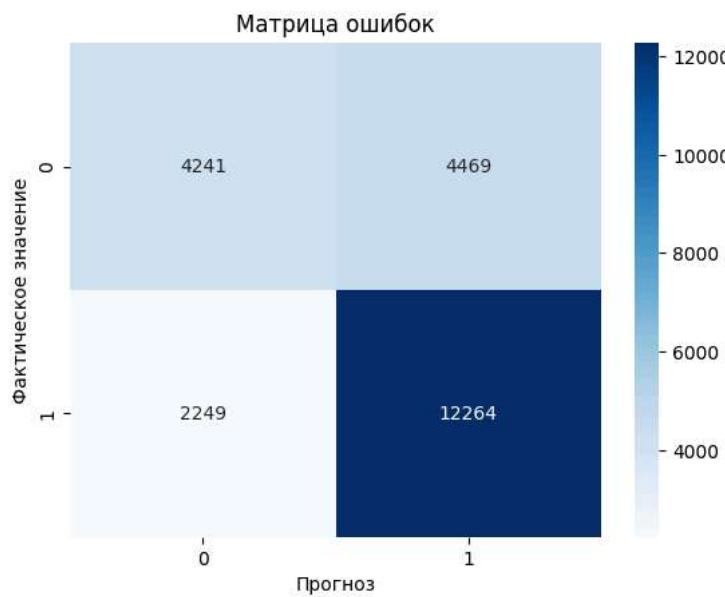
```
confusion_mat = confusion_matrix(y_test, y_pred2)
print("Точность:", accuracy_score(y_test, y_pred2))
print("Точность:", precision_score(y_test, y_pred2, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred2, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred2, average = 'binary'))
```

```
→ Точность: 0.7107608836067691
Точность: 0.7330224772835964
Полнота: 0.8448976779439124
F1-мера: 0.7849940782945488
```

```

sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();

```

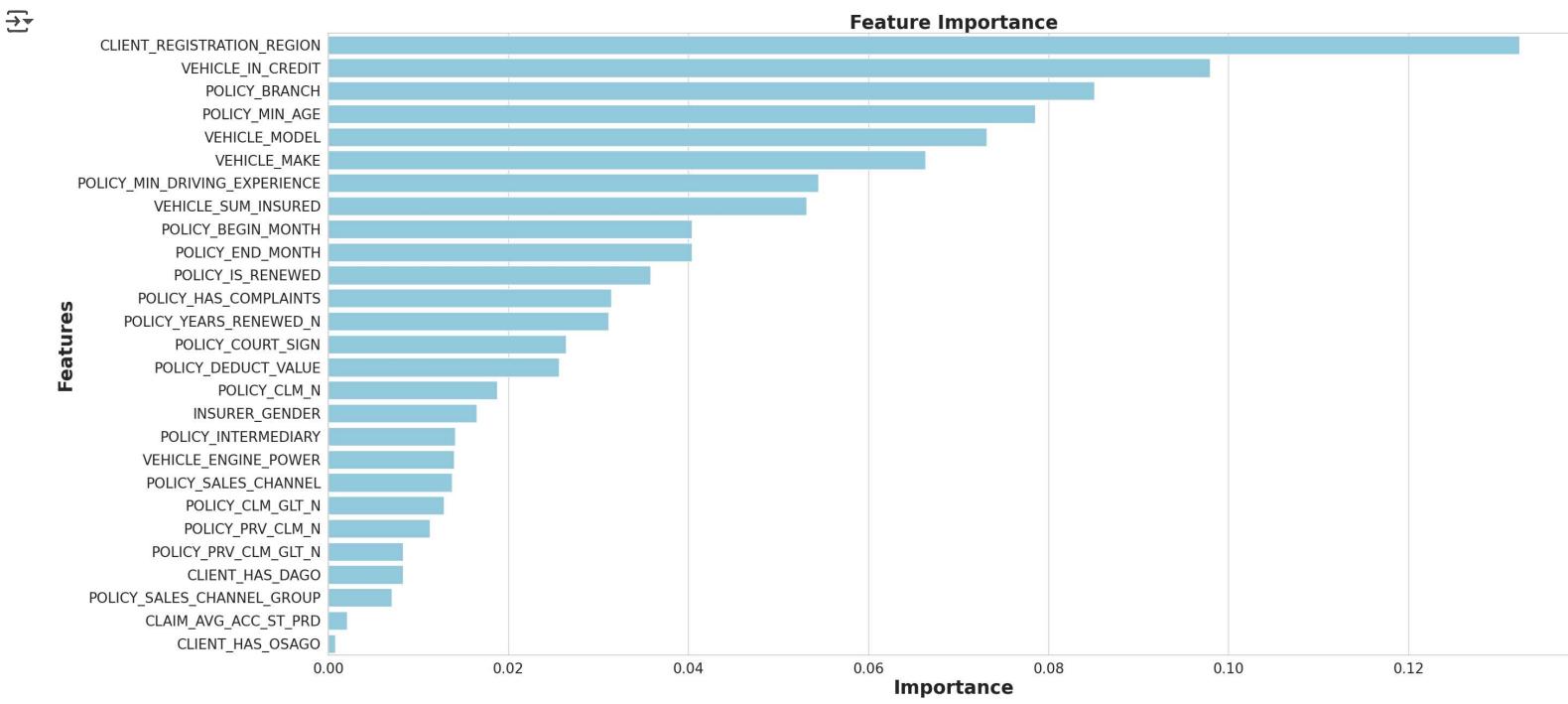


Визуализируем важность признаков для предсказания целевого признака:

```

feats = {}
for feature, importance in zip(df.columns, model2.feature_importances_):
    feats[feature] = importance
importances = pd.DataFrame.from_dict(feats, orient='index').rename(columns={0: 'Gini-Importance'})
importances = importances.sort_values(by='Gini-Importance', ascending=False)
importances = importances.reset_index()
importances = importances.rename(columns={'index': 'Features'})
sns.set(font_scale = 5)
sns.set(style="whitegrid", color_codes=True, font_scale = 1.7)
fig, ax = plt.subplots()
fig.set_size_inches(30,15)
sns.barplot(x=importances['Gini-Importance'], y=importances['Features'], data=importances, color='skyblue')
plt.xlabel('Importance', fontsize=25, weight = 'bold')
plt.ylabel('Features', fontsize=25, weight = 'bold')
plt.title('Feature Importance', fontsize=25, weight = 'bold')
display(plt.show())
display(importances)

```



None

**Features Gini-Importance**

0	CLIENT_REGISTRATION_REGION	0.132308
1	VEHICLE_IN_CREDIT	0.097913
2	POLICY_BRANCH	0.085123
3	POLICY_MIN_AGE	0.078517
4	VEHICLE_MODEL	0.073145
5	VEHICLE_MAKE	0.066327
6	POLICY_MIN_DRIVING_EXPERIENCE	0.054464
7	VEHICLE_SUM_INSURED	0.053118
8	POLICY_BEGIN_MONTH	0.040427
9	POLICY_END_MONTH	0.040408
10	POLICY_IS_RENEWED	0.035814
11	POLICY_HAS_COMPLAINTS	0.031467
12	POLICY_YEARS_RENEWED_N	0.031114
13	POLICY_COURT_SIGN	0.026431
14	POLICY_DEDUCT_VALUE	0.025613
15	POLICY_CLM_N	0.018752
16	INSURER_GENDER	0.016483
17	POLICY_INTERMEDIARY	0.014114
18	VEHICLE_ENGINE_POWER	0.013971
19	POLICY_SALES_CHANNEL	0.013724
20	POLICY_CLM_GLT_N	0.012860
21	POLICY_PRV_CLM_N	0.011306
22	POLICY_PRV_CLM_GLT_N	0.008342
23	CLIENT_HAS_DAGO	0.008307
24	POLICY_SALES_CHANNEL_GROUP	0.007065
25	CLAIM_AVG_ACC_ST_PRD	0.002116
26	CLIENT_HAS_OSAGO	0.000771

Видим, что все перекодированные нами на основе взаимосвязи с целевым признаком категориальные признаки имеют большую важность для модели

### Метод опорных векторов

```
from sklearn.svm import SVC

model3 = SVC(max_iter=5000, kernel='linear', class_weight='balanced')

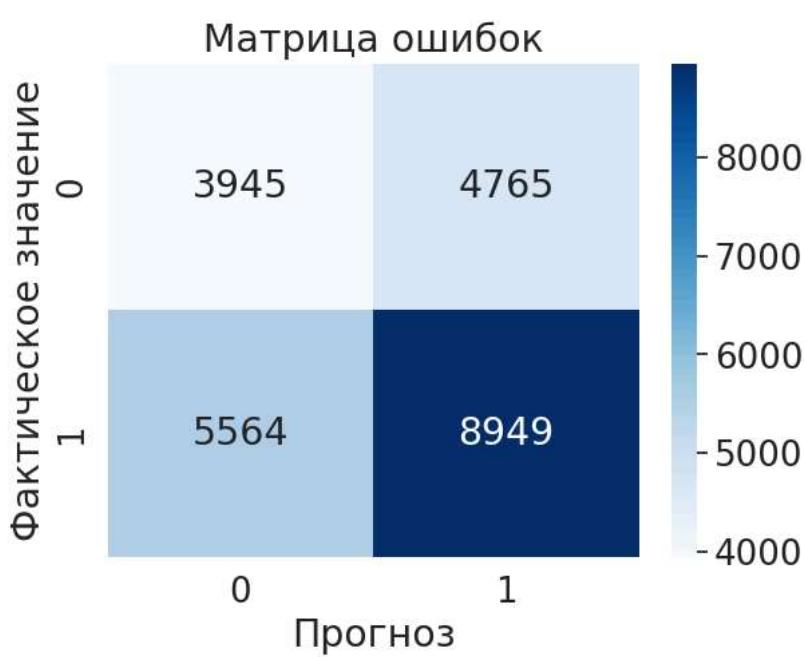
model3.fit(X_train, y_train)
y_pred3 = model3.predict(X_test)
print(classification_report(y_test, y_pred3))

→ /usr/local/lib/python3.11/dist-packages/sklearn/svm/_base.py:305: ConvergenceWarning: Solver terminated early (max_iter=5000). Consider pre-processing
  warnings.warn(
      precision     recall   f1-score   support
      0           0.41      0.45      0.43     8710
      1           0.65      0.62      0.63    14513
      accuracy          0.56
      macro avg       0.53      0.53      0.53    23223
  weighted avg       0.56      0.56      0.56    23223

confusion_mat = confusion_matrix(y_test, y_pred3)
print("Точность:", accuracy_score(y_test, y_pred3))
print("Точность:", precision_score(y_test, y_pred3, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred3, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred3, average = 'binary'))

→ Точность: 0.5552254230719545
Точность: 0.6525448446842642
Полнота: 0.6166195824433267
F1-мера: 0.6340737591667552

sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();
```



## Дерево решений

```
from sklearn.tree import DecisionTreeClassifier

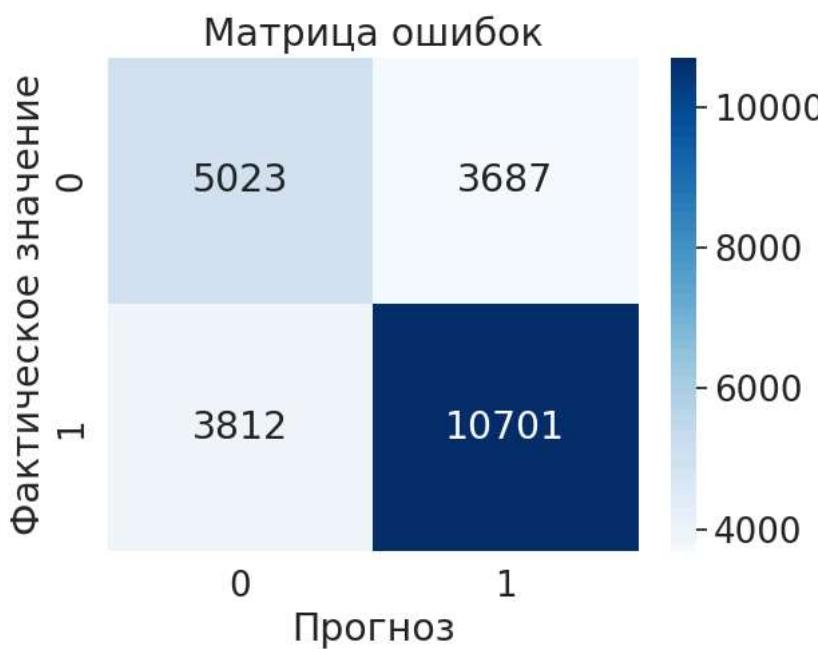
model4 = DecisionTreeClassifier(max_depth=10, min_samples_leaf=4, class_weight='balanced')
model4.fit(X_train, y_train)
y_pred4 = model4.predict(X_test)
print(classification_report(y_test, y_pred4))
```

	precision	recall	f1-score	support
0	0.57	0.58	0.57	8710
1	0.74	0.74	0.74	14513
accuracy			0.68	23223
macro avg	0.66	0.66	0.66	23223
weighted avg	0.68	0.68	0.68	23223

```
confusion_mat = confusion_matrix(y_test, y_pred4)
print("Точность:", accuracy_score(y_test, y_pred4))
print("Точность:", precision_score(y_test, y_pred4, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred4, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred4, average = 'binary'))
```

```
Точность: 0.6770873702794643
Точность: 0.7437447873227689
Полнота: 0.7373389375043065
F1-мера: 0.740528009411439
```

```
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();
```



## Логистическая регрессия

```
from sklearn.linear_model import LogisticRegression

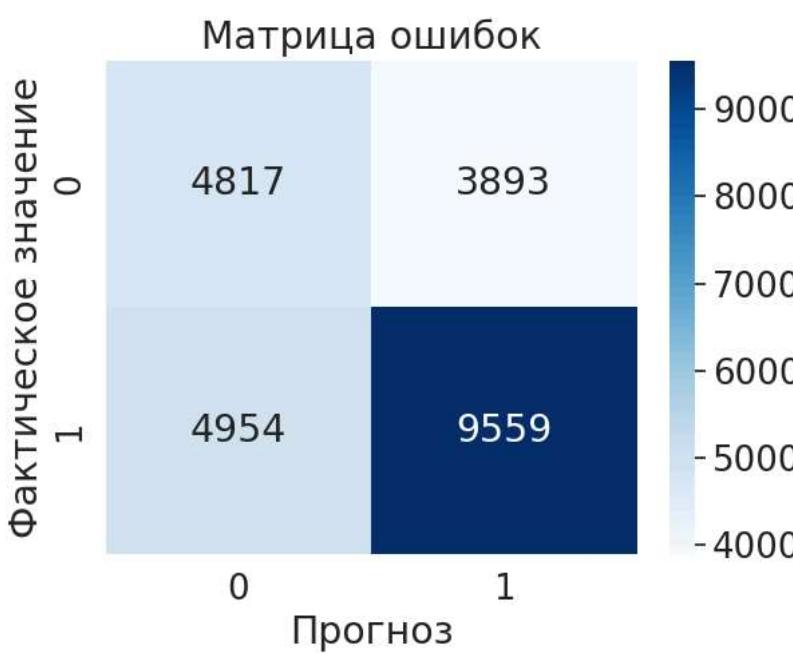
model5 = LogisticRegression(max_iter=1000, class_weight='balanced')
model5.fit(X_train, y_train)
y_pred5 = model5.predict(X_test)
print(classification_report(y_test, y_pred5))
```

	precision	recall	f1-score	support
0	0.49	0.55	0.52	8710
1	0.71	0.66	0.68	14513
accuracy			0.62	23223
macro avg	0.60	0.61	0.60	23223
weighted avg	0.63	0.62	0.62	23223

```
confusion_mat = confusion_matrix(y_test, y_pred5)
print("Точность:", accuracy_score(y_test, y_pred5))
print("Полнота:", recall_score(y_test, y_pred5, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred5, average = 'binary'))
```

```
→ Точность: 0.6190414675106576
Точность: 0.7106006541778174
Полнота: 0.6586508647419554
F1-мера: 0.6836402646164849
```

```
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();
```



## Стэкинговая модель

Построим двухуровневую модель - на первом, базовом уровне, применим RandomForestClassifier и DecisionTreeClassifier. Надстройкой над ними (моделью второго уровня) будет LogisticRegression

```
from sklearn.ensemble import StackingClassifier

# Определение базовых моделей
base_models = [
    ('rf', RandomForestClassifier(n_estimators=300, random_state=42, class_weight='balanced')),
    ('dt', DecisionTreeClassifier(max_depth=9, min_samples_leaf=4, class_weight='balanced'))
]

# Создание модели стекинга
model6 = StackingClassifier(estimators=base_models, final_estimator=LogisticRegression())
model6.fit(X_train, y_train)

# Предсказания
y_pred6 = model6.predict(X_test)
print(classification_report(y_test, y_pred6))
```

```
→
```

	precision	recall	f1-score	support
0	0.69	0.44	0.54	8710
1	0.72	0.88	0.79	14513
accuracy			0.71	23223
macro avg	0.70	0.66	0.66	23223
weighted avg	0.71	0.71	0.70	23223

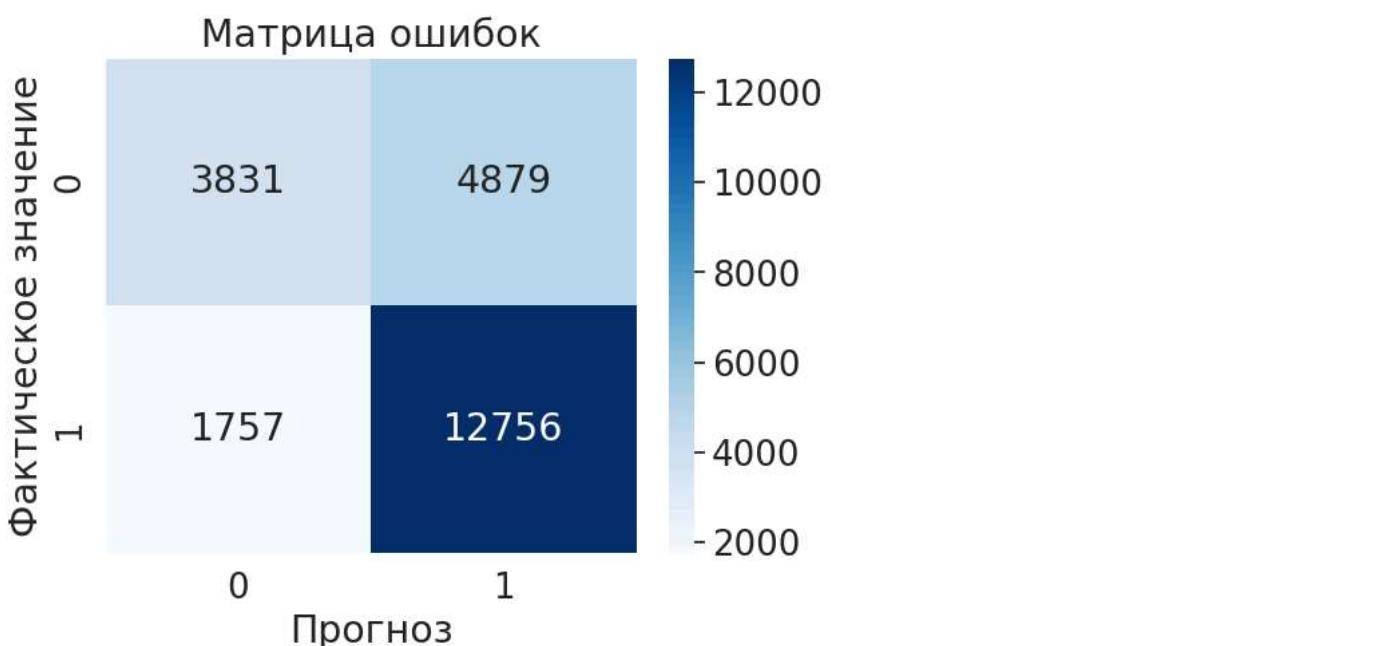
```
confusion_mat = confusion_matrix(y_test, y_pred6)
print("Точность:", accuracy_score(y_test, y_pred6))
print("Полнота:", recall_score(y_test, y_pred6, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred6, average = 'binary'))
```

```
→ Точность: 0.7142488050639453
Точность: 0.7233342784235894
Полнота: 0.8789361262316544
```

```

sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();

```



По качеству метрик среди использованных нами моделей лучшие результаты показала бустинговая модель CatBoost, RandomForestClassifier и стекинговая модель, т.е. ансамблевые модели. Совсем немного уступила модель DecisionTreeClassifier, но показала при этом существенно лучшие результаты по производительности и скорости, а также наименьшую ошибку прогноза в нужном нам классе (FP - False Positive). Возьмем ее за основу.

## TabPFN

Применим TabPFN - нейросетевую модель для анализа табличных данных, основанную на трансформере

```
pip install tabPFN
```

```

Requirement already satisfied: tabPFN in /usr/local/lib/python3.11/dist-packages (2.0.9)
Requirement already satisfied: torch<3,>=2.1 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (2.6.0+cu124)
Requirement already satisfied: scikit-learn<1.7,>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (1.6.1)
Requirement already satisfied: typing_extensions<4.4.0 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (4.13.2)
Requirement already satisfied: scipy<2,>=1.11.1 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (1.15.2)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (2.2.2)
Requirement already satisfied: einops<0.9,>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (0.8.1)
Requirement already satisfied: huggingface-hub<1,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from tabPFN) (0.30.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1,>=0.0.1->tabPFN) (4.67.1)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->tabPFN) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->tabPFN) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->tabPFN) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->tabPFN) (2025.2)
Requirement already satisfied: joblib>1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn<1.7,>=1.2.0->tabPFN) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn<1.7,>=1.2.0->tabPFN) (3.6.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (0.6.2)
Requirement already satisfied: nvidia-ncc1-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (3.2.0)

```

```
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.1->tabPFN) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch<3,>=2.1->tabPFN) (1.3.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1.4.0->tabPFN) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2>=2.1->tabPFN) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub<1,>=0.0.1->tabPFN)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub<1,>=0.0.1->tabPFN) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub<1,>=0.0.1->tabPFN) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub<1,>=0.0.1->tabPFN) (2025.4.17)
```

```
import torch
torch.cuda.empty_cache()
```

```
from tabPFN import TabPFNClassifier
```

```
# Инициализируем классификатор, ставим игнорирование максимального размера выборки (модель предусматривает ограничение 10000 строк)
```

```
model77 = TabPFNClassifier(ignore_pretraining_limits=True, balance_probabilities=True)
```

```
model77.fit(X_train, y_train)
```

```
→ /usr/local/lib/python3.11/dist-packages/tabPFN/base.py:89: UserWarning: Downloading model to /root/.cache/tabPFN/tabPFN-v2-classifier.ckpt.
```

```
    model, _, config_ = load_model_criterion_config(
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
```

```
The secret `HF_TOKEN` does not exist in your Colab secrets.
```

```
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab environment, and run the cell again.
```

```
You will be able to reuse this secret in all of your notebooks.
```

```
Please note that authentication is recommended but still optional to access public models or datasets.
```

```
    warnings.warn(
```

```
tabPFN-v2-classifier.ckpt: 100%
```

```
29.0M/29.0M [00:00<00:00, 172MB/s]
```

```
config.json: 100%
```

```
37.0/37.0 [00:00<00:00, 3.08kB/s]
```

```
/usr/local/lib/python3.11/dist-packages/tabPFN/classifier.py:422: UserWarning: Number of samples 54184 is greater than the maximum Number of samples 10000. This may lead to memory issues.
```

```
X, y, feature_names_in, n_features_in = validate_Xy_fit(
```

```
    TabPFNClassifier
```

```
    TabPFNClassifier(balance_probabilities=True, ignore_pretraining_limits=True)
```

Предсказываем значения. Так как у модели есть ограничение в 10000 строк, разбиваем всю выборку на батчи.

```
batch_size = 10000
```

```
y_pr = []
```

```
for i in range(0, len(X_test), batch_size):
    X_test_batch = X_test[i:i+batch_size]
```

```
y_pr.extend(model77.predict(X_test_batch))
```

```
y_pred77 = np.array(y_pr)
```

```
print(classification_report(y_test, y_pred77))
```

```
→ precision    recall   f1-score   support
   0       0.56      0.63      0.59      8710
   1       0.76      0.70      0.73     14513

accuracy                           0.68      23223
macro avg       0.66      0.67      0.66      23223
weighted avg    0.69      0.68      0.68      23223
```

```
confusion_mat = confusion_matrix(y_test, y_pred77)
print("Точность:", accuracy_score(y_test, y_pred77))
print("Точность:", precision_score(y_test, y_pred77, average='binary'))
print("Полнота:", recall_score(y_test, y_pred77, average='binary'))
print("F1-мера:", f1_score(y_test, y_pred77, average='binary'))
```

```
→ Точность: 0.675623304482625
Точность: 0.7609931199521388
Полнота: 0.7011644732308965
F1-мера: 0.7298547606239913
```

```
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap="Blues")
```

```
plt.xlabel("Прогноз")
```

```
plt.ylabel("Фактическое значение")
```

```
plt.title("Матрица ошибок")
```

```
plt.show();
```

## Матрица ошибок



```
#from tabPFN_extensions.interpretability.shap import get_shap_values, plot_shap
```

```
# Calculate SHAP values
# shap_values = get_shap_values(model77, X_test)

# Visualize feature importance
# plot_shap(shap_values)
```

Применим также **AutoTabPFNClassifier** - ансамбль исходных моделей с предобученными весами для лучших параметров предсказания.

```
from tabPFN_extensions.post_hoc_ensembles.sklearn_interface import AutoTabPFNClassifier
```

```
X_train_b=X_train[0:10000]
y_train_b=y_train[0:10000]
```

```
model78 = AutoTabPFNClassifier(max_time=120, device="cuda", ignore_pretraining_limits=True) # 120 секунд - время настройки
model78.fit(X_train_b, y_train_b)
```





```
warnings.warn()
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [5, 10] during transform.
warnings.warn()
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2] during transform.
warnings.warn()
/usr/local/lib/python3.11/dist-packages/tabPFN/base.py:89: UserWarning: Downloading model to /content/tabPFN-extensions/src/tabPFN_extensions/hpo/hpo_r
model, _, config_ = load_model_criterion_config()
tabPFN-v2-classifier-znskzxi4.ckpt: 100%                                         12.9M/12.9M [00:00<00:00, 311MB/s]

config.json: 100%                                         37.0/37.0 [00:00<00:00, 4.66kB/s]

INFO:tabPFN_extensions.post_hoc_ensembles.greedy_weighted_ensemble:Order of selections: [np.int64(0), np.int64(1), np.int64(0), np.int64(1), np.int64(0),
INFO:tabPFN_extensions.post_hoc_ensembles.greedy_weighted_ensemble:Val loss over iterations: [np.float64(-0.7343675719097967), np.float64(-0.7375466964
INFO:tabPFN_extensions.post_hoc_ensembles.greedy_weighted_ensemble:Model losses: [-0.73436757 -0.73083072 -0.72958065 -0.7243093 ]
INFO:tabPFN_extensions.post_hoc_ensembles.greedy_weighted_ensemble:Best weights: [0.5 0.4 0.  0.1]

▼          AutoTabPFNClassifier
AutoTabPFNClassifier(categorical_feature_indices=[3, 4, 10, 13, 14, 15, 16, 17,
                                                18, 19, 20, 22],
                      device='cuda', ignore_pretraining_limits=True,
                      max_time=120)
```

```

# Predict labels
batch_size = 10000
y_pr=[]

for i in range (0, len(X_test), batch_size):
    X_test_batch = X_test[i:i+batch_size]
    y_pr.extend(model78.predict(X_test_batch))

y_pred78 = np.array(y_pr)

→ /usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [1, 8] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [4] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [8] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2, 6] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [4, 5] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [0, 2, 10] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [8, 9] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [0, 2, 6] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2, 6] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [3, 5] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [5, 7] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [3] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [7] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [1, 2] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2, 6] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [3] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [1] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [5, 8] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2, 4] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [1, 2] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [0, 5] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [6, 10] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [7] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [6] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [2] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [7] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [1, 3] during transform
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [4, 7] during transform
  warnings.warn(


print(classification_report(y_test, y_pred78))

→      precision    recall   f1-score   support

          0       0.66      0.48      0.55      8710
          1       0.73      0.85      0.79     14513

   accuracy                           0.71      23223
  macro avg       0.69      0.66      0.67      23223
weighted avg       0.70      0.71      0.70      23223

confusion_mat = confusion_matrix(y_test, y_pred78)
print("Точность:", accuracy_score(y_test, y_pred78))
print("Точность:", precision_score(y_test, y_pred78, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred78, average = 'binary'))
print("F1-Мера:", f1_score(y_test, y_pred78, average = 'binary'))
```

```

confusion_mat = confusion_matrix(y_test, y_pred78)
print("Точность:", accuracy_score(y_test, y_pred78))
print("Точность:", precision_score(y_test, y_pred78, average = 'binary'))
print("Полнота:", recall_score(y_test, y_pred78, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred78, average = 'binary'))

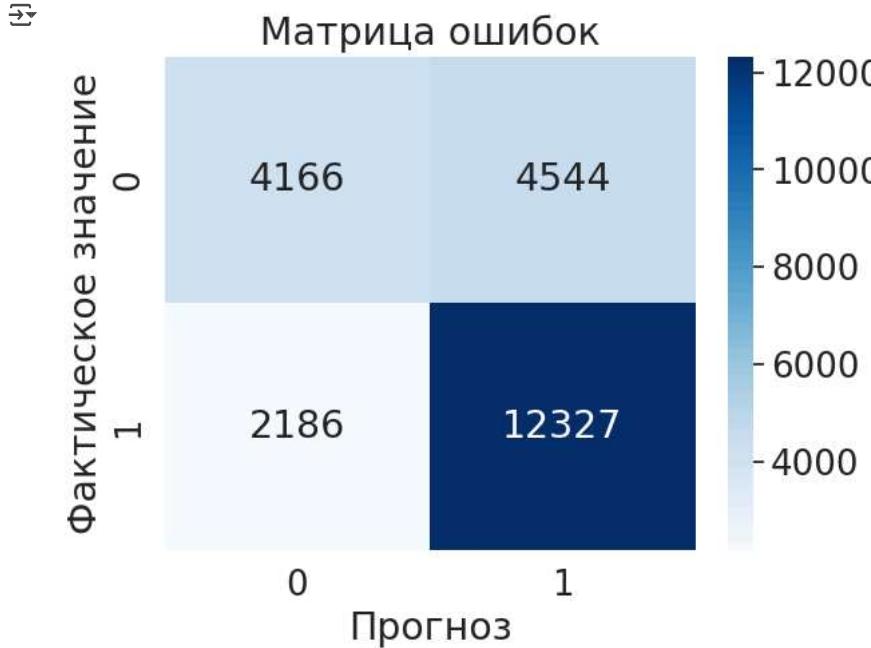
```

→ Точность: 0.7102010937432718  
 Точность: 0.7306620828640863  
 Полнота: 0.8493764211396679  
 F1-мера: 0.7855595207749172

```

sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();

```



**Применим Recursive Feature Elimination (RFE) для выбора количества признаков**

```

from sklearn.feature_selection import RFE

n_features = list(range(2, len(df.columns)-1))
metrics = []

model7 = DecisionTreeClassifier(max_depth=7, min_samples_leaf=4)

for k in n_features:
    rfe = RFE(model7, n_features_to_select=k)
    rfe = rfe.fit(X_train, y_train)
    X_train_rfe = rfe.transform(X_train)
    X_test_rfe = rfe.transform(X_test)
    model7.fit(X_train_rfe, y_train)
    y_pred7 = model7.predict(X_test_rfe)
    f1 = f1_score(y_test, y_pred7, average = 'binary')
    metrics.append(f1)

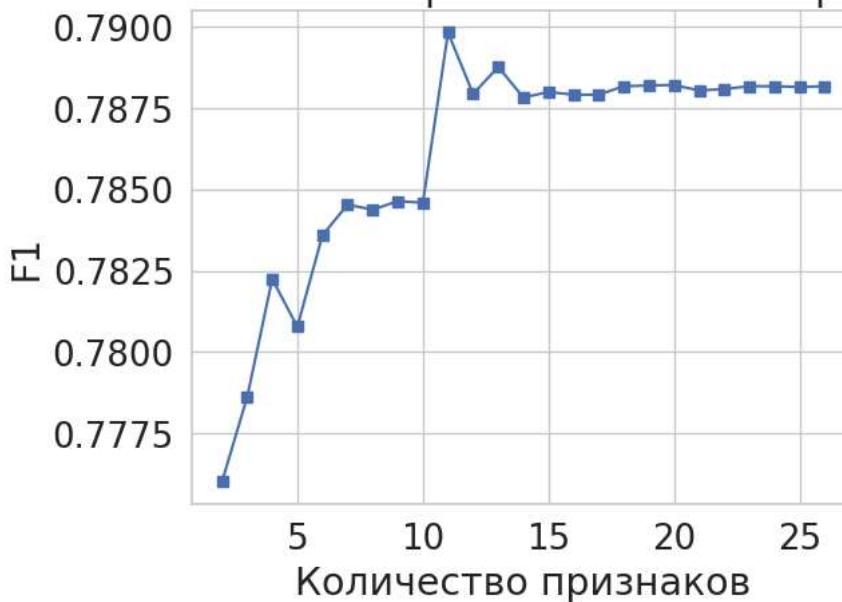
```

```

plt.plot(n_features, metrics, marker = 's')
plt.title('Зависимость F1 меры от количества признаков')
plt.xlabel('Количество признаков')
plt.ylabel('F1');
plt.show()

```

## Зависимость F1 меры от количества признаков



По графику можем увидеть, что оптимальным является 13 признаков. После этого качество модели не улучшается

```
model8 = DecisionTreeClassifier(max_depth=10, min_samples_leaf=4, class_weight='balanced')
rfe = RFE(model8, n_features_to_select=13)
rfe = rfe.fit(X_train, y_train)
```

```
# Показываем как признаки отсортированы по важности для модели по RFE:
selected_features = pd.DataFrame({
    'Feature': X.columns, # Здесь X.columns должен содержать названия столбцов, использованных для X
    'Ranking': rfe.ranking_
})
print(selected_features.sort_values(by='Ranking', ascending=False))
```

	Feature	Ranking
20	POLICY_COURT_SIGN	15
22	POLICY_HAS_COMPLAINTS	14
13	INSURER_GENDER	13
16	POLICY_PRV_CLM_N	12
17	POLICY_PRV_CLM_GLT_N	11
14	POLICY_CLM_N	10
10	VEHICLE_IN_CREDIT	9
3	POLICY_SALES_CHANNEL_GROUP	8
21	CLAIM_AVG_ACC_ST_PRD	7
6	POLICY_MIN_DRIVING_EXPERIENCE	6
0	POLICY_BEGIN_MONTH	5
25	CLIENT_REGISTRATION_REGION	4
19	CLIENT_HAS_OSAGO	3
7	VEHICLE_MAKE	2
8	VEHICLE_MODEL	1
1	POLICY_END_MONTH	1
2	POLICY_SALES_CHANNEL	1
4	POLICY_BRANCH	1
5	POLICY_MIN_AGE	1
9	VEHICLE_ENGINE_POWER	1
15	POLICY_CLM_GLT_N	1
12	POLICY_INTERMEDIARY	1
11	VEHICLE_SUM_INSURED	1
18	CLIENT_HAS_DAGO	1
23	POLICY_YEARS_RENEWED_N	1
24	POLICY_DEDUCT_VALUE	1
26	POLICY_PRICE_CHANGE	1

```
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)
model8.fit(X_train_rfe, y_train)
y_pred8 = model8.predict(X_test_rfe)
print(classification_report(y_test, y_pred8))
```

	precision	recall	f1-score	support
0	0.58	0.56	0.57	8710
1	0.74	0.76	0.75	14513
accuracy			0.68	23223
macro avg	0.66	0.66	0.66	23223
weighted avg	0.68	0.68	0.68	23223

```

confusion_mat = confusion_matrix(y_test, y_pred8)
print("Точность:", accuracy_score(y_test, y_pred8))
print("Полнота:", recall_score(y_test, y_pred8, average = 'binary'))
print("F1-мера:", f1_score(y_test, y_pred8, average = 'binary'))

```

→ Точность: 0.6835034233303191  
 Точность: 0.7407083809395792  
 Полнота: 0.7593881347757183  
 F1-мера: 0.7499319542732716

```

sns.heatmap(confusion_mat, annot=True, fmt="d", cmap = "Blues")
plt.xlabel("Прогноз")
plt.ylabel("Фактическое значение")
plt.title("Матрица ошибок")
plt.show();

```



Видим, что из всех случаев отказа от продления страхового полиса модель правильно предсказывает 55%. Из всех случаев продления страхового полиса модель правильно предсказала 77% случаев.

## Кросс-валидация

Оценим модель с найденными гиперпараметрами в случае однократного применения, при 5-кратной кросс-валидации, при 10-кратной кросс-валидации. За основу возьмем значение F1 score.

```

# Однократная оценка на тестовых данных
single_score = f1_score(y_test, y_pred8, average = 'binary')
print("Однократная оценка эффективности (accuracy):", single_score)

# 5-кратная кросс-валидация на всем датасете
cross_val_scores = cross_val_score(model8, X, y, cv=5, scoring='f1')
print("Среднее значение accuracy по 5-кратной кросс-валидации:", cross_val_scores.mean())

scores_10 = cross_val_score(model8, X, y, cv=10, scoring='f1')
print("Среднее значение accuracy при 10-кратной кросс-валидации:", scores_10.mean())

```

→ Однократная оценка эффективности (accuracy): 0.7499319542732716  
 Среднее значение accuracy по 5-кратной кросс-валидации: 0.7474244881453587  
 Среднее значение accuracy при 10-кратной кросс-валидации: 0.7490432759820888

Можно сделать вывод, что модель не склонна к переобучению - метрика F1 не снижается при 5 либо 10 кратной кросс-валидации

## Выходы

Исследование данных о характеристиках страхователей с целью предсказания возможного продления либо отказа от продления страхового полиса ОСАГО позволило построить модели машинного обучения с хорошими предсказательными способностями. В рамках исследования была проведена работа с категориальными признаками: кодировка осуществлялась исходя из корреляции с целевым признаком; были построены различные модели классификации, определены параметры точности по каждой модели, выбрана оптимальная модель для прогнозирования целевого признака. С помощью кросс-валидации была проверена гипотеза о стабильности модели и отсутствии переобучения, с помощью метода RFE определено оптимальное количество признаков,

необходимых для точного прогнозирования целевого параметра. Результаты исследования могут быть полезны для формулирования управленческих задач специалистов по привлечению клиентов в страховой компании, определения маркетинговой политики, формирования ценовой политики страховой компании.