Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____

КАФЕДРА _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

_____

_____

_____

_____

_____

_____

Студент _____ _____ _____
(Группа) (Подпись, дата) (Фамилия И.О.)

Руководитель курсовой работы _____ _____
(Подпись, дата) (Фамилия И.О.)

Консультант _____ _____
(Подпись, дата) (Фамилия И.О.)

20 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____

Студент группы _____

_____
(Фамилия, имя, отчество)

Тема курсовой работы _____
_____
_____

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_____

Источник тематики (кафедра, предприятие, НИР) _____

График выполнения работы: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание _____
_____
_____
_____
_____

Оформление курсовой работы:

Расчетно-пояснительная записка на _____ листах формата А4.

_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

1. Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках курсового проекта возможно проведение типового или нетипового исследования.

- Типовое исследование – решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
- Нетиповое исследование – решение нестандартной задачи. Тема должна быть согласована с преподавателем. Как правило, такая работа выполняется группой студентов.

2. Основная часть

Импорт библиотек:

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Загрузка и первичный анализ данных

Текстовое описание набора данных:

В качестве набора данных использован набор данных по прогнозированию мошеннических транзакций - IEEE-CIS Fraud Detection (https://www.kaggle.com/c/ieee-fraud-detection/).

Датасет состоит из четырех файлов:

- **train_{transaction, identity}.csv** - обучающая выборка
- **test_{transaction, identity}.csv** - тестовая выборка

Файлы **{train, test}_transaction.csv** и **{train, test}_identity.csv** соединены между собой полем **TransactionID**.

Колонка **isFraud** является целевым признаком, обозначающим, является ли данная транзакция мошеннической. Принимает значения 1 или 0.

Набор данных содержит следующие категориальные признаки:

- **ProductCD**
- **card1 - card6**
- **addr1, addr2**
- **P_emaildomain**
- **R_emaildomain**
- **M1 - M9**
- **DeviceType**
- **DeviceInfo**
- **id_12 - id_38**

Колонка **TransactionDT** содержит разницу во времени между транзакцией и заданной эталонной датой и временем.

Будем решать задачу бинарной классификации целевого признака в колонке **isFraud**.

Загрузка данных:

In [2]:

```
# Будем использовать только обучающую выборку
data = pd.merge(pd.read_csv('E:/train_transaction.csv', sep=","),
                pd.read_csv('E:/train_identity.csv', sep=","),
                on='TransactionID', how='left')
# Удаляем колонки с уникальными ID и временем каждой транзакции, т.к.
# они не понадабятся при построении моделей машинного обучения
data.drop(columns=['TransactionID', 'TransactionDT'], inplace=True)
```

Разведочный анализ данных

Основные характеристики датасета:

In [3]:

```
# Первые 5 строк датасета
data.head()
```

Out[3]:

	isFraud	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	card6	addr1
0	0	68.5	W	13926	NaN	150.0	discover	142.0	credit	315.0
1	0	29.0	W	2755	404.0	150.0	mastercard	102.0	credit	325.0
2	0	59.0	W	4663	490.0	150.0	visa	166.0	debit	330.0
3	0	50.0	W	18132	567.0	150.0	mastercard	117.0	debit	476.0
4	0	50.0	H	4497	514.0	150.0	mastercard	102.0	credit	420.0

5 rows × 432 columns

In [4]:

```
# Размер датасета: (кол-во строк, кол-во колонок)
data.shape
```

Out[4]:

```
(590540, 432)
```

In [5]:

```
# Список колонок с типами данных
data.dtypes
```

Out[5]:

```
isFraud             int64
TransactionAmt    float64
ProductCD          object
card1               int64
card2             float64
                   ...
id_36              object
id_37              object
id_38              object
DeviceType         object
DeviceInfo         object
Length: 432, dtype: object
```

In [6]:

```
# Список колонок с количеством пропущенных значений
data.isnull().sum()
```

Out[6]:

```
isFraud                0
TransactionAmt         0
ProductCD              0
card1                  0
card2               8933
                   ...
id_36             449555
id_37             449555
id_38             449555
DeviceType        449730
DeviceInfo        471874
Length: 432, dtype: int64
```

In [7]:

```
# Общее количество пропущенных значений
data.isnull().sum().sum()
```

Out[7]:

```
115523073
```

Вывод. Данный набор данных содержит пропуски в обучающей выборке, следовательно, нуждается в обработке пропусков в данных.

In [8]:

```
# Основные статистические характеристики набора данных
data.describe()
```

Out[8]:

	isFraud	TransactionAmt	card1	card2	card3	c
count	590540.000000	590540.000000	590540.000000	581607.000000	588975.000000	586281.00
mean	0.034990	135.027176	9898.734658	362.555488	153.194925	199.27
std	0.183755	239.162522	4901.170153	157.793246	11.336444	41.24
min	0.000000	0.251000	1000.000000	100.000000	100.000000	100.00
25%	0.000000	43.321000	6019.000000	214.000000	150.000000	166.00
50%	0.000000	68.769000	9678.000000	361.000000	150.000000	226.00
75%	0.000000	125.000000	14184.000000	512.000000	150.000000	226.00
max	1.000000	31937.391000	18396.000000	600.000000	231.000000	237.00

8 rows × 401 columns

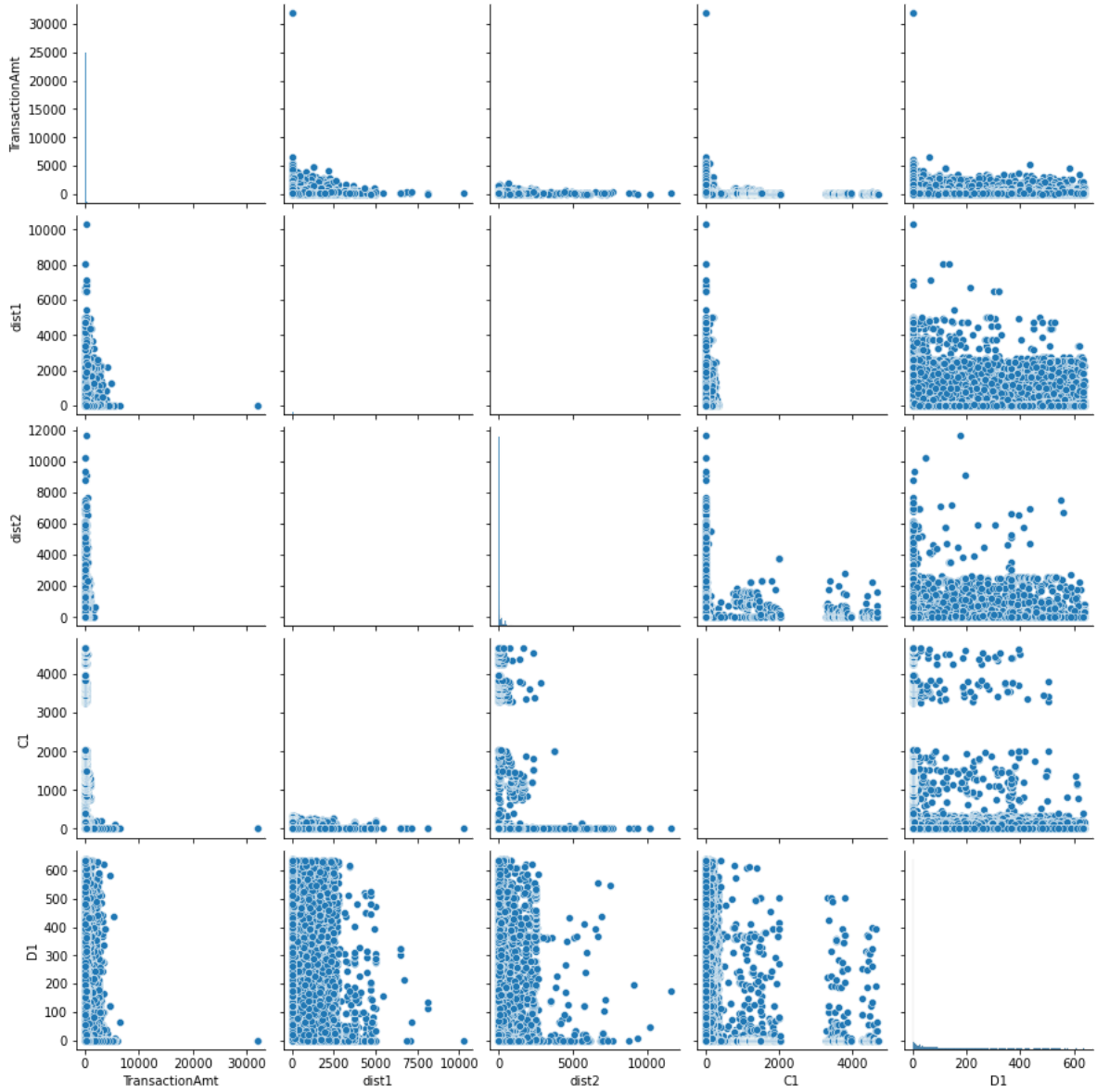
Построение графиков для понимания структуры данных

In [9]:

```
# Парные диаграммы некоторых численных признаков набора данных
sns.pairplot(data[['TransactionAmt', 'dist1', 'dist2', 'C1', 'D1']])
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x14d398f1ee0>
```

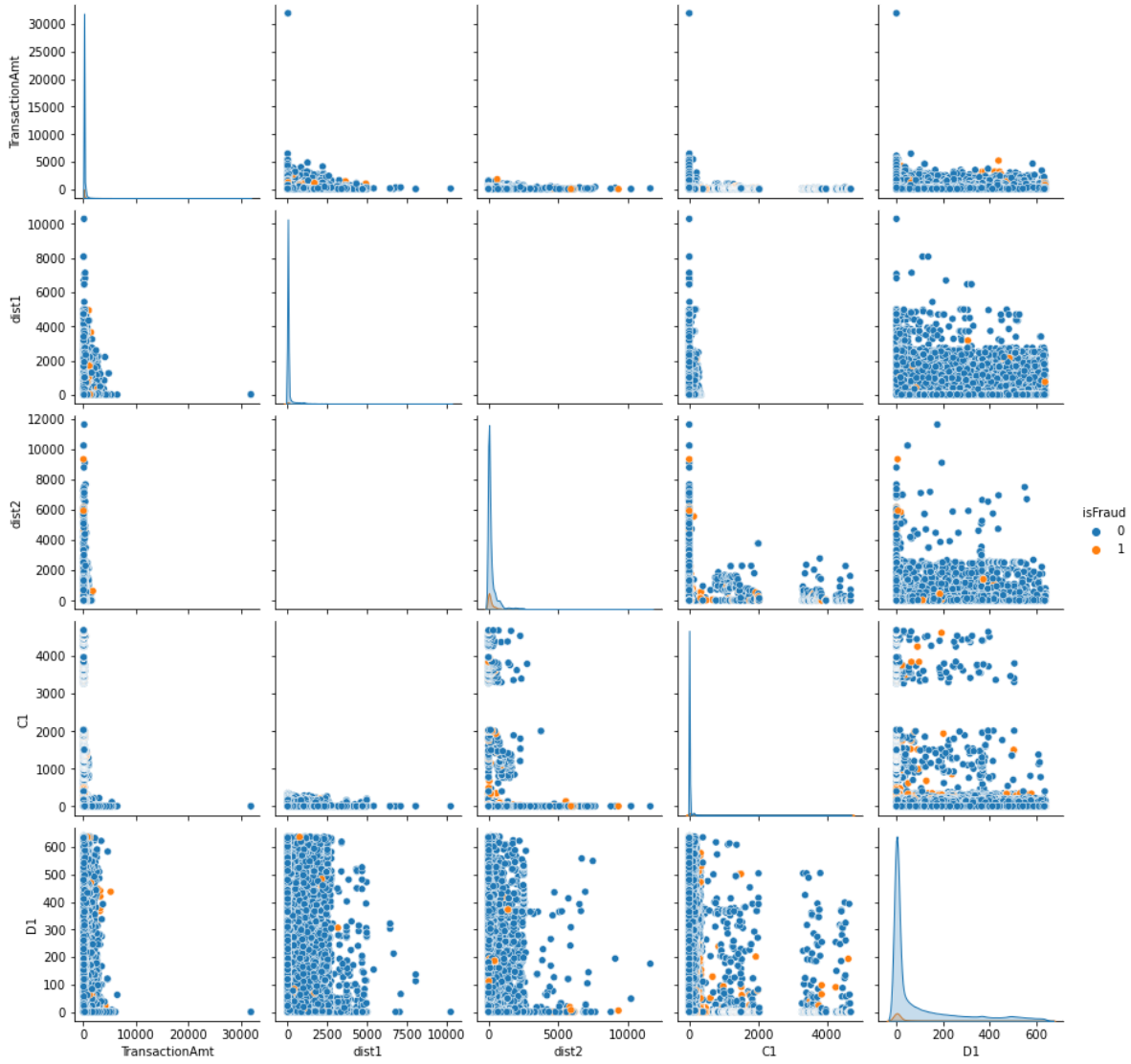


In [10]:

```
sns.pairplot(data[['isFraud', 'TransactionAmt', 'dist1', 'dist2', 'C1', 'D1']],
             hue='isFraud')
```

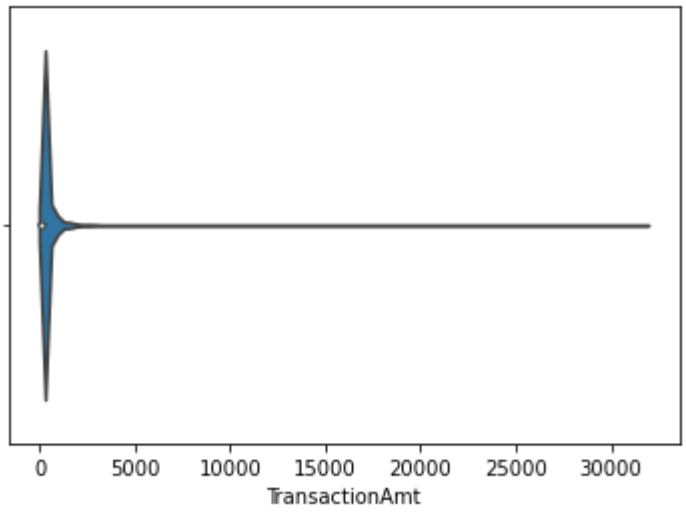
Out[10]:

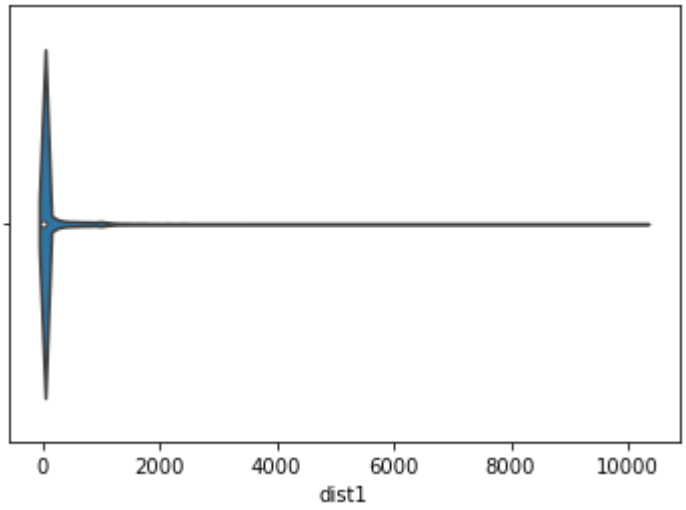
```
<seaborn.axisgrid.PairGrid at 0x14dd23ec610>
```

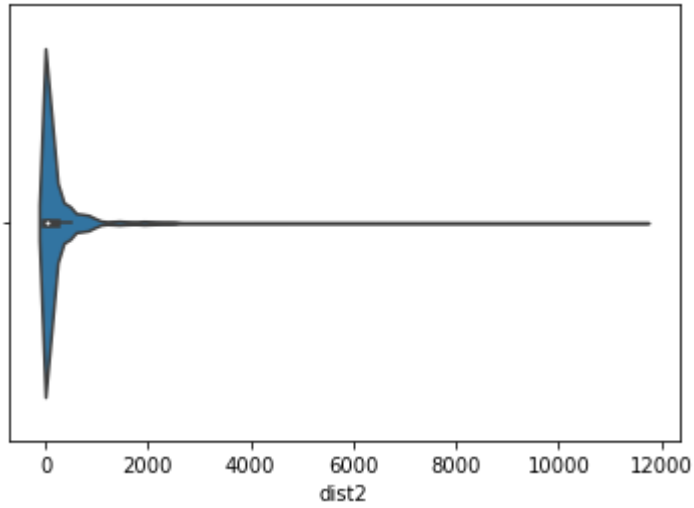


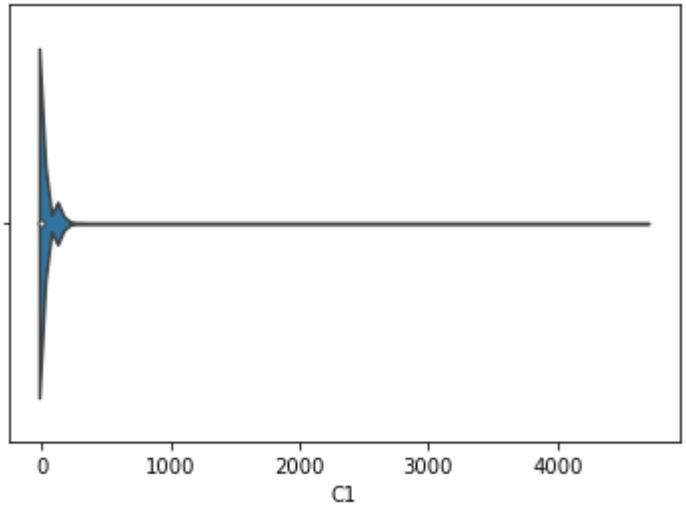
In [11]:

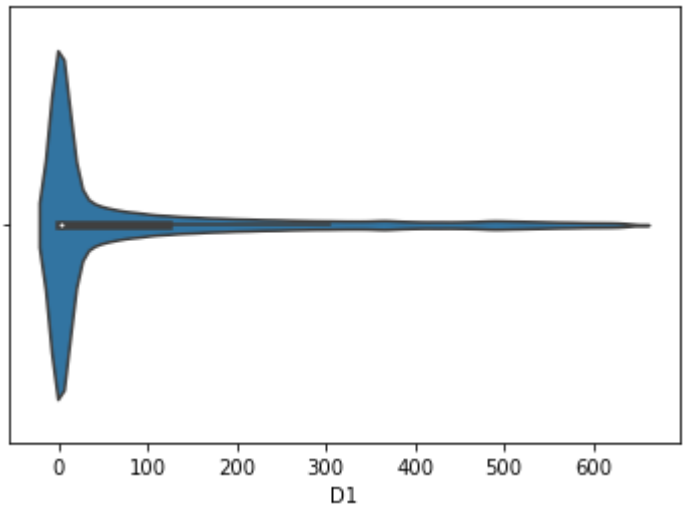
```
# Скрипичные диаграммы для числовых колонок
for col in ['TransactionAmt', 'dist1', 'dist2', 'C1', 'D1']:
    sns.violinplot(x=data[col])
    plt.show()
```






0
2000
4000
6000
8000
10000
12000
dist2


0
1000
2000
3000
4000
C1


0
100
200
300
400
500
600
D1

In [12]:

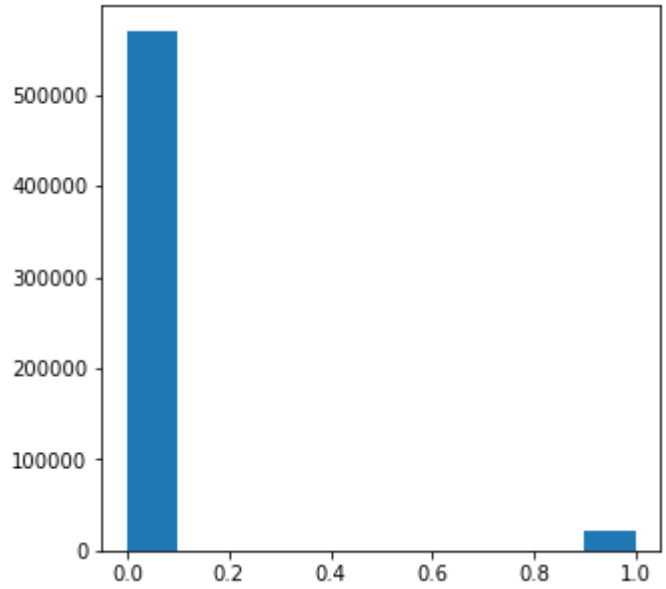
```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['isFraud'].unique()
```

Out[12]:

```
array([0, 1], dtype=int64)
```

In [13]:

```
# Оценим дисбаланс классов для isFraud
fig, ax = plt.subplots(figsize=(5,5))
plt.hist(data['isFraud'])
plt.show()
```



In [14]:

```
# Общее количество машеннических и обычных транзакций в датасете
data['isFraud'].value_counts()
```

Out[14]:

```
0    569877
1     20663
Name: isFraud, dtype: int64
```

In [15]:

```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['isFraud'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round((class_1 / total)*100, 4)))
```

```
Класс 0 составляет 96.5%, а класс 1 составляет 3.499%.
```

Вывод. В данном наборе данных у целевого признака присутствует сильный дисбаланс классов.

Обработка пропусков в данных

Обработка пропусков в числовых данных:

In [16]:

```
# Выберем числовые колонки с пропущенными значениями
total_count = data.shape[0]
num_cols = []
num_cols_null = []
drop_num_cols = []
# Категориальные признаки
cat_cols = ['ProductCD', 'card1', 'card2', 'card3', 'card4', 'card5', 'card6',
            'addr1', 'addr2', 'P_emaildomain', 'R_emaildomain',
            'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9',
            'DeviceType', 'DeviceInfo', 'id_12', 'id_13', 'id_14', 'id_15', 'id_16',
            'id_17', 'id_18', 'id_19', 'id_20', 'id_21', 'id_22', 'id_23', 'id_24',
            'id_25', 'id_26', 'id_27', 'id_28', 'id_29', 'id_30', 'id_31', 'id_32',
            'id_33', 'id_34', 'id_35', 'id_36', 'id_37', 'id_38']
# Цикл по колонкам датасета
for col in data.columns:
    if col not in cat_cols:
        num_cols.append(col)
        # Количество пустых значений
        temp_null_count = data[data[col].isnull()].shape[0]
        dt = str(data[col].dtype)
        if temp_null_count > 0:
            num_cols_null.append(col)
            temp_perc = round((temp_null_count / total_count) * 100.0, 2)
            print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(c
```

```
Колонка dist1. Тип данных float64. Количество пустых значений 352271, 59.6
5%.
Колонка dist2. Тип данных float64. Количество пустых значений 552913, 93.6
3%.
Колонка D1. Тип данных float64. Количество пустых значений 1269, 0.21%.
Колонка D2. Тип данных float64. Количество пустых значений 280797, 47.55%.
Колонка D3. Тип данных float64. Количество пустых значений 262878, 44.51%.
Колонка D4. Тип данных float64. Количество пустых значений 168922, 28.6%.
Колонка D5. Тип данных float64. Количество пустых значений 309841, 52.47%.
Колонка D6. Тип данных float64. Количество пустых значений 517353, 87.61%.
Колонка D7. Тип данных float64. Количество пустых значений 551623, 93.41%.
Колонка D8. Тип данных float64. Количество пустых значений 515614, 87.31%.
Колонка D9. Тип данных float64. Количество пустых значений 515614, 87.31%.
Колонка D10. Тип данных float64. Количество пустых значений 76022, 12.87%.
Колонка D11. Тип данных float64. Количество пустых значений 279287, 47.2
9%.
Колонка D12. Тип данных float64. Количество пустых значений 525823, 89.0
4%.
Колонка D13. Тип данных float64. Количество пустых значений 528588, 89.5
```

In [17]:

```
# Выбор числовых колонок,
# у которых количество пропущенных значений превышает 16%
for col in num_cols_null:
    temp_null_count = data[data[col].isnull()].shape[0]
    temp_perc = round((temp_null_count / total_count) * 100.0, 2)
    if temp_perc >= 16:
        drop_num_cols.append(col)
```

In [18]:

```
# Количество числовых колонок с пропущенными значениями и колонок,
# где процент пропущенных значений превышает 16%
(len(num_cols_null), len(drop_num_cols))
```

Out[18]:

```
(367, 213)
```

In [19]:

```
# Удаление числовых колонок, где процент пропущенных значений превышает 16%
data.drop(columns=drop_num_cols, inplace=True)
# Получившийся размер датасета
data.shape
```

Out[19]:

```
(590540, 219)
```

In [20]:

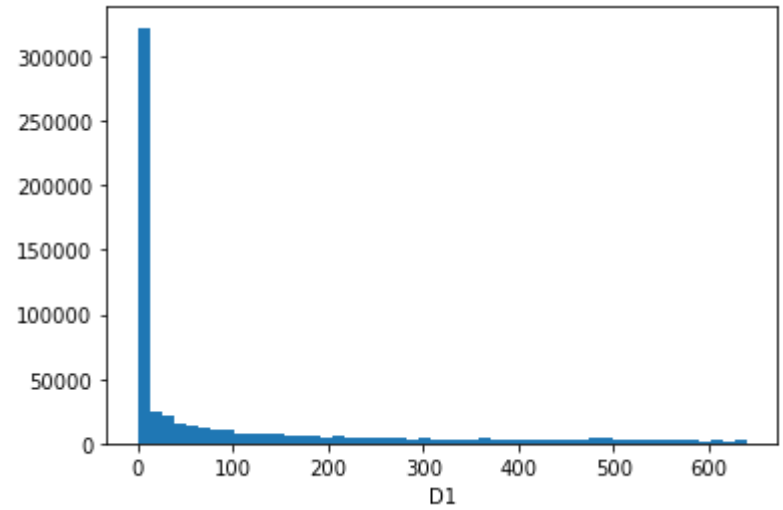
```
for item in drop_num_cols:
    num_cols.remove(item)
    num_cols_null.remove(item)
print('Количество оставшихся необработанных числовых колонок: {}'.format(len(num_cols)))
```

```
Количество оставшихся необработанных числовых колонок: 170
```

Импьютация (внедрение значений) в числовых данных:

In [21]:

```
# Гистограмма по признакам
for col in data[num_cols_null]:
    plt.hist(data[col], 50)
    plt.xlabel(col)
    plt.show()
```



In [22]:

```
from sklearn.impute import SimpleImputer
```

Импьютация стратегией 'медиана':

In [23]:

```
# Импьютация столбцов 'D1', 'D10' и 'D15' стратегией 'медиана'
imp_med = SimpleImputer(missing_values=np.nan, strategy='median')
data.loc[:, 'D1'] = imp_med.fit_transform(data[['D1']])
data.loc[:, 'D10'] = imp_med.fit_transform(data[['D10']])
data.loc[:, 'D15'] = imp_med.fit_transform(data[['D15']])
num_cols_null.remove('D1')
num_cols_null.remove('D10')
num_cols_null.remove('D15')
```

Импьютация стратегией 'мода':

In [24]:

```
# Импьютация остальных столбцов стратегией 'мода'
imp_mod = SimpleImputer(missing_values=np.NaN, strategy='most_frequent')
for item in num_cols_null:
    data.loc[:, item] = imp_mod.fit_transform(data[[item]])
```

Обработка пропусков в категориальных данных:

In [25]:

```
# Выберем категориальные колонки с пропущенными значениями
drop_cat_cols = []
cat_cols_null = []
# Цикл по категориальным колонкам датасета
for col in cat_cols:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count > 0:
        cat_cols_null.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col,
```

```
Колонка card2. Тип данных float64. Количество пустых значений 8933, 1.51%.
Колонка card3. Тип данных float64. Количество пустых значений 1565, 0.27%.
Колонка card4. Тип данных object. Количество пустых значений 1577, 0.27%.
Колонка card5. Тип данных float64. Количество пустых значений 4259, 0.72%.
Колонка card6. Тип данных object. Количество пустых значений 1571, 0.27%.
Колонка addr1. Тип данных float64. Количество пустых значений 65706, 11.13%.
Колонка addr2. Тип данных float64. Количество пустых значений 65706, 11.13%.
Колонка P_emaildomain. Тип данных object. Количество пустых значений 94456,
15.99%.
Колонка R_emaildomain. Тип данных object. Количество пустых значений 453249,
76.75%.
Колонка M1. Тип данных object. Количество пустых значений 271100, 45.91%.
Колонка M2. Тип данных object. Количество пустых значений 271100, 45.91%.
Колонка M3. Тип данных object. Количество пустых значений 271100, 45.91%.
Колонка M4. Тип данных object. Количество пустых значений 281444, 47.66%.
Колонка M5. Тип данных object. Количество пустых значений 350482, 59.35%.
Колонка M6. Тип данных object. Количество пустых значений 169360, 28.68%.
Колонка M7. Тип данных object. Количество пустых значений 346265, 58.64%.
Колонка M8. Тип данных object. Количество пустых значений 346252, 58.63%.
Колонка M9. Тип данных object. Количество пустых значений 346252, 58.63%.
Колонка DeviceType. Тип данных object. Количество пустых значений 449730, 7
6.16%.
Колонка DeviceInfo. Тип данных object. Количество пустых значений 471874, 7
9.91%.
Колонка id_12. Тип данных object. Количество пустых значений 446307, 75.58%.
Колонка id_13. Тип данных float64. Количество пустых значений 463220, 78.4
4%.
Колонка id_14. Тип данных float64. Количество пустых значений 510496, 86.4
5%.
Колонка id_15. Тип данных object. Количество пустых значений 449555, 76.13%.
Колонка id_16. Тип данных object. Количество пустых значений 461200, 78.1%.
Колонка id_17. Тип данных float64. Количество пустых значений 451171, 76.4%.
Колонка id_18. Тип данных float64. Количество пустых значений 545427, 92.3
6%.
Колонка id_19. Тип данных float64. Количество пустых значений 451222, 76.4
1%.
Колонка id_20. Тип данных float64. Количество пустых значений 451279, 76.4
2%.
Колонка id_21. Тип данных float64. Количество пустых значений 585381, 99.1
3%.
Колонка id_22. Тип данных float64. Количество пустых значений 585371, 99.1
2%.
Колонка id_23. Тип данных object. Количество пустых значений 585371, 99.12%.
Колонка id_24. Тип данных float64. Количество пустых значений 585793, 99.2%.
```

```
Колонка id_25. Тип данных float64. Количество пустых значений 585408, 99.1
3%.
Колонка id_26. Тип данных float64. Количество пустых значений 585377, 99.1
3%.
Колонка id_27. Тип данных object. Количество пустых значений 585371, 99.12%.
Колонка id_28. Тип данных object. Количество пустых значений 449562, 76.13%.
Колонка id_29. Тип данных object. Количество пустых значений 449562, 76.13%.
Колонка id_30. Тип данных object. Количество пустых значений 512975, 86.87%.
Колонка id_31. Тип данных object. Количество пустых значений 450258, 76.25%.
Колонка id_32. Тип данных float64. Количество пустых значений 512954, 86.8
6%.
Колонка id_33. Тип данных object. Количество пустых значений 517251, 87.59%.
Колонка id_34. Тип данных object. Количество пустых значений 512735, 86.82%.
Колонка id_35. Тип данных object. Количество пустых значений 449555, 76.13%.
Колонка id_36. Тип данных object. Количество пустых значений 449555, 76.13%.
Колонка id_37. Тип данных object. Количество пустых значений 449555, 76.13%.
Колонка id_38. Тип данных object. Количество пустых значений 449555, 76.13%.
```

In [26]:

```
# Выбор категориальных колонок,
# у которых количество пропущенных значений превышает 16%
for col in cat_cols_null:
    temp_null_count = data[data[col].isnull()].shape[0]
    temp_perc = round((temp_null_count / total_count) * 100.0, 2)
    if temp_perc >= 16:
        drop_cat_cols.append(col)
```

In [27]:

```
# Количество категориальных колонок с пропущенными значениями и колонок,
# где процент пропущенных значений превышает 16%
(len(cat_cols_null), len(drop_cat_cols))
```

Out[27]:

```
(47, 39)
```

In [28]:

```
# Удаление категориальных колонок,
# где процент пропущенных значений превышает 16%
data.drop(columns=drop_cat_cols, inplace=True)
# Получившийся размер датасета
data.shape
```

Out[28]:

```
(590540, 180)
```

In [29]:

```
for item in drop_cat_cols:
    cat_cols.remove(item)
    cat_cols_null.remove(item)
print('Количество оставшихся необработанных категориальных колонок: {}'.format(len(cat_cols
```

```
Количество оставшихся необработанных категориальных колонок: 10
```

Кодирование категориальных признаков

In [30]:

```
# Выберем категориальные колонки
# Цикл по колонкам датасета
for col in cat_cols:
    dt = str(data[col].dtype)
    temp_un = data[col].nunique()
    print('Колонка {}. Тип данных {}. Количество уникальных значений {}.'.format(col, dt, t
```

```
Колонка ProductCD. Тип данных object. Количество уникальных значений 5.
Колонка card1. Тип данных int64. Количество уникальных значений 13553.
Колонка card2. Тип данных float64. Количество уникальных значений 500.
Колонка card3. Тип данных float64. Количество уникальных значений 114.
Колонка card4. Тип данных object. Количество уникальных значений 4.
Колонка card5. Тип данных float64. Количество уникальных значений 119.
Колонка card6. Тип данных object. Количество уникальных значений 4.
Колонка addr1. Тип данных float64. Количество уникальных значений 332.
Колонка addr2. Тип данных float64. Количество уникальных значений 74.
Колонка P_emaildomain. Тип данных object. Количество уникальных значений 59.
```

Кодирование категорий наборами бинарных значений (one-hot encoding):

In [31]:

```
# Выберем категориальные колонки с малым количеством уникальных значений,
# чтобы сильно не расширять признаковое пространство набора данных
col_oh = ['ProductCD', 'card4', 'card6', 'P_emaildomain']
one_hot = pd.get_dummies(data[col_oh].astype(str))
one_hot.head()
```

Out[31]:

	ProductCD_C	ProductCD_H	ProductCD_R	ProductCD_S	ProductCD_W	card4_american express	ca
0	0	0	0	0	1	0	
1	0	0	0	0	1	0	
2	0	0	0	0	1	0	
3	0	0	0	0	1	0	
4	0	1	0	0	0	0	

5 rows × 75 columns

In [32]:

```
# Замена исходных категориальных колонок наборами бинарных значений
data = data.join(one_hot)
data.drop(columns=col_oh, inplace=True)
for item in col_oh:
    cat_cols.remove(item)
```

In [33]:

```
# Первые 5 строк получившегося датасета
data.head()
```

Out[33]:

	isFraud	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C1	C2	...	P_emaildo
0	0	68.5	13926	NaN	150.0	142.0	315.0	87.0	1.0	1.0	...	
1	0	29.0	2755	404.0	150.0	102.0	325.0	87.0	1.0	1.0	...	
2	0	59.0	4663	490.0	150.0	166.0	330.0	87.0	1.0	1.0	...	
3	0	50.0	18132	567.0	150.0	117.0	476.0	87.0	2.0	5.0	...	
4	0	50.0	4497	514.0	150.0	102.0	420.0	87.0	1.0	1.0	...	

5 rows × 251 columns

Кодирование категорий целочисленными значениями (label encoding):

In [34]:

```
from sklearn.preprocessing import LabelEncoder
```

In [35]:

```
# Label encoding оставшихся категориальных колонок
le = LabelEncoder()
for item in cat_cols:
    data.loc[:, item] = le.fit_transform(data[item])
# Первые 5 строк категориальных колонок после label encoding
data[cat_cols].head()
```

Out[35]:

	card1	card2	card3	card5	addr1	addr2
0	10095	9432	42	38	166	62
1	1372	303	42	2	173	62
2	2833	389	42	58	178	62
3	13341	466	42	14	282	62
4	2712	413	42	2	241	62

In [36]:

```
# Проверка, что в датасете не осталось пропущенных значений
print('Количество пропущенных значений: {}'.format(data.isnull().sum().sum()))
```

```
Количество пропущенных значений: 0
```

Масштабирование данных

MinMax масштабирование

In [37]:

```
from sklearn.preprocessing import MinMaxScaler
```

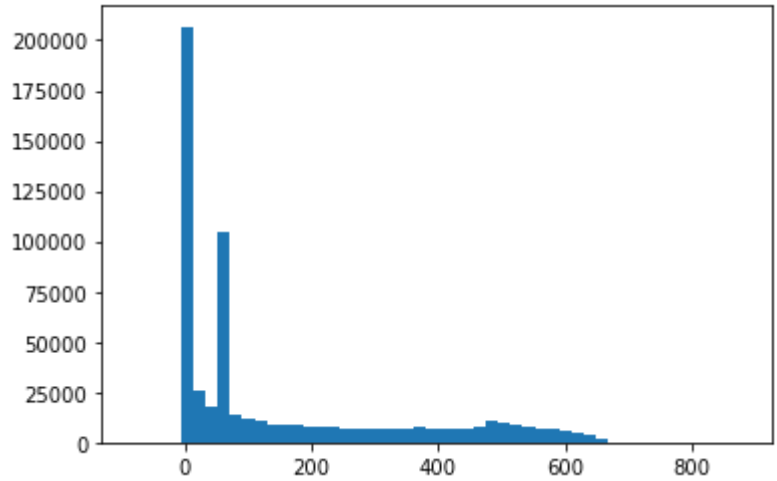
In [38]:

```
# Выберем колонки с количеством уникальных значений больше 2
cols = []
for col in num_cols:
    temp_un = data[col].nunique()
    if temp_un > 2:
        cols.append(col)
    print('Колонка {}. Количество уникальных значений {}.'.format(col, temp_un))
```

```
Колонка isFraud. Количество уникальных значений 2.
Колонка TransactionAmt. Количество уникальных значений 20902.
Колонка C1. Количество уникальных значений 1657.
Колонка C2. Количество уникальных значений 1216.
Колонка C3. Количество уникальных значений 27.
Колонка C4. Количество уникальных значений 1260.
Колонка C5. Количество уникальных значений 319.
Колонка C6. Количество уникальных значений 1328.
Колонка C7. Количество уникальных значений 1103.
Колонка C8. Количество уникальных значений 1253.
Колонка C9. Количество уникальных значений 205.
Колонка C10. Количество уникальных значений 1231.
Колонка C11. Количество уникальных значений 1476.
Колонка C12. Количество уникальных значений 1199.
Колонка C13. Количество уникальных значений 1597.
Колонка C14. Количество уникальных значений 1108.
Колонка D1. Количество уникальных значений 641.
Колонка D10. Количество уникальных значений 818.
Колонка D15. Количество уникальных значений 859.
```

In [39]:

```
# Гистограмма распределения какого-нибудь числового признака
plt.hist(data['D15'], 50)
plt.show()
```

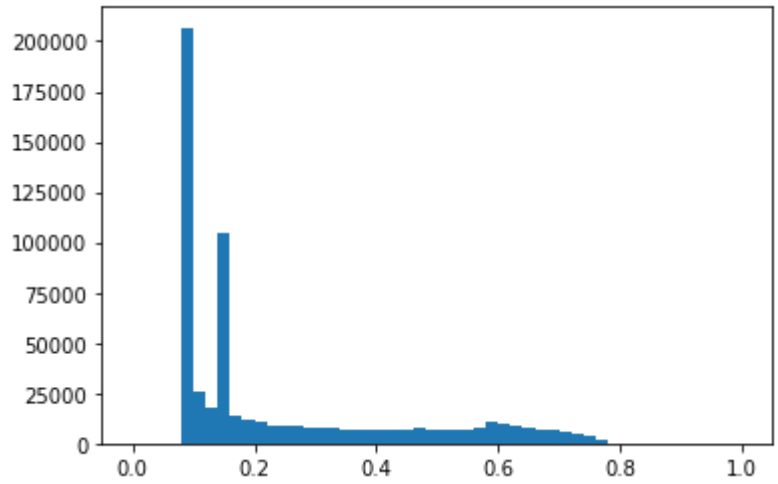


In [40]:

```
# MinMax масштабирование
mms = MinMaxScaler()
for item in cols:
    data.loc[:, item] = mms.fit_transform(data[[item]])
```

In [41]:

```
# Гистограмма распределения этого признака после масштабирования
plt.hist(data['D15'], 50)
plt.show()
```



Вывод. MinMax масштабирование изменило диапазон измерения величины на 0...1 и не повлияло на распределение этой величины.

Корреляционный анализ данных

Проверка корреляции признаков позволяет решить две задачи:

- Понять какие признаки (колонки датасета) наиболее сильно коррелируют с целевым признаком. Наличие сильной корреляции с целевым признаком говорит о линейной зависимости этих признаков и сильно облегчает задау построения моделей машинного обучения.
- Понять какие нецелевые признаки линейно зависимы между собой. Линейно зависимые признаки, как правило, очень плохо влияют на качество моделей. Поэтому если несколько признаков линейно зависимы, то для построения модели из них выбирают какой-то один признак.

In [42]:

```
# Вывод наиболее коррелирующих признаков с целевым признаком
corrFr = data[data.columns.drop(cat_cols)].corrwith(data['isFraud']).sort_values(ascending=
corrFr
```

Out[42]:

```
isFraud        1.000000
V86            0.222343
V87            0.221568
V79            0.167299
V94            0.161454
                 ...
card6_debit   -0.099779
V69           -0.102396
V90           -0.102635
V29           -0.102738
ProductCD_W   -0.135549
Length: 245, dtype: float64
```

Вывод. Данный набор данных не содержит простых зависимостей с целевым признаком.

In [43]:

```
# Анализ сильно коррелированных признаков и удаление одного из них из датасета
drop_col = []
for col in corrFr.index:
    if col not in drop_col:
        tmp = data[data.columns.drop(cat_cols)].corrwith(data[col])
        flg = True
        for i in range(1, tmp.shape[0]):
            if col is not tmp.index[i]:
                if abs(tmp[i]) > 0.98:
                    if flg:
                        print('Признак {} имеет корреляцию с целевым признаком: {}.'.format
                        flg = False
                    print('Признак {} имеет корреляцию с признаком {}: {}.'.format(col, tmp
                    print('Признак {} имеет корреляцию с целевым признаком: {}.'.format(tmp
                    if abs(corrFr[col]) < abs(corrFr[tmp.index[i]]):
                        print('Удаляем признак {}, т.к. он меньше коллерирует с целевым при
                        data.drop(columns=col, inplace=True)
                        drop_col.append(col)
                        break;
                    else:
                        print('Удаляем признак {}, т.к. он меньше коллерирует с целевым при
                        data.drop(columns=tmp.index[i], inplace=True)
                        drop_col.append(tmp.index[i])
        if not flg:
            print()
```

```
Признак V17 имеет корреляцию с целевым признаком: 0.15897168090753624.
Признак V17 имеет корреляцию с признаком V18: 0.9916176966010278.
Признак V18 имеет корреляцию с целевым признаком: 0.1589329174050446.
Удаляем признак V18, т.к. он меньше коллерирует с целевым признаком.

Признак V15 имеет корреляцию с целевым признаком: 0.15475457773791504.
Признак V15 имеет корреляцию с признаком V16: 0.986945558622141.
Признак V16 имеет корреляцию с целевым признаком: 0.15164513658016676.
Удаляем признак V16, т.к. он меньше коллерирует с целевым признаком.

Признак V31 имеет корреляцию с целевым признаком: 0.1410017948405061.
Признак V31 имеет корреляцию с признаком V32: 0.9864628749581449.
Признак V32 имеет корреляцию с целевым признаком: 0.13851735945669646.
Удаляем признак V32, т.к. он меньше коллерирует с целевым признаком.

Признак card6_credit имеет корреляцию с целевым признаком: 0.1005075851937
8097.
Признак card6_credit имеет корреляцию с признаком card6_debit: -0.99279867
0400973.
```

In [44]:

```
# Окончательный датасет
data.head()
```

Out[44]:

	isFraud	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C2	C3	...	P_en
0	0	0.002137	10095	9432	42	38	166	62	0.000176	0.0	...	
1	0	0.000900	1372	303	42	2	173	62	0.000176	0.0	...	
2	0	0.001840	2833	389	42	58	178	62	0.000176	0.0	...	
3	0	0.001558	13341	466	42	14	282	62	0.000879	0.0	...	
4	0	0.001558	2712	413	42	2	241	62	0.000176	0.0	...	

5 rows × 220 columns

In [45]:

```
# Окончательный размер датасета
data.shape
```

Out[45]:

```
(590540, 220)
```

In [46]:

```
# Сохранение получившегося датасета
data.to_csv('E:/train.csv')
```

Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи классификации будем использовать:

Метрика Balanced Accuracy:

Метрика используется для работы с несбалансированными наборами данных (как данный датасет). Она определяет средний процент (долю в диапазоне от 0 до 1) правильно определенных классов по каждому классу.

Используется функция balanced_accuracy_score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html#sklearn.metrics.balanced_ac

In [47]:

```
from sklearn.metrics import balanced_accuracy_score
```

Метрика recall (полнота):

$recall = \frac{TP}{TP+FN}$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов. Эта метрика в данной предметной области позволяет оценить количество мошеннических транзакций, которые были ошибочно приняты за обычные, что является очень важной оценкой.

Используется функция recall_score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score).

In [48]:

```
from sklearn.metrics import recall_score
```

Метрика ROC AUC:

Используется для оценки качества бинарной классификации. Данная метрика используется для оценки качества модели на сайте соревнования с данным датасетом.

Основана на вычислении следующих характеристик:

$TPR = \frac{TP}{TP+FN}$ - True Positive Rate, откладывается по оси ординат. Совпадает с recall.

$FPR = \frac{FP}{FP+TN}$ - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

Идеальная ROC-кривая проходит через точки (0,0)-(0,1)-(1,1), то есть через верхний левый угол графика.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая, тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция roc_auc_score. (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score)

In [49]:

```
from sklearn.metrics import roc_auc_score
```

Выбор моделей для решения задачи классификации

Для задачи классификации будем использовать следующие модели:

- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

Модели "Случайный лес" и "Градиентный бустинг" являются ансамблевыми моделями.

Формирование обучающей и тестовой выборок на основе исходного набора данных

Используем метод train_test_split:

In [50]:

```
from sklearn.model_selection import train_test_split
```

In [51]:

```
data_train, data_test, data_y_train, data_y_test = train_test_split(data[data.columns.drop(
# Размер обучающей и тестовой выборок
(data_train.shape, data_test.shape, data_y_train.shape, data_y_test.shape)
```

Out[51]:

```
((442905, 219), (147635, 219), (442905,), (147635,))
```

Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров

Модель "Метод ближайших соседей":

In [52]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [53]:

```
# Словари для сохранения значений метрик разных моделей
baseline_predicted_rocauc = {}
baseline_predicted_recall = {}
baseline_predicted_bAccur = {}
```

In [54]:

```
KNC = KNeighborsClassifier(n_jobs=-1).fit(data_train, data_y_train)
predict = KNC.predict(data_test)
predict_p = KNC.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
baseline_predicted_rocauc["KN"] = roc_auc_score(data_y_test, predict_p[:,1])
baseline_predicted_recall["KN"] = recall_score(data_y_test, predict)
baseline_predicted_bAccur["KN"] = balanced_accuracy_score(data_y_test, predict)
```

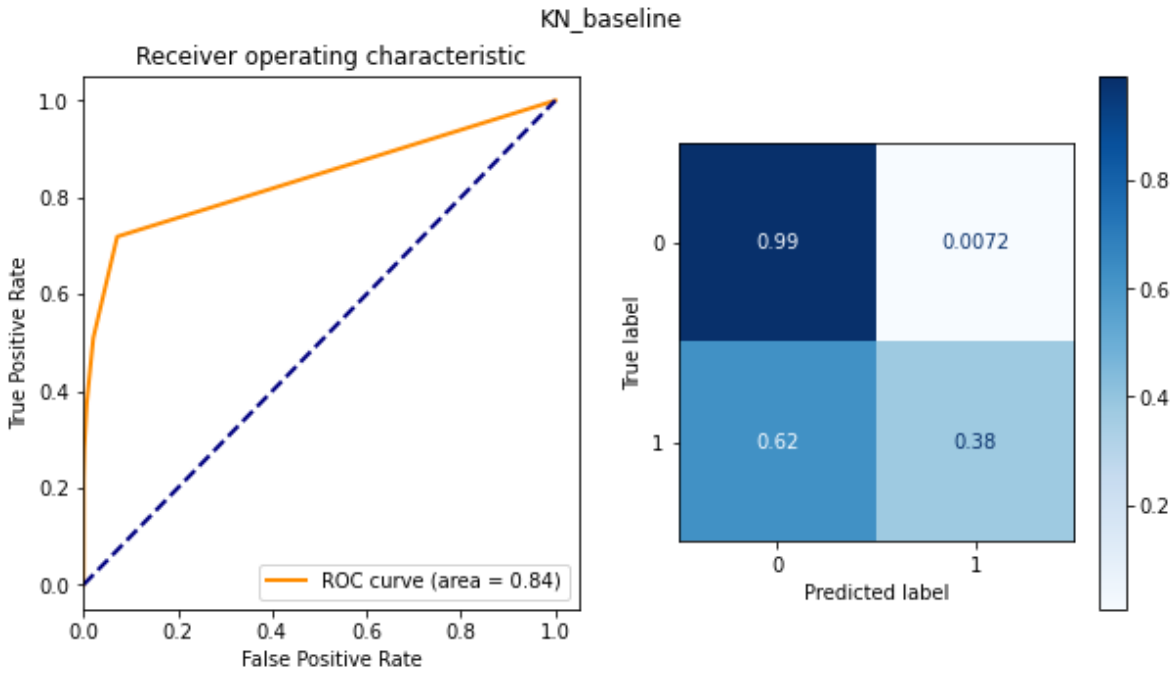
In [55]:

```
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import roc_curve

# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    lw = 2
    ax.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_xlim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
    ax.legend(loc="lower right")

# Отрисовка ROC-кривой с матрицей ошибок
def draw_m(model, modelName, data_test, data_y_test, predict_p):
    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(data_y_test, predict_p, ax[0])
    plot_confusion_matrix(model, data_test, data_y_test, ax=ax[1],
                      display_labels=['0','1'],
                      cmap=plt.cm.Blues, normalize='true')
    fig.suptitle(modelName)
    plt.show()
```

In [56]:

```
print('Метод ближайших соседей:')
print('ROC AUC: ', baseline_predicted_rocauc["KN"])
print('Recall: ', baseline_predicted_recall["KN"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["KN"])
draw_m(KNC, 'KN_baseline', data_test, data_y_test, predict_p[:,1])
```

```
Метод ближайших соседей:
ROC AUC:  0.8372066763690009
Recall:  0.3754379135850526
Balanced Accuracy:  0.6841223898472573
```



Модель "Машина опорных векторов":

In [57]:

```
from sklearn.svm import SVC
```

In [58]:

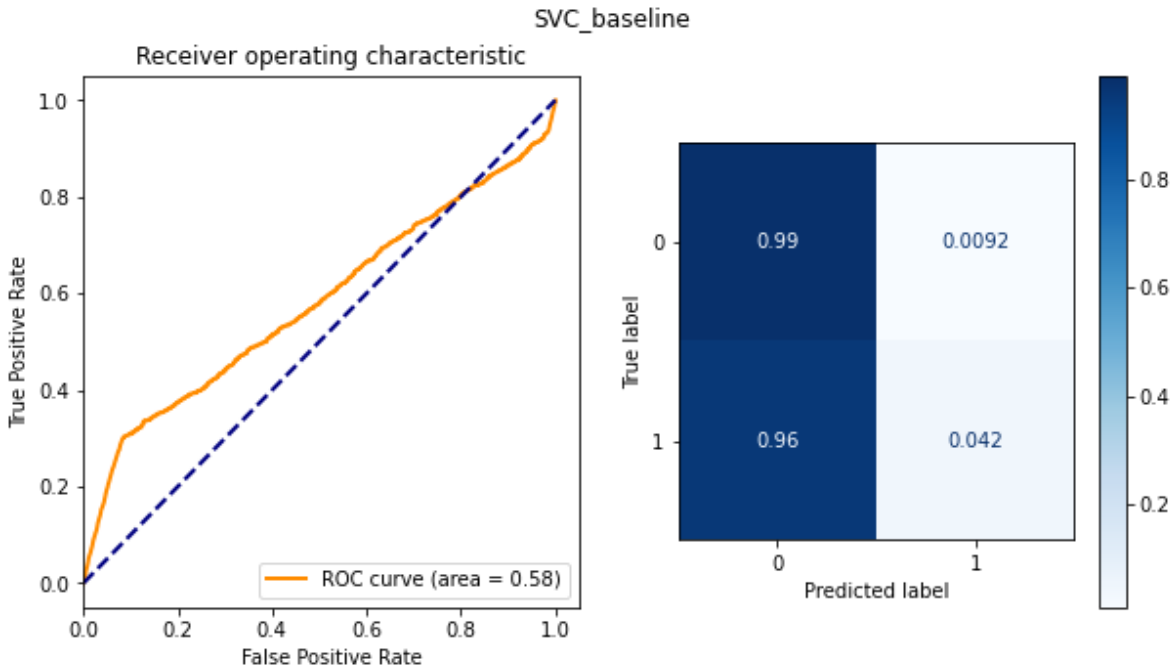
```
svc = SVC(kernel='rbf', random_state=1, probability=True, cache_size=2000, max_iter=100).fi
predict = svc.predict(data_test)
predict_p = svc.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
baseline_predicted_rocauc["SVC"] = roc_auc_score(data_y_test, predict_p[:,1])
baseline_predicted_recall["SVC"] = recall_score(data_y_test, predict)
baseline_predicted_bAccur["SVC"] = balanced_accuracy_score(data_y_test, predict)
```

```
D:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:246: Converg
enceWarning: Solver terminated early (max_iter=100).  Consider pre-processin
g your data with StandardScaler or MinMaxScaler.
  warnings.warn('Solver terminated early (max_iter=%i).'
```

In [59]:

```
print('Машина опорных векторов:')
print('ROC AUC: ', baseline_predicted_rocauc["SVC"])
print('Recall: ', baseline_predicted_recall["SVC"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["SVC"])
draw_m(svc, 'SVC_baseline', data_test, data_y_test, predict_p[:,1])
```

```
Машина опорных векторов:
ROC AUC:  0.5791999785409971
Recall:  0.041650447644998055
Balanced Accuracy:  0.5162216181325547
```



Модель "Решающее дерево":

In [60]:

```
from sklearn.tree import DecisionTreeClassifier
```

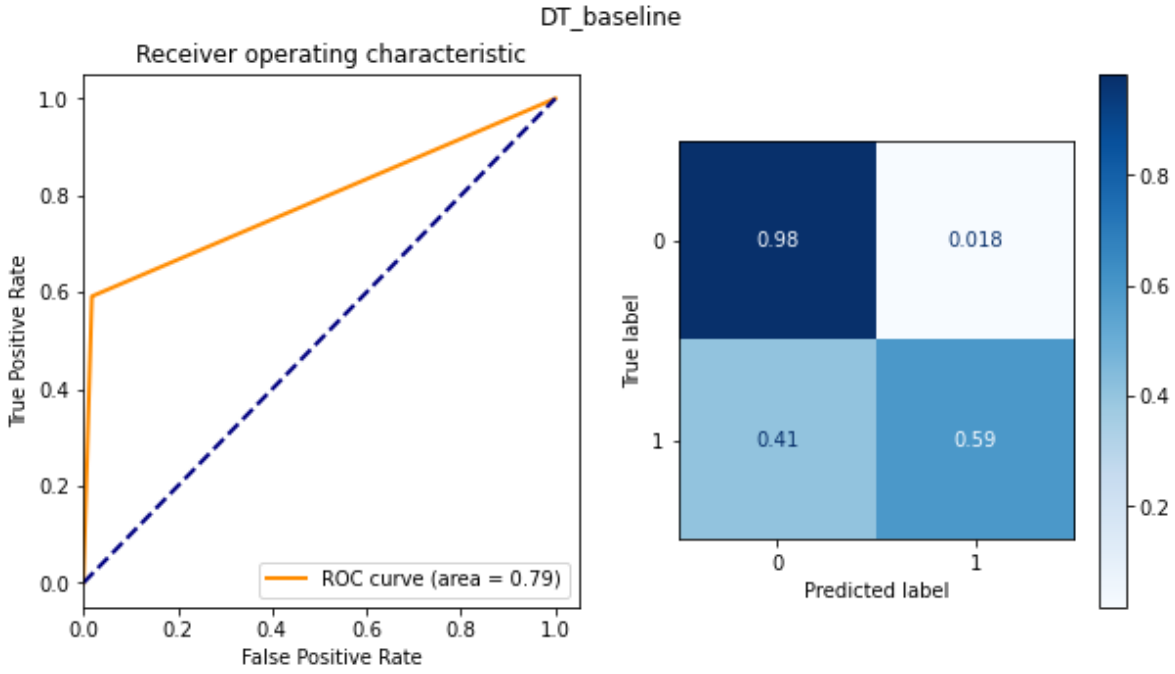
In [61]:

```
dtc = DecisionTreeClassifier(random_state=1).fit(data_train, data_y_train)
predict = dtc.predict(data_test)
predict_p = dtc.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
baseline_predicted_rocauc["DT"] = roc_auc_score(data_y_test, predict_p[:,1])
baseline_predicted_recall["DT"] = recall_score(data_y_test, predict)
baseline_predicted_bAccur["DT"] = balanced_accuracy_score(data_y_test, predict)
```

In [62]:

```
print('Решающее дерево:')
print('ROC AUC: ', baseline_predicted_rocauc["DT"])
print('Recall: ', baseline_predicted_recall["DT"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["DT"])
draw_m(dtc, 'DT_baseline', data_test, data_y_test, predict_p[:,1])
```

```
Решающее дерево:
ROC AUC:  0.786755931457974
Recall:  0.591475282210977
Balanced Accuracy:  0.7866988543240123
```



Модель "Случайный лес":

In [63]:

```
from sklearn.ensemble import RandomForestClassifier
```

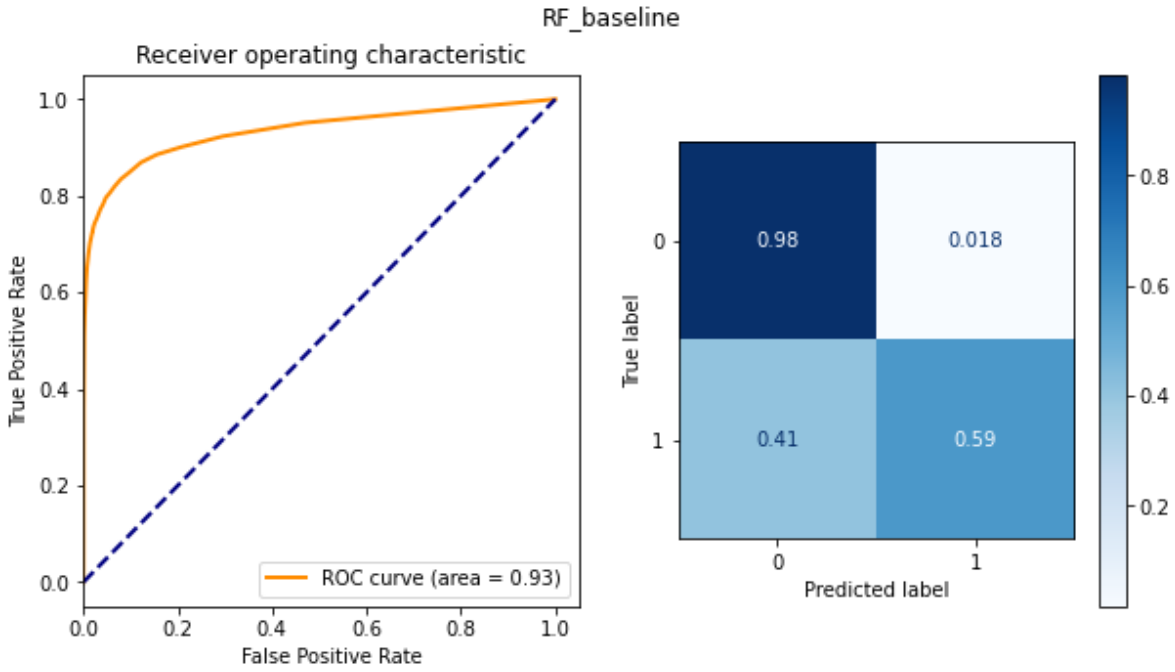
In [64]:

```
RFC = RandomForestClassifier(random_state=1, n_jobs=-1).fit(data_train, data_y_train)
predict = RFC.predict(data_test)
predict_p = RFC.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
baseline_predicted_rocauc["RF"] = roc_auc_score(data_y_test, predict_p[:,1])
baseline_predicted_recall["RF"] = recall_score(data_y_test, predict)
baseline_predicted_bAccur["RF"] = balanced_accuracy_score(data_y_test, predict)
```

In [65]:

```
print('Случайный лес:')
print('ROC AUC: ', baseline_predicted_rocauc["RF"])
print('Recall: ', baseline_predicted_recall["RF"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["RF"])
draw_m(dtc, 'RF_baseline', data_test, data_y_test, predict_p[:,1])
```

```
Случайный лес:
ROC AUC:  0.9328265098547771
Recall:  0.48871156091864537
Balanced Accuracy:  0.7435662901542637
```



Модель "Градиентный бустинг":

In [66]:

```
from lightgbm import LGBMClassifier
```

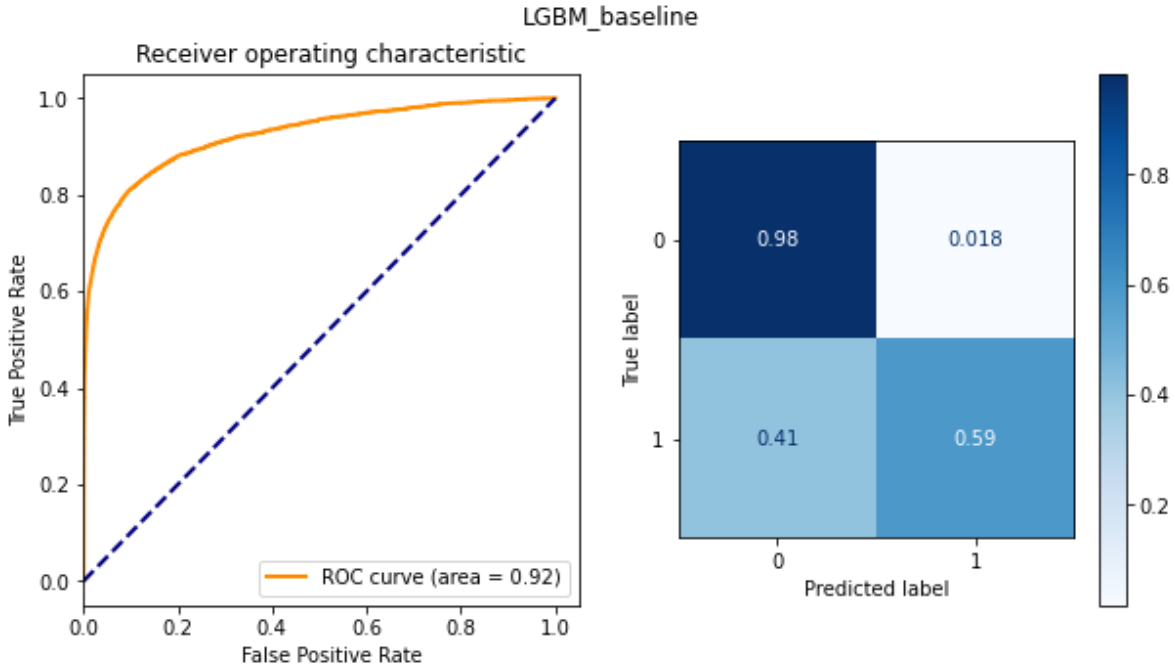
In [67]:

```
LGBMC = LGBMClassifier(random_state=1).fit(data_train, data_y_train)
predict = LGBMC.predict(data_test)
predict_p = LGBMC.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
baseline_predicted_rocauc["LGBM"] = roc_auc_score(data_y_test, predict_p[:,1])
baseline_predicted_recall["LGBM"] = recall_score(data_y_test, predict)
baseline_predicted_bAccur["LGBM"] = balanced_accuracy_score(data_y_test, predict)
```

In [68]:

```
print('Градиентный бустинг:')
print('ROC AUC: ', baseline_predicted_rocauc["LGBM"])
print('Recall: ', baseline_predicted_recall["LGBM"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["LGBM"])
draw_m(dtc, 'LGBM_baseline', data_test, data_y_test, predict_p[:,1])
```

```
Градиентный бустинг:
ROC AUC:  0.9238636891068323
Recall:  0.40054495912806537
Balanced Accuracy:  0.6992303523613548
```



Подбор гиперпараметров для выбранных моделей

Модель "Метод ближайших соседей":

In [69]:

```
from sklearn.model_selection import GridSearchCV
```

In [70]:

```
# Гиперпараметры
n_range = np.array(range(0,11,5))
n_range[0] = 1
tuned_parameters = [{'n_neighbors': n_range}]
```

In [71]:

```
%%time
GSCV = GridSearchCV(KNeighborsClassifier(n_jobs=-1), tuned_parameters, cv=3, scoring='roc_a
GSCV.fit(data_train, data_y_train)
```

```
Wall time: 13min 26s
```

Out[71]:

```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(n_jobs=-1),
             param_grid=[{'n_neighbors': array([ 1,  5, 10])}],
             scoring='roc_auc')
```

In [72]:

```
# Лучшее значение параметров
GSCV.best_params_
```

Out[72]:

```
{'n_neighbors': 10}
```

In [73]:

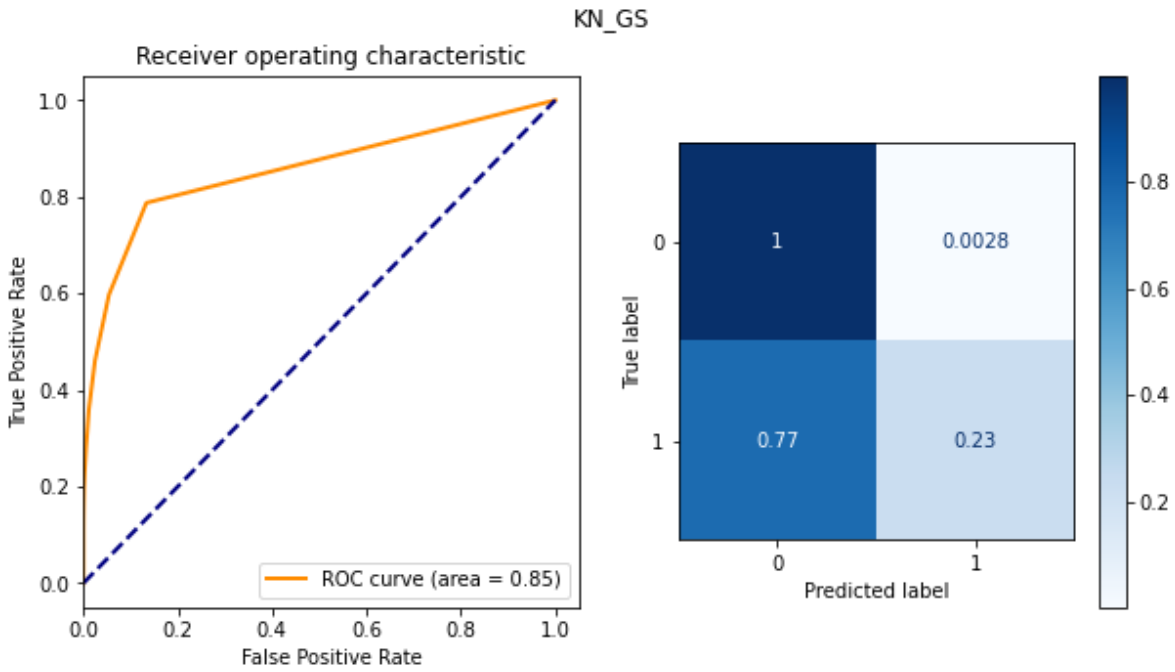
```
gscv_predicted_rocauc = {}
gscv_predicted_recall = {}
gscv_predicted_bAccur = {}
```

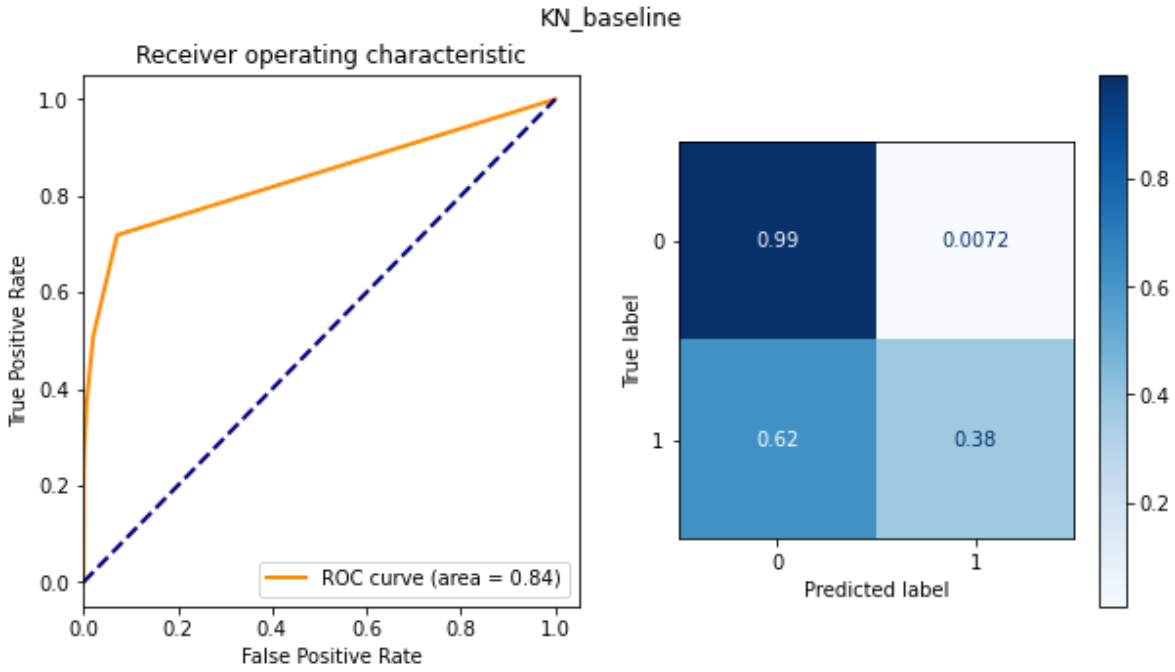
In [74]:

```
predict = GSCV.best_estimator_.predict(data_test)
predict_p = GSCV.best_estimator_.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
gscv_predicted_rocauc["KN"] = roc_auc_score(data_y_test, predict_p[:,1])
gscv_predicted_recall["KN"] = recall_score(data_y_test, predict)
gscv_predicted_bAccur["KN"] = balanced_accuracy_score(data_y_test, predict)
```

In [75]:

```
# Сравнение baseline модели с найденной лучшей моделью
draw_m(GSCV.best_estimator_, 'KN_GS', data_test, data_y_test, predict_p[:,1])
draw_m(KNC, 'KN_baseline', data_test, data_y_test, KNC.predict_proba(data_test)[:,1])
print('Метод ближайших соседей (Grid Search):')
print('ROC AUC: ', gscv_predicted_rocauc["KN"])
print('Recall: ', gscv_predicted_recall["KN"])
print('Balanced Accuracy: ', gscv_predicted_bAccur["KN"])
print()
print('Метод ближайших соседей (Baseline):')
print('ROC AUC: ', baseline_predicted_rocauc["KN"])
print('Recall: ', baseline_predicted_recall["KN"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["KN"])
```






```
Метод ближайших соседей (Grid Search):
ROC AUC:  0.8534342243007154
Recall:  0.2298559750875827
```

```
Balanced Accuracy:  0.6135104138404853

Метод ближайших соседей (Baseline):
ROC AUC:  0.8372066763690009
Recall:  0.3754379135850526
Balanced Accuracy:  0.6841223898472573
```

Подбор гиперпараметров улучшил качество модели.

Модель "Машина опорных векторов":

In [76]:

```
# Гиперпараметры
n_range = np.array(range(0, 3, 1))
n_range[0] = 0.1
tuned_parameters = [{'C': n_range}]
```

In [77]:

```
%%time
GSCV = GridSearchCV(SVC(kernel='rbf', random_state=1, probability=True, max_iter=100), tune
GSCV.fit(data_train, data_y_train)
```

```
Wall time: 9min

D:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:246: Converg
enceWarning: Solver terminated early (max_iter=100).  Consider pre-processin
g your data with StandardScaler or MinMaxScaler.
  warnings.warn('Solver terminated early (max_iter=%i).'
```

Out[77]:

```
GridSearchCV(cv=3,
             estimator=SVC(max_iter=100, probability=True, random_state=1),
             n_jobs=-1, param_grid=[{'C': array([0, 1, 2])}],
             scoring='roc_auc')
```

In [78]:

```
# Лучшее значение параметров
GSCV.best_params_
```

Out[78]:

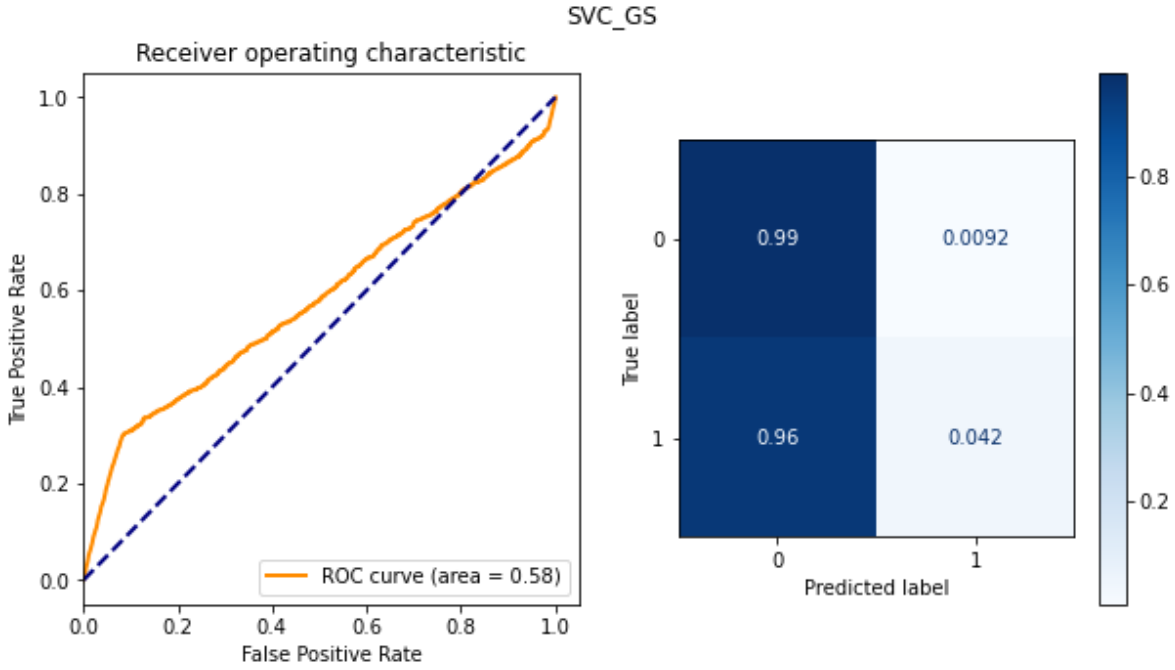
```
{'C': 1}
```

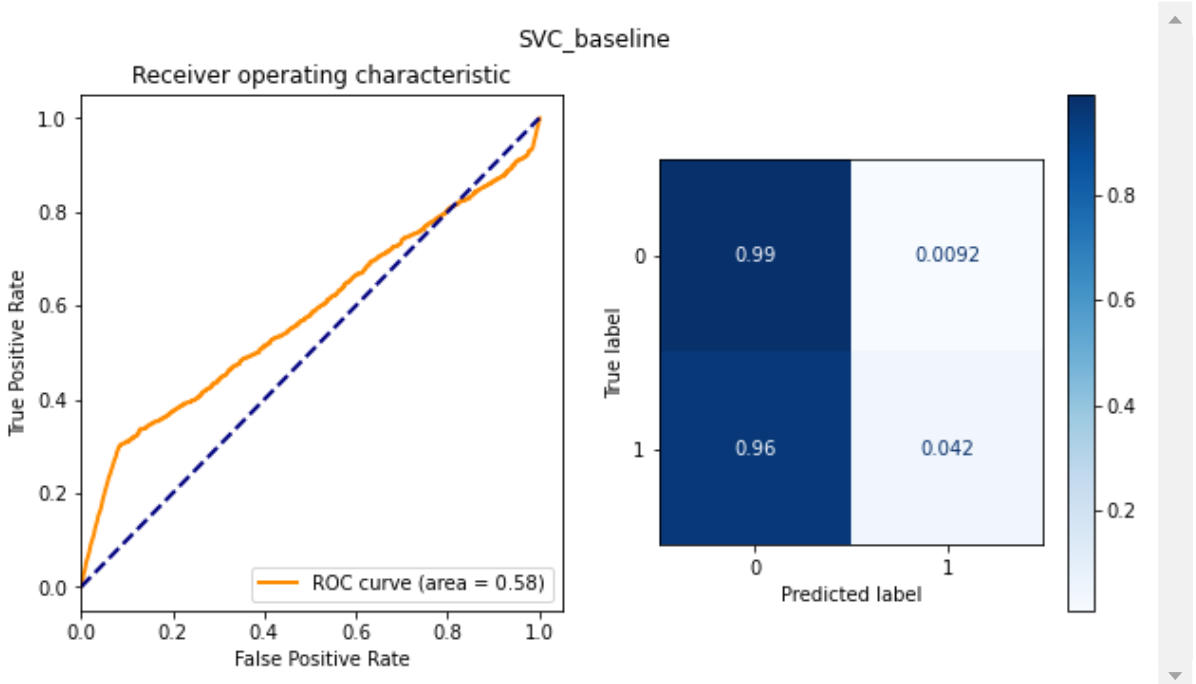
In [79]:

```
predict = GSCV.best_estimator_.predict(data_test)
predict_p = GSCV.best_estimator_.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
gscv_predicted_rocauc["SVC"] = roc_auc_score(data_y_test, predict_p[:,1])
gscv_predicted_recall["SVC"] = recall_score(data_y_test, predict)
gscv_predicted_bAccur["SVC"] = balanced_accuracy_score(data_y_test, predict)
```

In [80]:

```
# Сравнение baseline модели с найденной лучшей моделью
draw_m(GSCV.best_estimator_, 'SVC_GS', data_test, data_y_test, predict_p[:,1])
draw_m(svc, 'SVC_baseline', data_test, data_y_test, svc.predict_proba(data_test)[:,1])
print('Метод ближайших соседей (Grid Search):')
print('ROC AUC: ', gscv_predicted_rocauc["SVC"])
print('Recall: ', gscv_predicted_recall["SVC"])
print('Balanced Accuracy: ', gscv_predicted_bAccur["SVC"])
print()
print('Метод ближайших соседей (Baseline):')
print('ROC AUC: ', baseline_predicted_rocauc["SVC"])
print('Recall: ', baseline_predicted_recall["SVC"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["SVC"])
```





```
Метод ближайших соседей (Grid Search):
ROC AUC:  0.5791999785409971
Recall:  0.041650447644998055
Balanced Accuracy:  0.5162216181325547

Метод ближайших соседей (Baseline):
ROC AUC:  0.5791999785409971
Recall:  0.041650447644998055
Balanced Accuracy:  0.5162216181325547
```

Модель "Решающее дерево":

In [81]:

```
# Гиперпараметры
n_range = np.array(range(0,3,1))
n_range[0] = 0.1
tuned_parameters = {
    'max_depth': np.array(range(1,62,20)),
    'min_samples_leaf': n_range,
    'max_features': np.array(range(1,10,3))
}
```

In [82]:

```
%%time
GSCV = GridSearchCV(DecisionTreeClassifier(random_state=1), tuned_parameters, cv=3, scoring
GSCV.fit(data_train, data_y_train)
```

```
Wall time: 1min 21s
```

Out[82]:

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=1), n_jobs=
-1,
             param_grid={'max_depth': array([ 1, 21, 41, 61]),
                         'max_features': array([1, 4, 7]),
                         'min_samples_leaf': array([0, 1, 2])},
             scoring='roc_auc')
```

In [83]:

```
# Лучшее значение параметров
GSCV.best_params_
```

Out[83]:

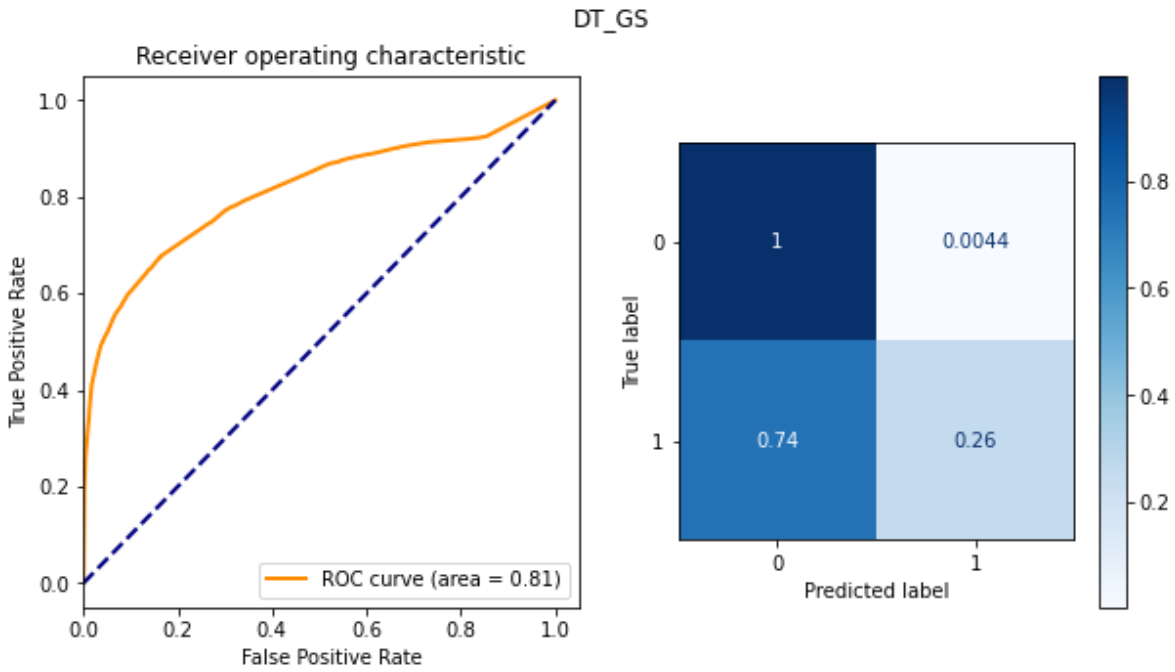
```
{'max_depth': 21, 'max_features': 7, 'min_samples_leaf': 2}
```

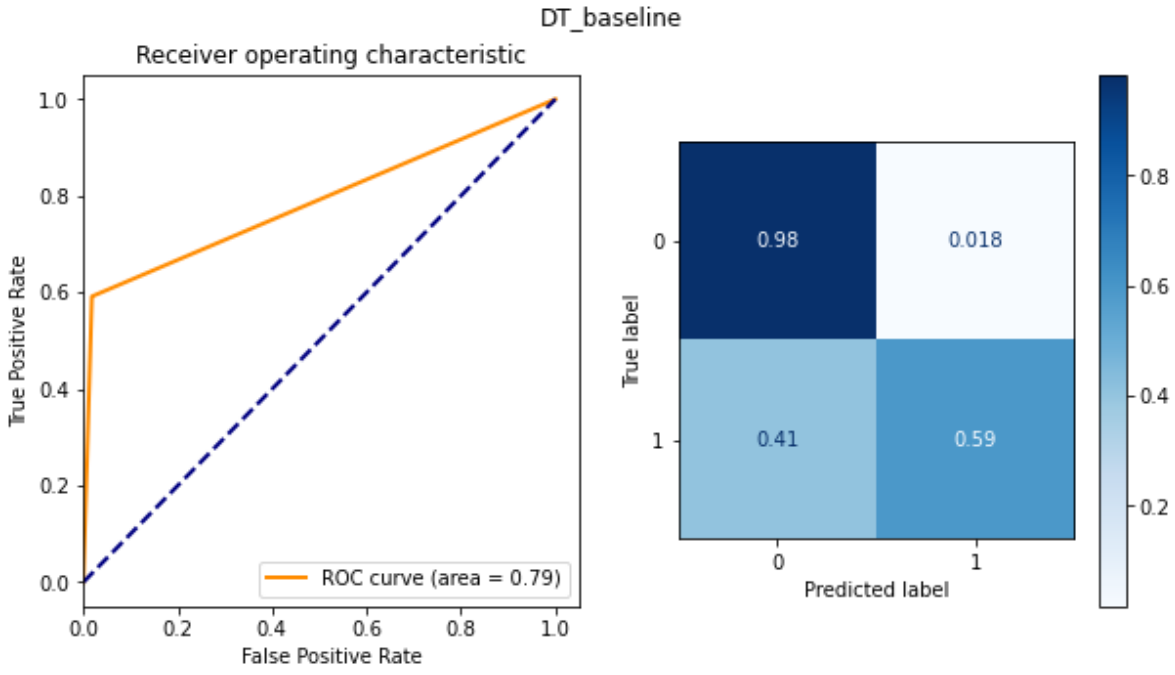
In [84]:

```
predict = GSCV.best_estimator_.predict(data_test)
predict_p = GSCV.best_estimator_.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
gscv_predicted_rocauc["DT"] = roc_auc_score(data_y_test, predict_p[:,1])
gscv_predicted_recall["DT"] = recall_score(data_y_test, predict)
gscv_predicted_bAccur["DT"] = balanced_accuracy_score(data_y_test, predict)
```

In [85]:

```
# Сравнение baseline модели с найденной лучшей моделью
draw_m(GSCV.best_estimator_, 'DT_GS', data_test, data_y_test, predict_p[:,1])
draw_m(dtc, 'DT_baseline', data_test, data_y_test, dtc.predict_proba(data_test)[:,1])
print('Метод ближайших соседей (Grid Search):')
print('ROC AUC: ', gscv_predicted_rocauc["DT"])
print('Recall: ', gscv_predicted_recall["DT"])
print('Balanced Accuracy: ', gscv_predicted_bAccur["DT"])
print()
print('Метод ближайших соседей (Baseline):')
print('ROC AUC: ', baseline_predicted_rocauc["DT"])
print('Recall: ', baseline_predicted_recall["DT"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["DT"])
```






```
Метод ближайших соседей (Grid Search):
ROC AUC:  0.8104181800356847
Recall:  0.25943947061113276
```

```
Balanced Accuracy:  0.6275021447598005

Метод ближайших соседей (Baseline):
ROC AUC:  0.786755931457974
Recall:  0.591475282210977
Balanced Accuracy:  0.7866988543240123
```

Подбор гиперпараметров улучшил качество модели.

Модель "Случайный лес":

In [91]:

```
# Гиперпараметры
tuned_parameters = {
    'n_estimators': np.array(range(50,160,50)),
    'max_features': np.array(range(1,10,3)),
    'min_samples_split': np.array([0.1, 0.4, 0.9]),
    'min_samples_leaf': np.array([0.1, 0.3, 0.5]),
    'max_depth': np.array(range(1,62,20))
}
```

In [92]:

```
%%time
GSCV = GridSearchCV(RandomForestClassifier(random_state=1, n_jobs=-1), tuned_parameters, cv
GSCV.fit(data_train, data_y_train)
```

```
Wall time: 59min 5s
```

Out[92]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(n_jobs=-1, random_state=
1),
             param_grid={'max_depth': array([ 1, 21, 41, 61]),
                         'max_features': array([1, 4, 7]),
                         'min_samples_leaf': array([0.1, 0.3, 0.5]),
                         'min_samples_split': array([0.1, 0.4, 0.9]),
                         'n_estimators': array([ 50, 100, 150])},
             scoring='roc_auc')
```

In [93]:

```
# Лучшее значение параметров
GSCV.best_params_
```

Out[93]:

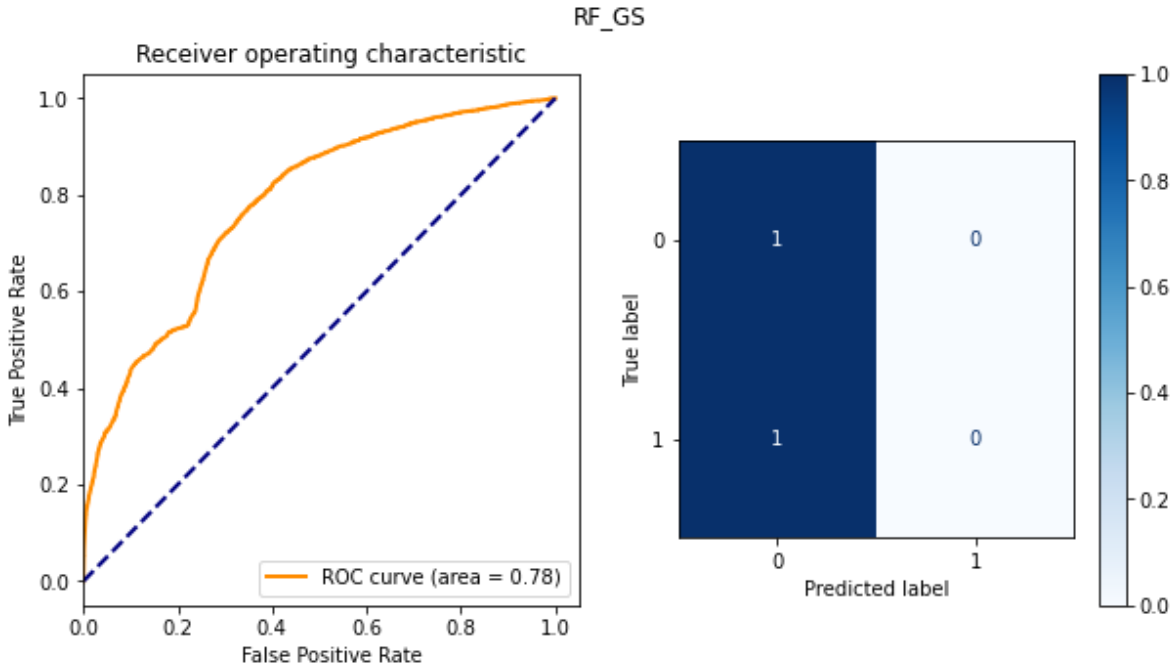
```
{'max_depth': 21,
 'max_features': 7,
 'min_samples_leaf': 0.1,
 'min_samples_split': 0.1,
 'n_estimators': 100}
```

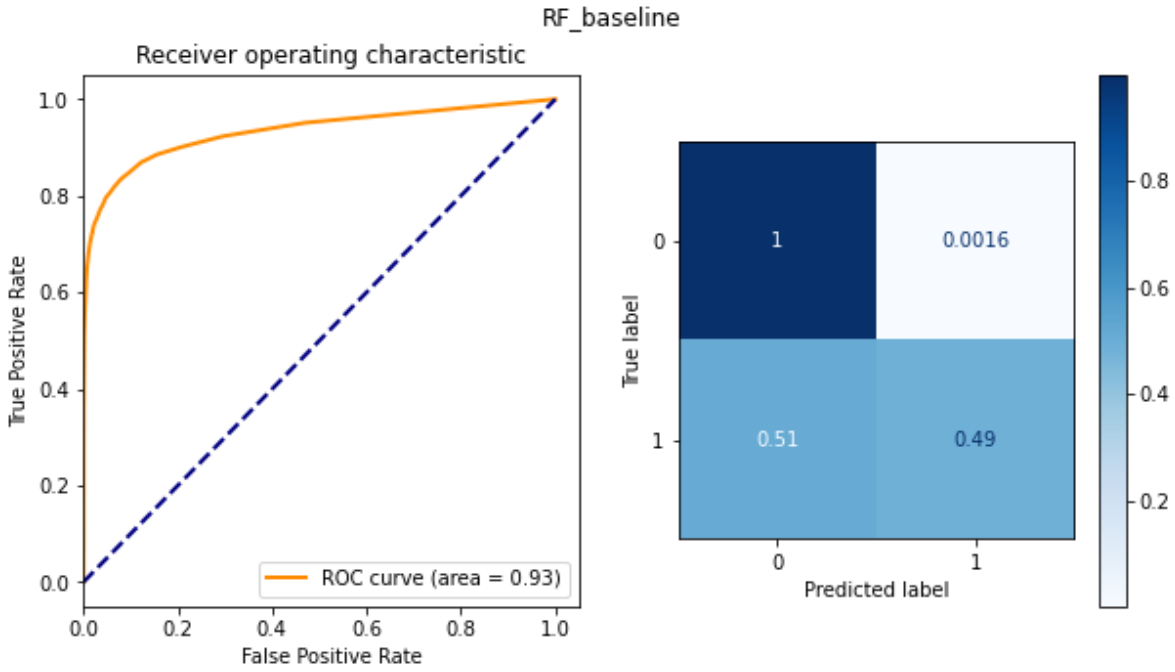
In [94]:

```
predict = GSCV.best_estimator_.predict(data_test)
predict_p = GSCV.best_estimator_.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
gscv_predicted_rocauc["RF"] = roc_auc_score(data_y_test, predict_p[:,1])
gscv_predicted_recall["RF"] = recall_score(data_y_test, predict)
gscv_predicted_bAccur["RF"] = balanced_accuracy_score(data_y_test, predict)
```

In [96]:

```
# Сравнение baseline модели с найденной лучшей моделью
draw_m(GSCV.best_estimator_, 'RF_GS', data_test, data_y_test, predict_p[:,1])
draw_m(RFC, 'RF_baseline', data_test, data_y_test, RFC.predict_proba(data_test)[:,1])
print('Метод ближайших соседей (Grid Search):')
print('ROC AUC: ', gscv_predicted_rocauc["RF"])
print('Recall: ', gscv_predicted_recall["RF"])
print('Balanced Accuracy: ', gscv_predicted_bAccur["RF"])
print()
print('Метод ближайших соседей (Baseline):')
print('ROC AUC: ', baseline_predicted_rocauc["RF"])
print('Recall: ', baseline_predicted_recall["RF"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["RF"])
```






```
Метод ближайших соседей (Grid Search):
ROC AUC:  0.7789977026634555
Recall:  0.0
```

```
Balanced Accuracy:  0.5

Метод ближайших соседей (Baseline):
ROC AUC:  0.9328265098547771
Recall:  0.48871156091864537
Balanced Accuracy:  0.7435662901542637
```

Модель "Градиентный бустинг":

In [102]:

```
# Гиперпараметры
tuned_parameters = {
    'num_leaves': np.array(range(21,45,10)),
    'min_data_in_leaf': np.array(range(0,4,1)),
    'max_depth': np.array(range(-1,62,20)),
    'min_gain_to_split': np.array(range(0,4,1)),
    'min_sum_hessian_in_leaf': np.array(range(0,3,1))
}
```

In [104]:

```
%%time
GSCV = GridSearchCV(LGBMClassifier(random_state=1), tuned_parameters, cv=3, scoring='roc_au
GSCV.fit(data_train, data_y_train)
```

In [ ]:

```
# Лучшее значение параметров
GSCV.best_params_
```

In [ ]:

```
predict = GSCV.best_estimator_.predict(data_test)
predict_p = GSCV.best_estimator_.predict_proba(data_test)
# Сохранение значений оценок метрик для данной модели
gscv_predicted_rocauc["LGBM"] = roc_auc_score(data_y_test, predict_p[:,1])
gscv_predicted_recall["LGBM"] = recall_score(data_y_test, predict)
gscv_predicted_bAccur["LGBM"] = balanced_accuracy_score(data_y_test, predict)
```

In [ ]:

```
# Сравнение baseline модели с найденной лучшей моделью
draw_m(GSCV.best_estimator_, 'LGBM_GS', data_test, data_y_test, predict_p[:,1])
draw_m(LGBMC, 'LGBM_baseline', data_test, data_y_test, LGBMC.predict_proba(data_test)[:,1])
print('Метод ближайших соседей (Grid Search):')
print('ROC AUC: ', gscv_predicted_rocauc["LGBM"])
print('Recall: ', gscv_predicted_recall["LGBM"])
print('Balanced Accuracy: ', gscv_predicted_bAccur["LGBM"])
print()
print('Метод ближайших соседей (Baseline):')
print('ROC AUC: ', baseline_predicted_rocauc["LGBM"])
print('Recall: ', baseline_predicted_recall["LGBM"])
print('Balanced Accuracy: ', baseline_predicted_bAccur["LGBM"])
```

Применение к выбранному набору данных AutoML

Используем библиотеку TPOT (https://github.com/EpistasisLab/tpot)

In [97]:

```
from tpot import TPOTClassifier
```

```
D:\ProgramData\Anaconda3\lib\site-packages\tpot\builtins\__init__.py:36: Use
rWarning: Warning: optional dependency `torch` is not available. - skipping
import of NN models.
  warnings.warn("Warning: optional dependency `torch` is not available. - sk
ipping import of NN models.")
```

In [98]:

```
tpot = TPOTClassifier(random_state=1, n_jobs=-1, scoring='roc_auc', verbosity=2, max_time_m
tpot.fit(data_train, data_y_train)
```

Optimization Progress: 45/100 [32:28<16:03, 17.51s/pipeline]

45%

```
32.67 minutes have elapsed. TPOT will close down.
TPOT closed during evaluation in one generation.
WARNING: TPOT may not provide a good pipeline if TPOT is stopped/interrupted
in a early generation.


TPOT closed prematurely. Will use the current best pipeline.

Best pipeline: DecisionTreeClassifier(SelectPercentile(input_matrix, percent
ile=96), criterion=entropy, max_depth=6, min_samples_leaf=8, min_samples_spl
it=11)
```

Out[98]:

```
TPOTClassifier(max_time_mins=30, n_jobs=-1, random_state=1, scoring='roc_au
c',
               verbosity=2)
```

In [99]:

```
# Информация о лучшей модели
tpot.fitted_pipeline_
```

Out[99]:

```
Pipeline(steps=[('selectpercentile', SelectPercentile(percentile=96)),
                ('decisiontreeclassifier',
                 DecisionTreeClassifier(criterion='entropy', max_depth=6,
                                        min_samples_leaf=8,
                                        min_samples_split=11,
                                        random_state=1))])
```

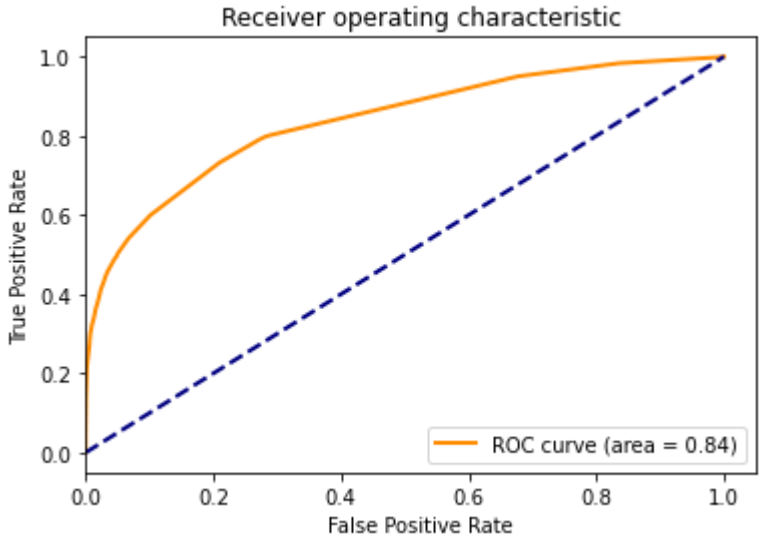
In [100]:

```
predict = tpot.predict(data_test)
predict_p = tpot.predict_proba(data_test)
tpot_predicted_rocauc = roc_auc_score(data_y_test, predict_p[:,1])
tpot_predicted_recall = recall_score(data_y_test, predict)
tpot_predicted_bAccur = balanced_accuracy_score(data_y_test, predict)
```

In [101]:

```
print('Оценки AutoML (TPOT) модели:')
print('ROC AUC: ', tpot_predicted_rocauc)
print('Recall: ', tpot_predicted_recall)
print('Balanced Accuracy: ', tpot_predicted_bAccur)
fig, ax = plt.subplots()
draw_roc_curve(data_y_test, predict_p[:,1], ax)
plt.show()
```

```
Оценки AutoML (TPOT) модели:
ROC AUC:  0.8371889866779217
Recall:  0.21564811210587778
Balanced Accuracy:  0.6066871900136539
```



Формирование выводов о качестве построенных моделей на основе выбранных метрик

In [105]:

```
# Визуализация качества моделей
def vis_models_quality(dit1, dit2, dit3, str_headers, nm, figsize=(10, 5)):
    dt1 = dict(sorted(dit1.items(), key=lambda x: x[1]))
    fig, ax = plt.subplots(ncols=3, figsize=figsize)
    pos = np.arange(len(dt1))
    rects = ax[0].barh(pos, dt1.values(),
                       align='center',
                       height=0.5,
                       tick_label=list(dt1.keys()))
    ax[0].set_title(str_headers[0])
    for a,b in zip(pos, dt1.values()):
        ax[0].text(0.2, a-0.1, str(round(b,3)), color='white')

    dt2 = dict(sorted(dit2.items(), key=lambda x: x[1]))
    pos = np.arange(len(dt2))
    rects = ax[1].barh(pos, dt2.values(),
                       align='center',
                       height=0.5,
                       tick_label=list(dt2.keys()))
    ax[1].set_title(str_headers[1])
    for a,b in zip(pos, dt2.values()):
        ax[1].text(0.2, a-0.1, str(round(b,3)), color='white')

    dt3 = dict(sorted(dit3.items(), key=lambda x: x[1]))
    pos = np.arange(len(dt3))
    rects = ax[2].barh(pos, dt3.values(),
                       align='center',
                       height=0.5,
                       tick_label=list(dt3.keys()))
    ax[2].set_title(str_headers[2])
    for a,b in zip(pos, dt3.values()):
        ax[2].text(0.2, a-0.1, str(round(b,3)), color='white')
    fig.suptitle(nm)
    plt.show()
```

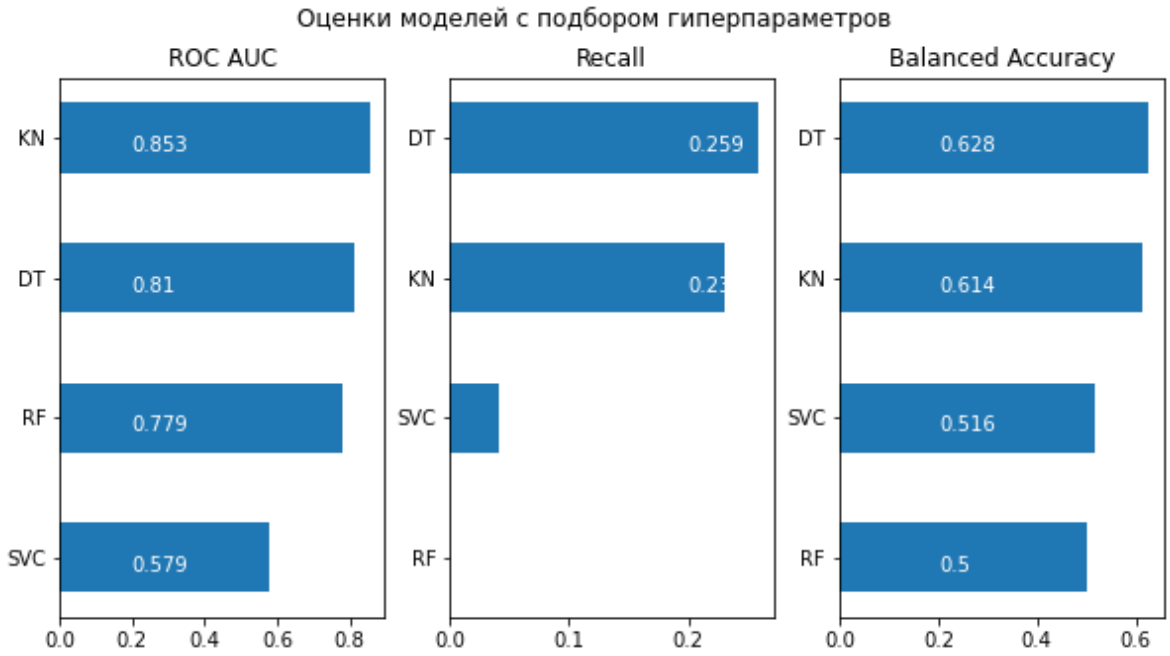
In [106]:

```
print('Оценки AutoML (TPOT) модели:')
print('ROC AUC: ', tpot_predicted_rocauc)
print('Recall: ', tpot_predicted_recall)
print('Balanced Accuracy: ', tpot_predicted_bAccur)
```

```
Оценки AutoML (TPOT) модели:
ROC AUC:  0.8371889866779217
Recall:  0.21564811210587778
Balanced Accuracy:  0.6066871900136539
```

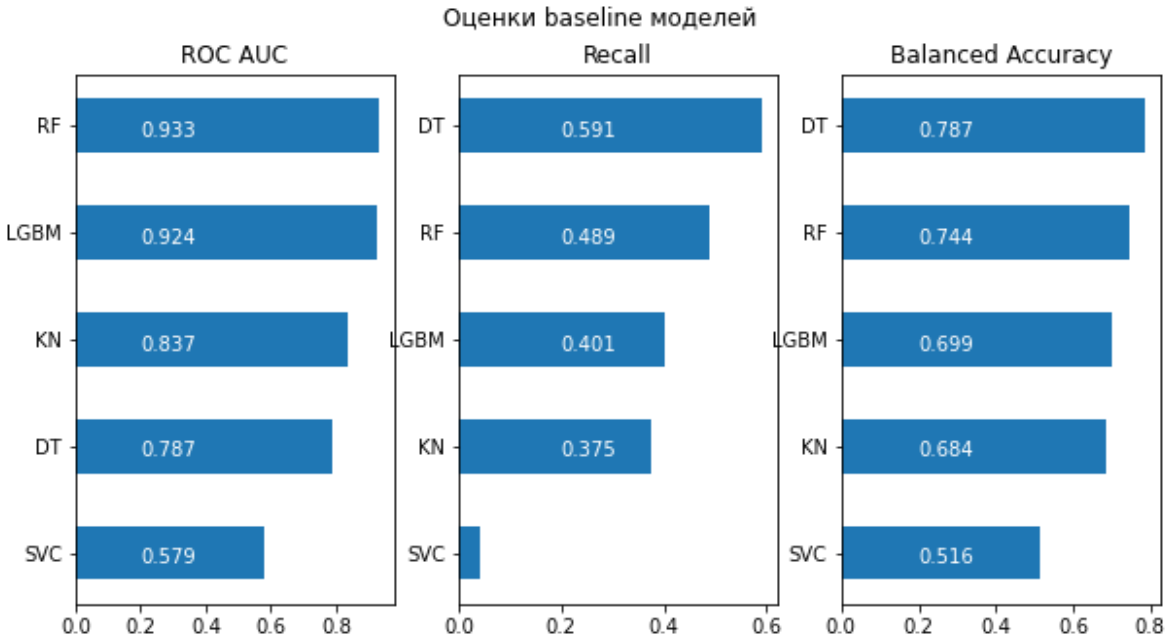
In [107]:

```
vis_models_quality(gscv_predicted_rocauc, gscv_predicted_recall, gscv_predicted_bAccur, ['R
```



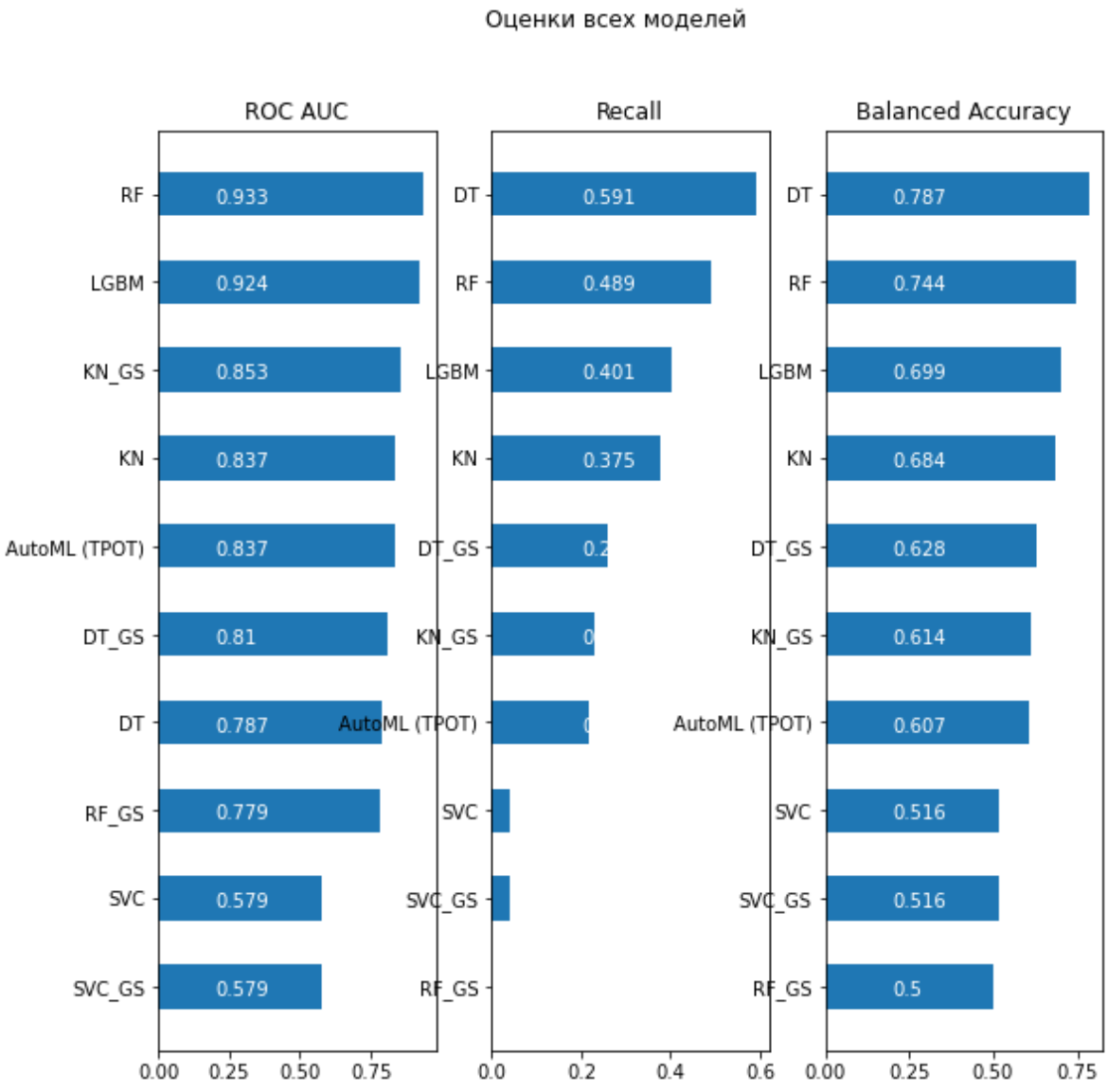
In [108]:

```
vis_models_quality(baseline_predicted_rocauc, baseline_predicted_recall, baseline_predicted
```



In [109]:

```
ra_all = {'AutoML (TPOT)':tpot_predicted_rocauc}
for i in gscv_predicted_rocauc:
    ra_all[i+'_GS'] = gscv_predicted_rocauc[i]
for i in baseline_predicted_rocauc:
    ra_all[i] = baseline_predicted_rocauc[i]

re_all = {'AutoML (TPOT)':tpot_predicted_recall}
for i in gscv_predicted_recall:
    re_all[i+'_GS'] = gscv_predicted_recall[i]
for i in baseline_predicted_recall:
    re_all[i] = baseline_predicted_recall[i]

ba_all = {'AutoML (TPOT)':tpot_predicted_bAccur}
for i in gscv_predicted_bAccur:
    ba_all[i+'_GS'] = gscv_predicted_bAccur[i]
for i in baseline_predicted_bAccur:
    ba_all[i] = baseline_predicted_bAccur[i]

vis_models_quality(ra_all, re_all, ba_all, ['ROC AUC', 'Recall', 'Balanced Accuracy'], 'Оце
```



Вывод. На основании трех метрик, лучшей моделью оказалась **"Случайный лес"**. В отдельных случаях наиболее подходящей моделью является **"Дерево решений"**.

Разработка макета веб-приложения, предназначенного для анализа данных

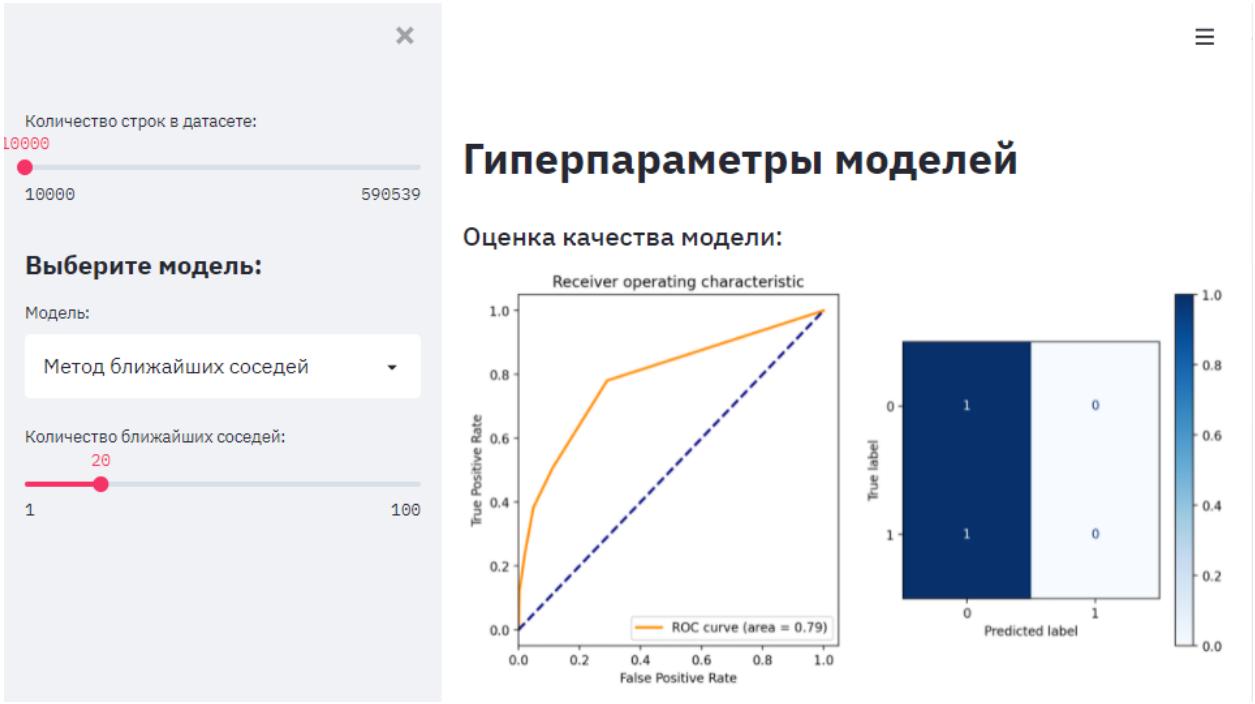
Текст программы

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from operator import itemgetter

@st.cache
def load_data():
    '''
    Загрузка данных
    '''
    org_data = pd.read_csv('E:/train.csv')
    org_data.drop(columns=['Unnamed: 0'], inplace=True)
    return org_data


def preprocess_data(data_in, rows):
    '''
    Разделение выборки на обучающую и тестовую
    '''
    data_train, data_test, data_y_train, data_y_test = train_test_split(data_in[0
:rows][data_in.columns.drop('isFraud')], data_in[0:rows]['isFraud'], random_state
=1)
    return data_train, data_test, data_y_train, data_y_test


# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    #plt.figure()
    lw = 2
    ax.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_xlim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
```

```
    ax.set_title('Receiver operating characteristic')
    ax.legend(loc="lower right")


st.title('Гиперпараметры моделей')
data = load_data()
row_slider = st.sidebar.slider('Количество строк в датасете:', min_value=10000, m
ax_value=data.shape[0]-1, value=10000, step=1000)
data_train, data_test, data_y_train, data_y_test = preprocess_data(data, row_slid
er)
st.sidebar.header('Выберите модель:')
sb = st.sidebar.selectbox('Модель:', ('Метод ближайших соседей', 'Решающее дерево
'))
if sb == 'Метод ближайших соседей':
    neig_slider = st.sidebar.slider('Количество ближайших соседей:', min_value=1,
 max_value=100, value=1, step=1)

    KNC = KNeighborsClassifier(n_neighbors=neig_slider, n_jobs=-
1).fit(data_train, data_y_train)
    data_test_predicted_knc = KNC.predict_proba(data_test)[:,1]

    st.subheader('Оценка качества модели:')
    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(data_y_test, data_test_predicted_knc, ax[0])
    plot_confusion_matrix(KNC, data_test, data_y_test, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    st.pyplot(fig)

if sb == 'Решающее дерево':
    max_depth = st.sidebar.slider('Глубина дерева:', min_value=1, max_value=100,
value=50, step=1)
    max_features = st.sidebar.slider('Число признаков для выбора расщепления:', m
in_value=1, max_value=30, value=14, step=1)
    min_samples_leaf = st.sidebar.slider('Минимальное количество выборок:', min_v
alue=1, max_value=5, value=1, step=1)

    dtc = DecisionTreeClassifier(random_state=1, max_depth=max_depth, max_feature
s=max_features, min_samples_leaf=min_samples_leaf).fit(data_train, data_y_train)
    data_test_predicted_dtc = dtc.predict_proba(data_test)[:,1]

    st.subheader('Оценка качества модели:')
    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(data_y_test, data_test_predicted_dtc, ax[0])
    plot_confusion_matrix(dtc, data_test, data_y_test, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    st.pyplot(fig)

    st.subheader('Важность признаков:')
```

```
# Сортировка значений важности признаков по убыванию
list_to_sort = list(zip(data_train.columns.values, dtc.feature_importances_))
sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
# Названия признаков
labels = [x for x,_ in sorted_list]
# Важности признаков
data = [x for _,x in sorted_list]
# Вывод графика
fig, ax = plt.subplots(figsize=(18,5))
ind = np.arange(len(labels))
plt.bar(ind, data)
plt.xticks(ind, labels, rotation='vertical')
# Вывод значений
for a,b in zip(ind, data):
    plt.text(a-0.05, b+0.01, str(round(b,3)))
st.pyplot(fig)

st.subheader('Список признаков, отсортированный на основе важности:')
labels
```

Экранные формы веб-приложения



Количество строк в датасете:

10000

10000 590539

Выберите модель:

Модель:

Решающее дерево

Глубина дерева:

90

1 100

Число признаков для выбора расщепления:

14

1 30

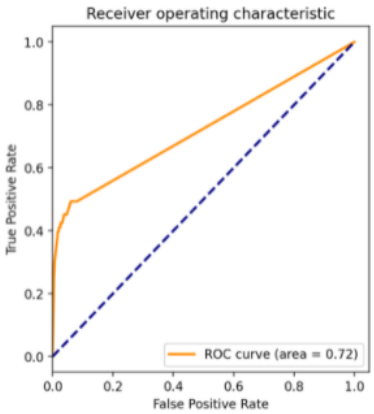
Минимальное количество выборок:

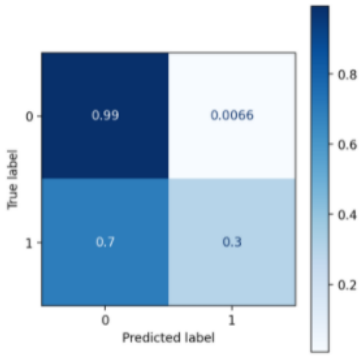
2

1 5

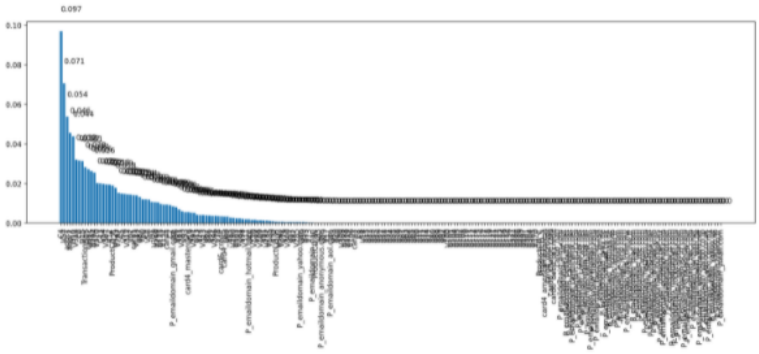
Гиперпараметры моделей

Оценка качества модели:





Важность признаков:



Список признаков, отсортированный на основе важности:


```
[
  0 : "C4"
  1 : "V53"
```

3. Заключение

В результате выполнения курсовой работы были построены и оценены несколькими метриками различные модели машинного обучения по выявлению мошеннических транзакций в выбранном наборе данных. В ходе выполнения работы были изучены и опробованы способы разведочного анализа данных, корреляционного анализа данных, методы обработки пропусков в данных, кодирования категориальных признаков, масштабирования данных, формирования обучающей и тестовой выборок, подбора гиперпараметров для различных моделей, был разработан макет веб-приложения, предназначенного для анализа данных и опробовано применение автоматического машинного обучения (AutoML) для построение модели. В результате было получено несколько хороших по оценкам различных метрик моделей машинного обучения для выполнения поставленной задачи.

4. Список литературы

1) Конспект лекций по дисциплине «Технологии машинного обучения». М.: МГТУ им. Н.Э.Баумана. 2021 г.

2) Pandas: powerful Python data analysis toolkit. Release 1.2.4 / Wes McKinney and the Pandas Development Team, 2021. - 3311 с.
3) Документация по библиотеке «sklearn». URL: https://scikit-learn.org/stable/index.html (Дата обращения: 03.06.2021)
4) Документация по библиотеке «Streamlit». URL: https://docs.streamlit.io/en/stable/ (Дата обращения: 03.06.2021)