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 Радиотехнический (РТ)

КАФЕДРА Системы обработки информации и управления (ИУ-5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Прогнозирование оформления клиентом срочного депозита

Студент РТ5-61Б _____ _____ Т. М. Алиев
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ Ю. Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

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

2022 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 __РТ5_____
(Индекс)

_____ _____
(И.О.Фамилия)

« _____ » _____ 2022___ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме ___Прогнозирование оформления клиентом срочного депозита_____

_____

_____

Студент группы ____РТ5-61Б_____

_____Алиев Тимур Муратович_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

_____учебная_____

Источник тематики (кафедра, предприятие, НИР) _____кафедра_____

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

Техническое задание *решить задачу классификации на основе материалов дисциплины по выбранной предметной области*_____

_____

_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _23_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

_____

Дата выдачи задания « _15_ » __февраля__ 2022 г.

Руководитель НИР _____ ____Ю. Е. Гапанюк___
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Т. М. Алиев_____
(Подпись, дата) (И.О.Фамилия)

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

Содержание

1) Введение

В качестве предметной области был выбран датасет с информацией о клиентах банка. В исследовании будет решаться задача бинарной классификации.

Данные связаны с кампаниями прямого маркетинга португальского банковского учреждения. Маркетинговые кампании были основаны на телефонных звонках. Часто требовалось более одного контакта с одним и тем же клиентом, чтобы узнать, будет ли продукт (срочный банковский депозит) подписан («да») или нет («нет»).

Цель классификации — предсказать, подпишется ли клиент на срочный депозит (переменная y).

2) Описание датасета

В качестве набора данных мы будем использовать набор данных прогнозирования подпишется ли клиент на срочный депозит: https://www.kaggle.com/datasets/hariharanpavan/bank-marketing-dataset-analysis-classification

Данные связаны с кампаниями прямого маркетинга (телефонными звонками) португальского банковского учреждения. Цель классификации — предсказать, подпишется ли клиент на срочный депозит (переменная y).

- Age Возраст клиента
- Job Работа заказчика
- Martial Семейное положение клиента
- Education Уровень образования клиента
- Default Имеет ли кредит по умолчанию?
- Housing Если у клиента есть жилищный кредит
- Loan Имеет личный кредит
- Balance Индивидуальный баланс клиента
- Contact Тип связи
- Month Последний контактный месяц года
- Day Последний контактный день недели
- Duration Длительность последнего контакта, в секундах
- Campaign Количество контактов, выполненных во время этой кампании и для этого клиента
- Pdays Количество дней, прошедших с момента последнего контакта с клиентом из предыдущей кампании
- Previous Количество контактов, выполненных до этой кампании и для этого клиента
- Poutcome результат предыдущей маркетинговой кампании
- Y целевой признак (подпишется ли клиент на срочный депозит)

3) Импорт библиотек

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
%matplotlib inline
sns.set(style="ticks")
import warnings
warnings.filterwarnings('ignore')
```

4) Загрузка данных

```
first_data = pd.read_csv('bank-full.csv')
```

```
# Удалим дубликаты записей, если они присутствуют
data = first_data.drop_duplicates()
```

5) Проведение разведочного анализа данных

Основные характеристики датасета

```
data.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

```
data.shape
```

```
(45211, 17)
```

```
# Список колонок
data.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

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

```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
y            object
dtype: object
```

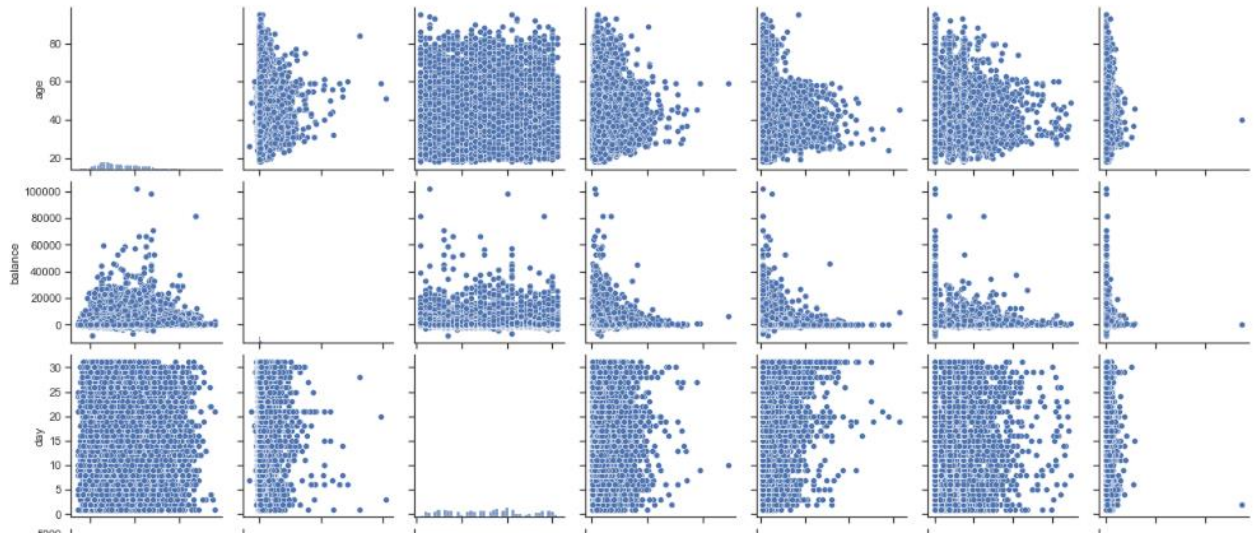
```
# Проверим наличие пустых значений
data.isnull().sum()
```

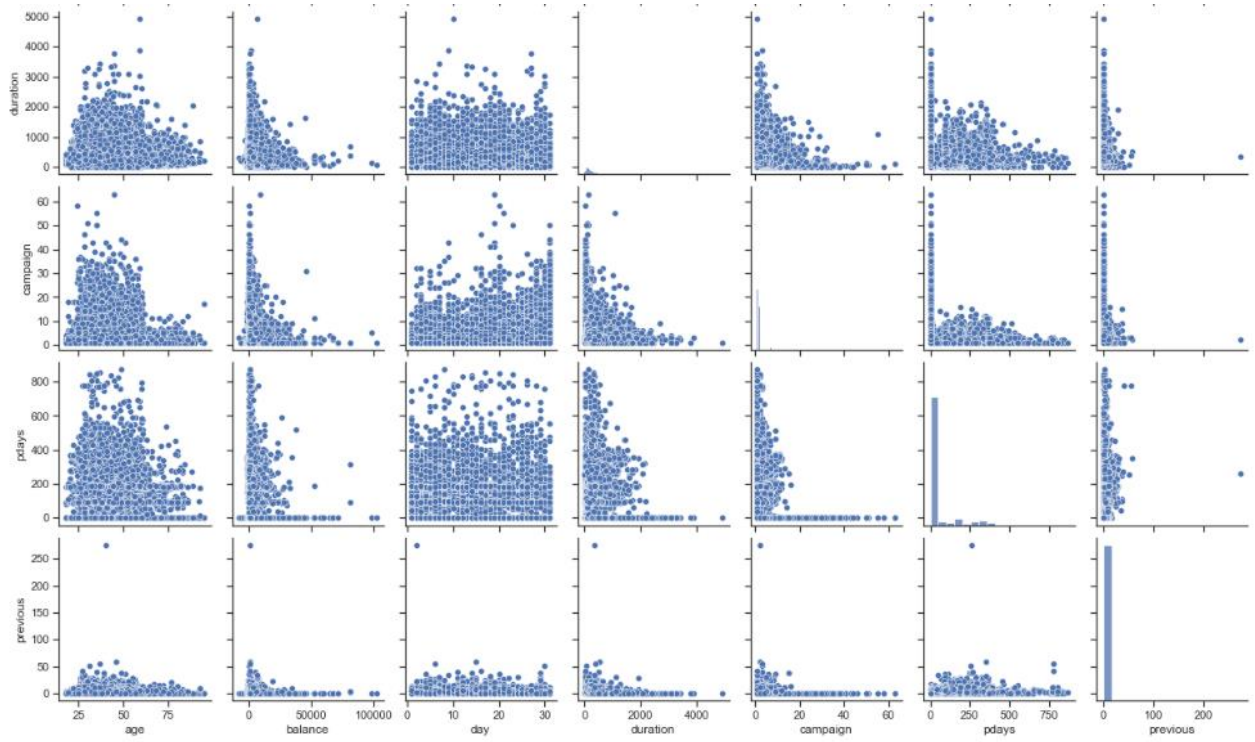
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

6) Построение графиков для понимания структуры данных

```
# Парные диаграммы
sns.pairplot(data)
```

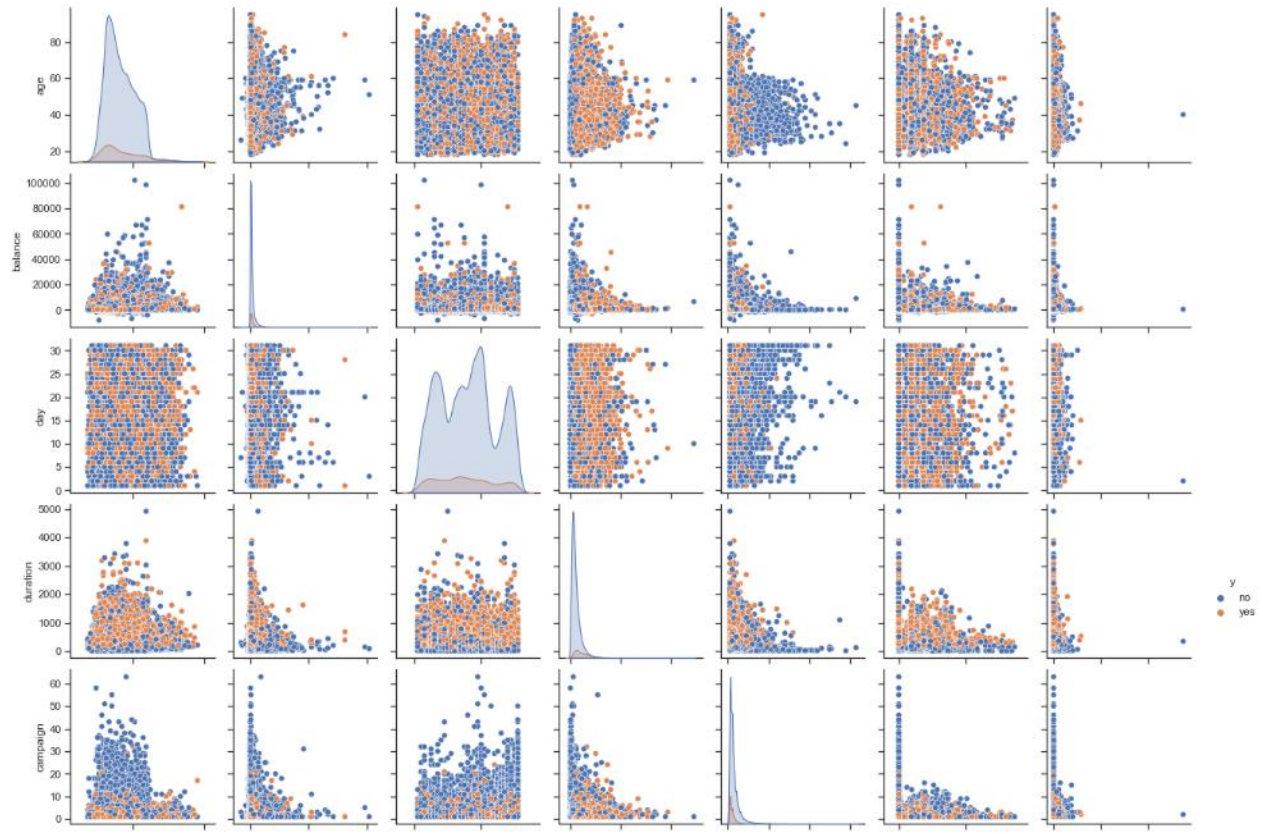
```
<seaborn.axisgrid.PairGrid at 0x20ba3a47a60>
```

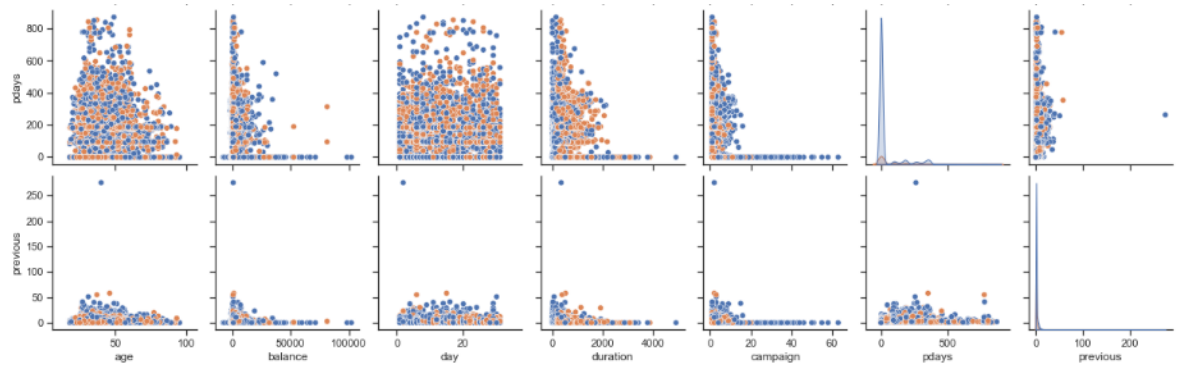





```
1 sns.pairplot(data, hue="y")
```

```
<seaborn.axisgrid.PairGrid at 0x20bac4a4d90>
```

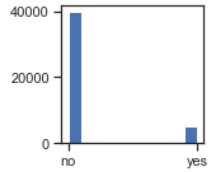





```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 'no' и 'yes', потом преобразуем к 0 и 1
data['y'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
# Оценим дисбаланс классов для stroke
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['y'])
plt.show()
```



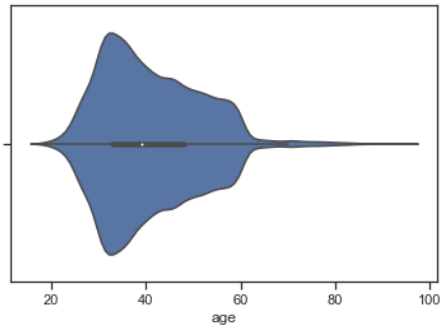

```
data['y'].value_counts()
```

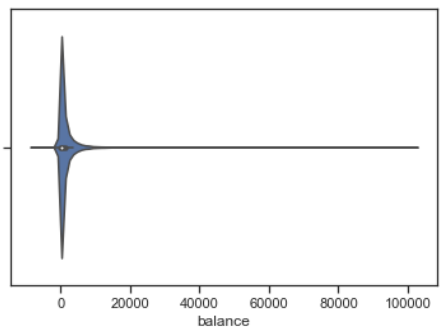
```
no     39922
yes     5289
Name: y, dtype: int64
```

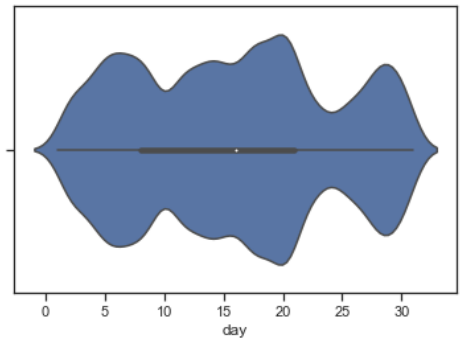
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['y'].value_counts()
print('Класс no составляет {}%, а класс yes составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

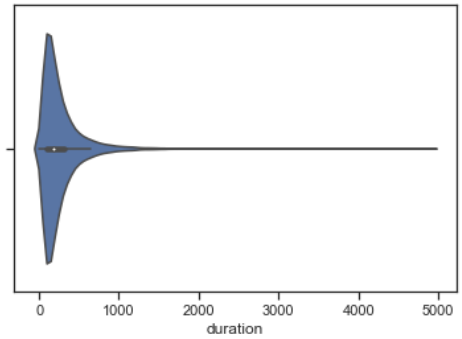
```
Класс no составляет 88.3%, а класс yes составляет 11.700000000000001%.
```

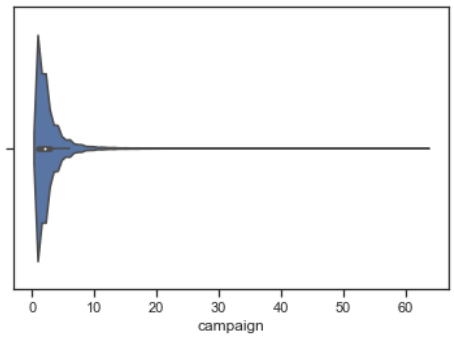
```
# Скрипичные диаграммы для числовых колонок
for col in ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']:
    sns.violinplot(x=data[col])
    plt.show()
```

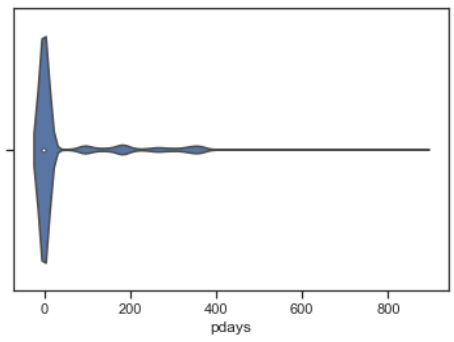


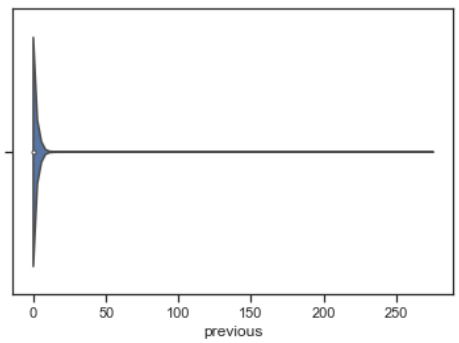



0
5
10
15
20
25
30
day


0
1000
2000
3000
4000
5000
duration


0
10
20
30
40
50
60
campaign


0
200
400
600
800
pdays


0
50
100
150
200
250
previous

7) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей

```
data.dtypes
```

```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
y            object
dtype: object
```

Категориальные признаки присутствуют, закодируем их.

```
data['job'].unique()
```

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

```
job = LabelEncoder()
code_job = job.fit_transform(data["job"])
data["job"] = code_job
data = data.astype({"job":"int64"})
np.unique(code_job)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
data['marital'].unique()
```

```
array(['married', 'single', 'divorced'], dtype=object)
```

```
marital = LabelEncoder()
code_marital = marital.fit_transform(data["marital"])
data["marital"] = code_marital
data = data.astype({"marital":"int64"})
np.unique(code_marital)
```

```
array([0, 1, 2])
```

```
data['education'].unique()
```

```
array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)
```

```
education = LabelEncoder()
code_education = education.fit_transform(data["education"])
data["education"] = code_education
data = data.astype({"education":"int64"})
np.unique(code_education)
```

```
array([0, 1, 2, 3])
```

```
data['default'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
default = LabelEncoder()
code_default = default.fit_transform(data["default"])
data["default"] = code_default
data = data.astype({"default":"int64"})
np.unique(code_default)
```

```
array([0, 1])
```

```
data['housing'].unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
housing = LabelEncoder()
code_housing = housing.fit_transform(data["housing"])
data["housing"] = code_housing
data = data.astype({"housing":"int64"})
np.unique(code_housing)
```

```
array([0, 1])
```

```
data['loan'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
loan = LabelEncoder()
code_loan = loan.fit_transform(data["loan"])
data["loan"] = code_loan
data = data.astype({"loan":"int64"})
np.unique(code_loan)
```

```
array([0, 1])
```

```
data['contact'].unique()
```

```
array(['unknown', 'cellular', 'telephone'], dtype=object)
```

```
contact = LabelEncoder()
code_contact = contact.fit_transform(data["contact"])
data["contact"] = code_contact
data = data.astype({"contact":"int64"})
np.unique(code_contact)
```

```
array([0, 1, 2])
```

```
data['month'].unique()
```

```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb',
       'mar', 'apr', 'sep'], dtype=object)
```

```
month = LabelEncoder()
code_month = month.fit_transform(data["month"])
data["month"] = code_month
data = data.astype({"month":"int64"})
np.unique(code_month)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
data['poutcome'].unique()
```

```
array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

```
poutcome = LabelEncoder()
code_poutcome = poutcome.fit_transform(data["poutcome"])
data["poutcome"] = code_poutcome
data = data.astype({"poutcome":"int64"})
np.unique(code_poutcome)
```

```
array([0, 1, 2, 3])
```

```
data['y'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
y = LabelEncoder()
code_y = y.fit_transform(data["y"])
data["y"] = code_y
data = data.astype({"y":"int64"})
np.unique(code_y)
```

```
array([0, 1])
```

```
# Числовые колонки для масштабирования
scale_cols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
```

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

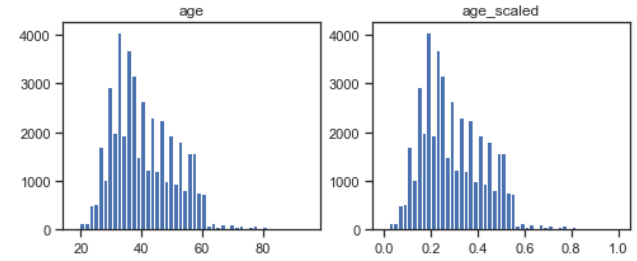
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

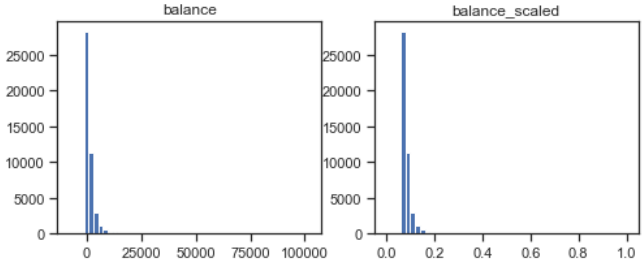
```
data.head()
```

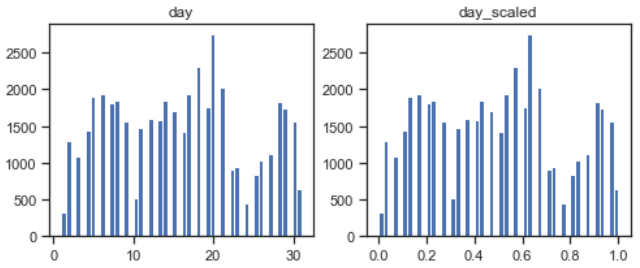
	age	job	marital	education	default	balance	housing	loan	contact	day	...	previous	poutcome	y	age_scaled	balance_scaled	day_scaled	duration_sc
0	58	4	1	2	0	2143	1	0	2	5	...	0	3	0	0.519481	0.092259	0.133333	0.05
1	44	9	2	1	0	29	1	0	2	5	...	0	3	0	0.337662	0.073067	0.133333	0.03
2	33	2	1	1	0	2	1	1	2	5	...	0	3	0	0.194805	0.072822	0.133333	0.01
3	47	1	1	3	0	1506	1	0	2	5	...	0	3	0	0.376623	0.086476	0.133333	0.01
4	33	11	2	3	0	1	0	0	2	5	...	0	3	0	0.194805	0.072812	0.133333	0.04

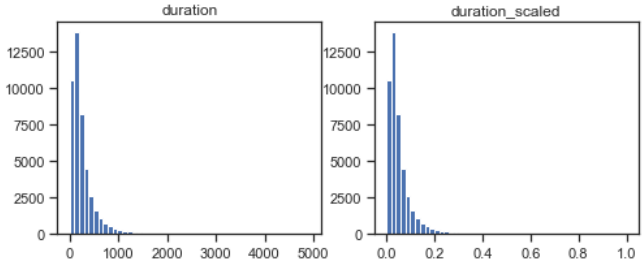
5 rows × 24 columns

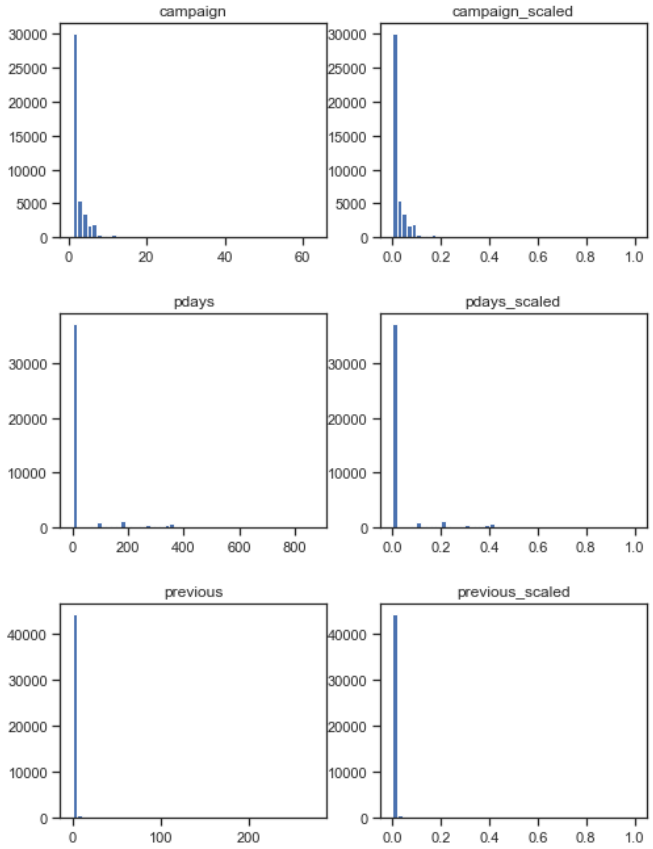
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```










campaign
campaign_scaled
pdays
pdays_scaled
previous
previous_scaled

8) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения

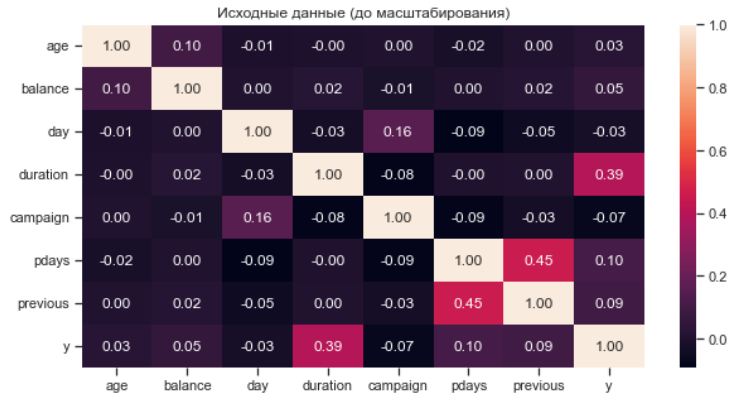
```
# Воспользуемся наличием тестовых выборок,
# включив их в корреляционную матрицу
corr_cols_1 = scale_cols + ['y']
corr_cols_1
```

```
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous', 'y']
```

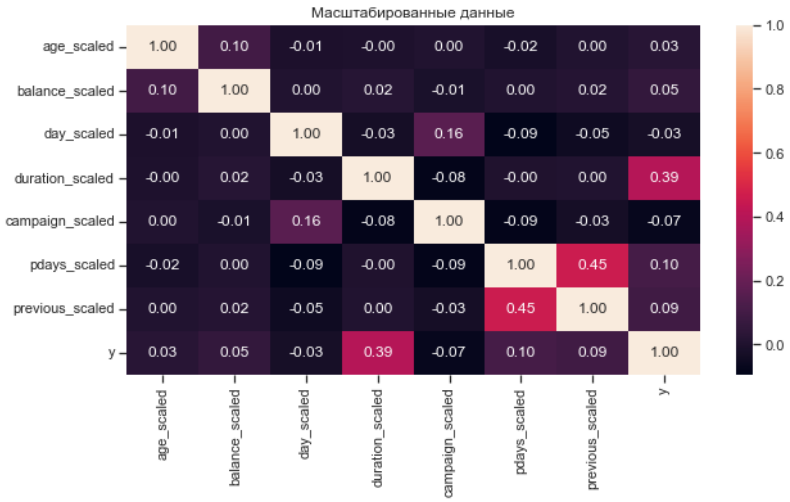
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['y']
corr_cols_2
```

```
['age_scaled',
 'balance_scaled',
 'day_scaled',
 'duration_scaled',
 'campaign_scaled',
 'pdays_scaled',
 'previous_scaled',
 'y']
```

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
ax.set_title('Масштабированные данные')
plt.show()
```



На основе корреляционной матрицы можно сделать следующие выводы:

Корреляционные матрицы для исходных и масштабированных данных совпадают.

Целевой признак классификации "y" наиболее сильно коррелирует с duration (0.39) и pdays (0.10). Эти признаки обязательно следует оставить в модели классификации.

9) Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи классификации будем использовать:

Метрики, формируемые на основе матрицы ошибок:

Метрика precision: Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Метрика recall (полнота): Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Метрика F1-мера: Для того, чтобы объединить precision и recall в единую метрику используется Fβ-мера, которая вычисляется как среднее гармоническое от precision и recall:

Метрика ROC AUC:

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.

10) Сохранение и визуализация метрик

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества.

```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

11) Выбор наиболее подходящих моделей для решения задачи классификации или регрессии

Для задачи классификации будем использовать следующие модели:

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

12) Формирование обучающей и тестовой выборок на основе исходного набора данных

```
X_train, X_test, y_train, y_test = train_test_split(data, data.y, random_state=1)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((33908, 24), (33908,), (11303, 24), (11303,))
```

13) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки

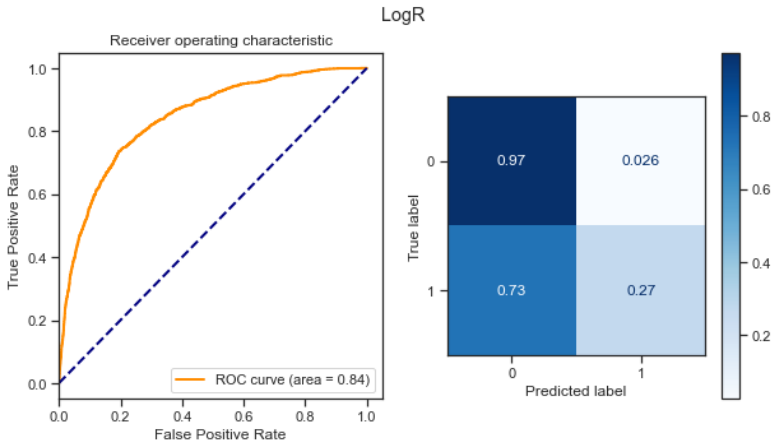
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(probability=True),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

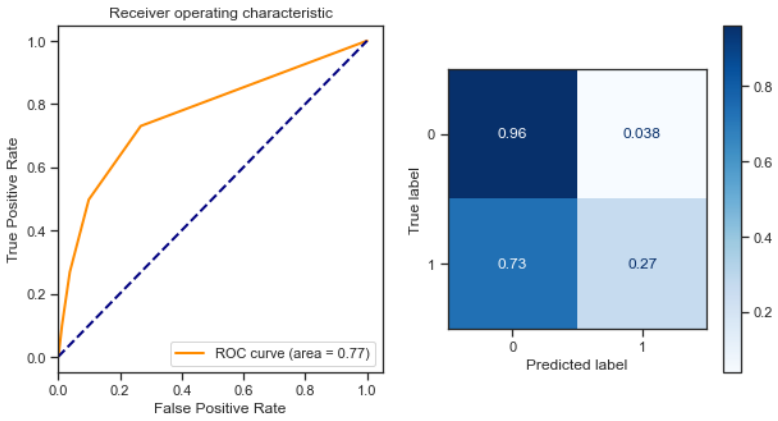
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    #plt.figure()
    lw = 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")
```

```python
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, y_train)
    # Предсказание значений
    Y_pred = model.predict(X_test)
    # Предсказание вероятности класса "1" для roc auc
    Y_pred_proba_temp = model.predict_proba(X_test)
    Y_pred_proba = Y_pred_proba_temp[:,1]

    precision = precision_score(y_test.values, Y_pred)
    recall = recall_score(y_test.values, Y_pred)
    f1 = f1_score(y_test.values, Y_pred)
    roc_auc = roc_auc_score(y_test.values, Y_pred_proba)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)

    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(y_test.values, Y_pred_proba, ax[0])
    plot_confusion_matrix(model, X_test, y_test.values, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    fig.suptitle(model_name)
    plt.show()
```

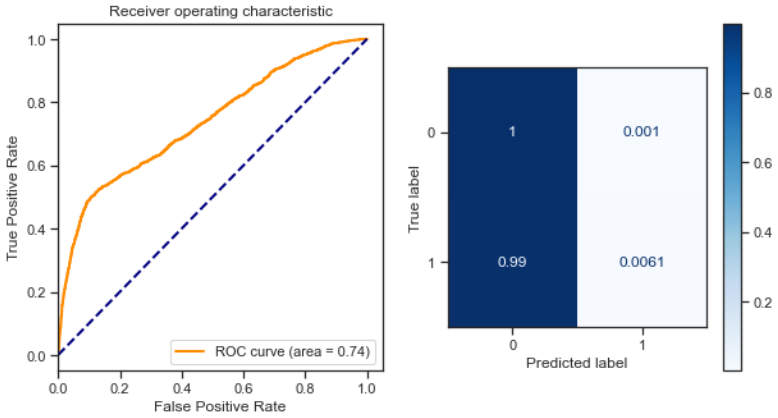
```python
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```



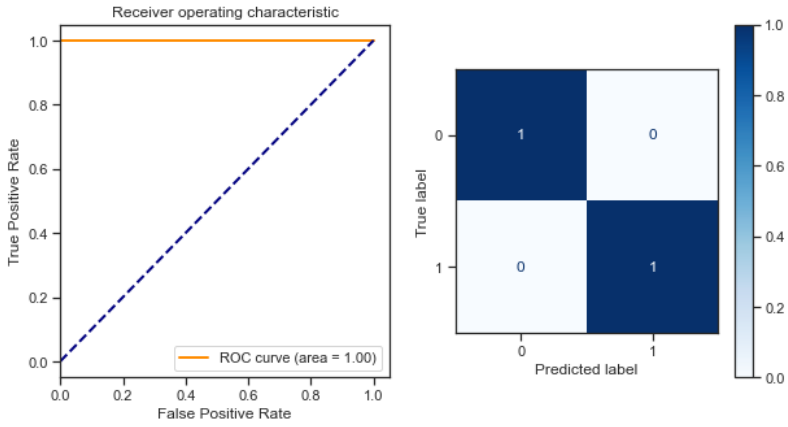
KNN_5



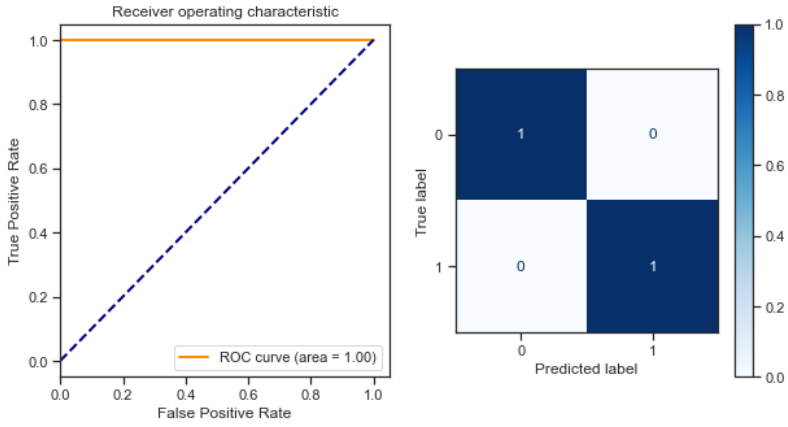
SVC

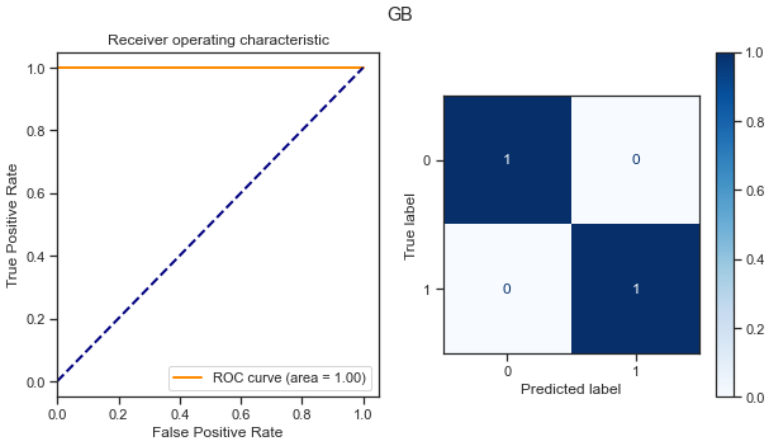


Tree



RF




GB
Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
True label
Predicted label
1
0
0
1

14) Подбор гиперпараметров для выбранных моделей

```
X_train.shape
```

```
(33908, 24)
```

```
n_range_list = list(range(0,1250,50))
n_range_list[0] = 1
```

```
n_range = np.array(n_range_list)
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([   1,   50,  100,  150,  200,  250,  300,  350,  400,  450,  500,
         550,  600,  650,  700,  750,  800,  850,  900,  950, 1000, 1050,
        1100, 1150, 1200])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='roc_auc')
clf_gs.fit(X_train, y_train)
```

```
CPU times: total: 9min 21s
Wall time: 6min 48s

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([   1,   50,  100,  150,  200,  250,  300,  350,  400,  450,  500,
        550,  600,  650,  700,  750,  800,  850,  900,  950, 1000, 1050,
       1100, 1150, 1200])}],
             scoring='roc_auc')
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=100)
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

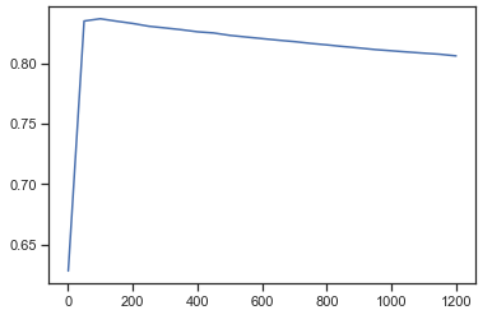
```
{'n_neighbors': 100}
```

```
clf_gs_best_params_txt = str(clf_gs.best_params_['n_neighbors'])
clf_gs_best_params_txt
```

```
'100'
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x20bb227ab90>]
```

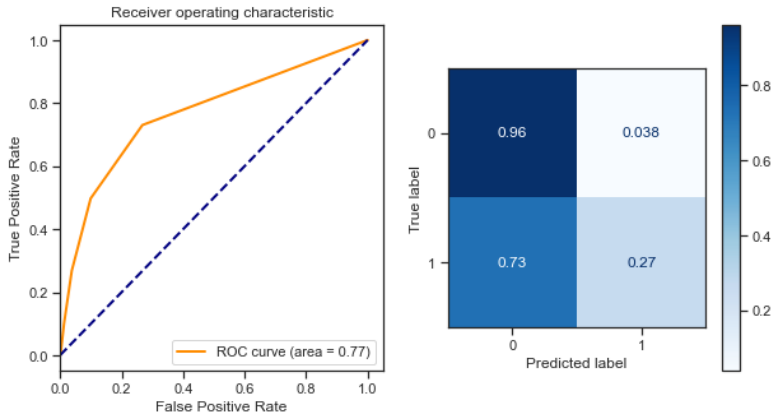


15) Сравнение качества полученных моделей с качеством baseline-моделей

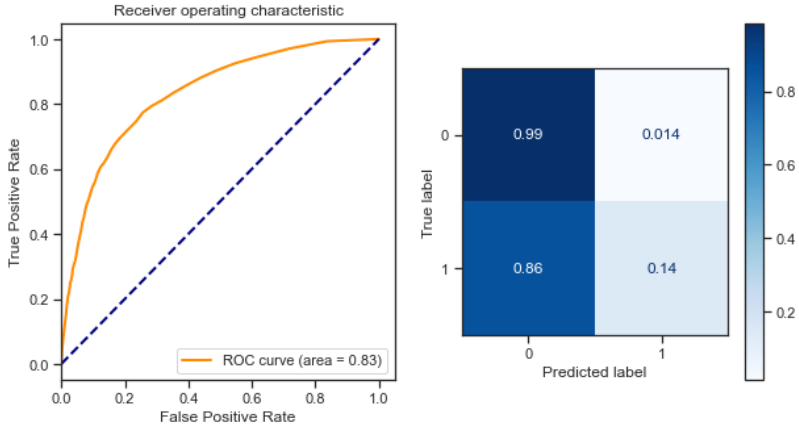
```
clas_models_grid = {'KNN_5':KNeighborsClassifier(n_neighbors=5),
                    str('KNN_' + clf_gs_best_params_txt):clf_gs.best_estimator_}
```

```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

KNN_5



KNN_100

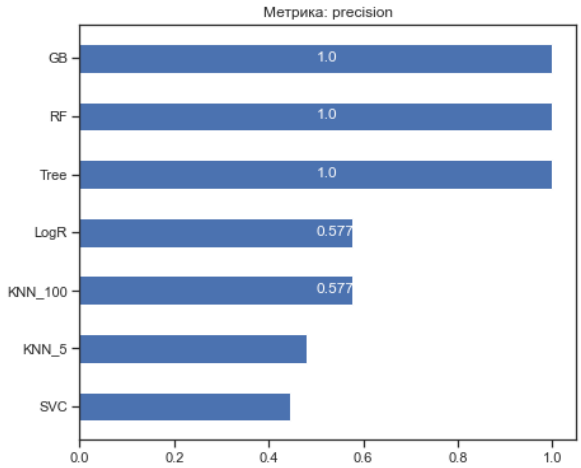


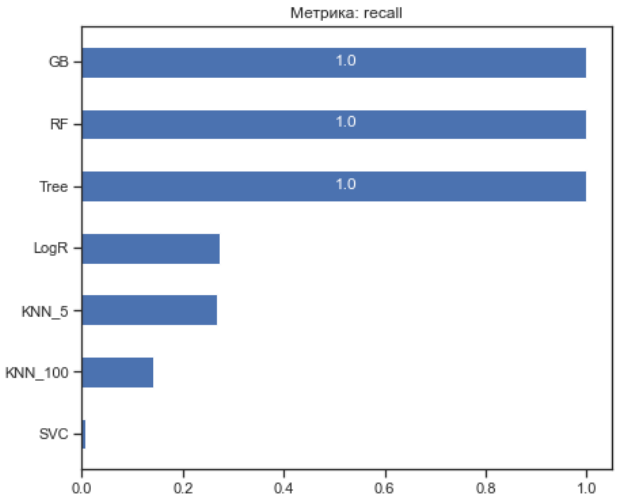
16) Формирование выводов о качестве построенных моделей на основе выбранных метрик

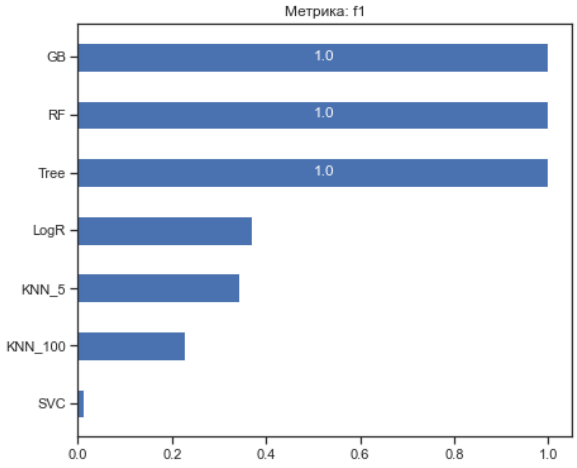
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

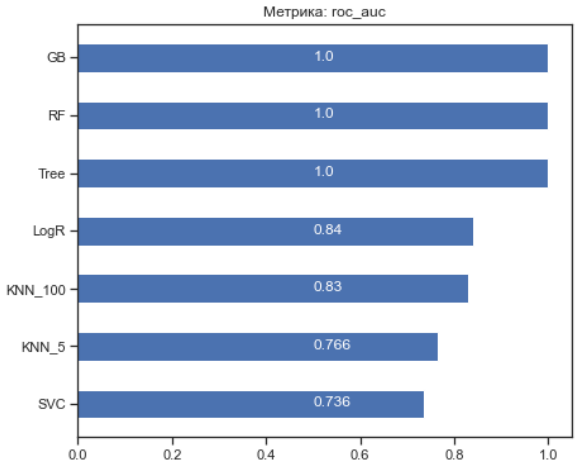
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```









17) Заключение

Три модели: градиентный бустинг, дерево и случайный лес показали одинаково высокий результат.

18) Список использованных источников информации

1. GitHub репозиторий курса «Технологии машинного обучения» 2022 год. URL: https://github.com/ugapanyuk/ml_course_2022/wiki/COURSE_TMO
2. Matplotlib URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html
3. Scikit-learn URL: https://scikit-learn.org/stable/index.html