|  |  |
| --- | --- |
|  | **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 \_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

**\_\_\_*Предсказательная модель заработной платы*\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_ИУ5-64Б\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_И. Д. Малютин**

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

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_****Ю.Е. Гапанюк\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2024 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 \_\_\_ИУ5\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_В.И. Терехов\_\_

(И.О.Фамилия)

«\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме \_\_\_\_\_\_Предсказательная модель заработной платы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_ИУ5-64Б\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Малютин Илья Дмитриевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИССЛЕДОВАТЕЛЬСКАЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_КАФЕДРА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_\_ нед., 50% к \_\_\_\_ нед., 75% к \_\_\_ нед., 100% к \_\_\_\_ нед.

***Техническое задание:*** решение задачи машинного обучения на основе материалов дисциплины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на 38 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Ю.Е. Гапанюк\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_И.Д. Малютин\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

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

# Научно-исследовательская работа по Технологиям машинного обучения на тему Предсказательная модель заработной платы.

## Поиск и выбор набора данных для построения моделей машинного обучения.

В качестве набора данных мы будем использовать набор данных для бинарной классификации людей имеющих зарплату более 50000 - <https://archive.ics.uci.edu/dataset/2/adult>

Эта задача является очень актуальной для банковской сферы в ситуациях, когда необходимо понять можно ли конкретному человеку выдать кредит или нет.

Датасет состоит из трех файлов:

* adult.data - обучающая выборка
* adult.test - тестовая выборка (50% от размера обучающей выборки)

Каждый файл содержит следующие колонки:

* age - Возраст.
* workclass - Класс трудоустроенности.
* fnlwgt - Весовой параметр.
* education - Образование.
* education-num - Уровень образование (вычисляемый относительно параметра education).
* marital-status - "Брачное" положение
* occupation - Профессия.
* relationship - Семейной положение.
* race - Расовая принадлежность
* sex - Пол
* capital-gain - Прирост капитала
* capital-loss - Потеря капитала
* hours-per-week - Количество рабочих часов в неделе
* native-country - Страна происхождения
* income - Целевой бинарный признак - >50K, <=50K

В рассматриваемом примере будем решать задачу классификации:

* Для решения **задачи классификации** в качестве целевого признака будем использовать "income". Поскольку признак содержит только значения >50K, <=50K, то это задача бинарной классификации.

### ВАЖНО!!! В первую строку файлов датасета были добавлены названия столбцов, так как в исходном датасете они отсутствуют

age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country, income

### Импорт библиотек

Импортируем библиотеки с помощью команды import. Как правило, все команды import размещают в первых ячейках ноутбука.

#Вспомогательные функции для кросс валидации  
# !/usr/bin/env python  
  
import pandas as pd  
from plotly.subplots import make\_subplots  
import plotly.graph\_objs as go  
import pprint  
from scipy import stats  
from IPython.display import display  
  
def plot\_grid\_search(clf):  
 """Plot as many graphs as parameters are in the grid search results.  
  
 Each graph has the values of each parameter in the X axis and the Score in the Y axis.  
  
 Parameters  
 ----------  
 clf: estimator object result of a GridSearchCV  
 This object contains all the information of the cross validated results for all the parameters combinations.  
 """  
 # Convert the cross validated results in a DataFrame ordered by `rank\_test\_score` and `mean\_fit\_time`.  
 # As it is frequent to have more than one combination with the same max score,  
 # the one with the least mean fit time SHALL appear first.  
 cv\_results = pd.DataFrame(clf.cv\_results\_).sort\_values(by=['rank\_test\_score', 'mean\_fit\_time'])  
  
 # Get parameters  
 parameters = cv\_results['params'][0].keys()  
  
 # Calculate the number of rows and columns necessary  
 rows = -(-len(parameters) // 2)  
 columns = min(len(parameters), 2)  
 # Create the subplot  
 fig = make\_subplots(rows=rows, cols=columns)  
 # Initialize row and column indexes  
 row = 1  
 column = 1  
  
 # For each of the parameters  
 for parameter in parameters:  
  
 # As all the graphs have the same traces, and by default all traces are shown in the legend,  
 # the description appears multiple times. Then, only show legend of the first graph.  
 if row == 1 and column == 1:  
 show\_legend = True  
 else:  
 show\_legend = False  
  
 # Mean test score  
 mean\_test\_score = cv\_results[cv\_results['rank\_test\_score'] != 1]  
 fig.add\_trace(go.Scatter(  
 name='Mean test score',  
 x=mean\_test\_score['param\_' + parameter],  
 y=mean\_test\_score['mean\_test\_score'],  
 mode='markers',  
 marker=dict(size=mean\_test\_score['mean\_fit\_time'],  
 color='SteelBlue',  
 sizeref=2. \* cv\_results['mean\_fit\_time'].max() / (40. \*\* 2),  
 sizemin=4,  
 sizemode='area'),  
 text=mean\_test\_score['params'].apply(  
 lambda x: pprint.pformat(x, width=-1).replace('{', '').replace('}', '').replace('\n', '<br />')),  
 showlegend=show\_legend),  
 row=row,  
 col=column)  
  
 # Best estimators  
 rank\_1 = cv\_results[cv\_results['rank\_test\_score'] == 1]  
 fig.add\_trace(go.Scatter(  
 name='Best estimators',  
 x=rank\_1['param\_' + parameter],  
 y=rank\_1['mean\_test\_score'],  
 mode='markers',  
 marker=dict(size=rank\_1['mean\_fit\_time'],  
 color='Crimson',  
 sizeref=2. \* cv\_results['mean\_fit\_time'].max() / (40. \*\* 2),  
 sizemin=4,  
 sizemode='area'),  
 text=rank\_1['params'].apply(str),  
 showlegend=show\_legend),  
 row=row,  
 col=column)  
  
 fig.update\_xaxes(title\_text=parameter, row=row, col=column)  
 fig.update\_yaxes(title\_text='Score', row=row, col=column)  
  
 # Check the linearity of the series  
 # Only for numeric series  
 if pd.to\_numeric(cv\_results['param\_' + parameter], errors='coerce').notnull().all():  
 x\_values = cv\_results['param\_' + parameter].sort\_values().unique().tolist()  
 r = stats.linregress(x\_values, range(0, len(x\_values))).rvalue  
 # If not so linear, then represent the data as logarithmic  
 if r < 0.86:  
 fig.update\_xaxes(type='log', row=row, col=column)  
  
 # Increment the row and column indexes  
 column += 1  
 if column > columns:  
 column = 1  
 row += 1  
  
 # Show first the best estimators  
 fig.update\_layout(legend=dict(traceorder='reversed'),  
 width=columns \* 360 + 100,  
 height=rows \* 360,  
 title='Best score: {:.6f} with {}'.format(cv\_results['mean\_test\_score'].iloc[0],  
 str(cv\_results['params'].iloc[0]).replace('{',  
 '').replace(  
 '}', '')),  
 hovermode='closest',  
 template='none')  
 fig.show()  
  
  
def table\_grid\_search(clf, all\_columns=False, all\_ranks=False, save=True):  
 """Show tables with the grid search results.  
  
 Parameters  
 ----------  
 clf: estimator object result of a GridSearchCV  
 This object contains all the information of the cross validated results for all the parameters combinations.  
  
 all\_columns: boolean, default: False  
 If true all columns are returned. If false, the following columns are dropped:  
  
 - params. As each parameter has a column with the value.  
 - std\_\*. Standard deviations.  
 - split\*. Split scores.  
  
 all\_ranks: boolean, default: False  
 If true all ranks are returned. If false, only the rows with rank equal to 1 are returned.  
  
 save: boolean, default: True  
 If true, results are saved to a CSV file.  
 """  
 # Convert the cross validated results in a DataFrame ordered by `rank\_test\_score` and `mean\_fit\_time`.  
 # As it is frequent to have more than one combination with the same max score,  
 # the one with the least mean fit time SHALL appear first.  
 cv\_results = pd.DataFrame(clf.cv\_results\_).sort\_values(by=['rank\_test\_score', 'mean\_fit\_time'])  
  
 # Reorder  
 columns = cv\_results.columns.tolist()  
 # rank\_test\_score first, mean\_test\_score second and std\_test\_score third  
 columns = columns[-1:] + columns[-3:-1] + columns[:-3]  
 cv\_results = cv\_results[columns]  
  
 if save:  
 cv\_results.to\_csv('--'.join(cv\_results['params'][0].keys()) + '.csv', index=True, index\_label='Id')  
  
 # Unless all\_columns are True, drop not wanted columns: params, std\_\* split\*  
 if not all\_columns:  
 cv\_results.drop('params', axis='columns', inplace=True)  
 cv\_results.drop(list(cv\_results.filter(regex='^std\_.\*')), axis='columns', inplace=True)  
 cv\_results.drop(list(cv\_results.filter(regex='^split.\*')), axis='columns', inplace=True)  
  
 # Unless all\_ranks are True, filter out those rows which have rank equal to one  
 if not all\_ranks:  
 cv\_results = cv\_results[cv\_results['rank\_test\_score'] == 1]  
 cv\_results.drop('rank\_test\_score', axis='columns', inplace=True)  
 cv\_results = cv\_results.style.hide\_index()  
  
 display(cv\_results)

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.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  
%matplotlib inline  
sns.set(style="ticks")

### Загрузка данных

Загрузим файлы датасета в помощью библиотеки Pandas.

Не смотря на то, что файлы имеют расширение txt они представляют собой данные в формате CSV (<https://ru.wikipedia.org/wiki/CSV>). Часто в файлах такого формата в качестве разделителей используются символы ",", ";" или табуляция. Поэтому вызывая метод read\_csv всегда стоит явно указывать разделитель данных с помощью параметра sep. Чтобы узнать какой разделитель используется в файле его рекомендуется предварительно посмотреть в любом текстовом редакторе.

# Обучающая выборка  
original\_train = pd.read\_csv(r'adult.data', delimiter=", ", engine="python")  
# Тестовая выборка  
original\_test\_1 = pd.read\_csv(r'adult.test', delimiter=", ", engine="python")

# Удалим дубликаты записей, если они присутствуют  
train = original\_train.drop\_duplicates()  
test\_1 = original\_test\_1.drop\_duplicates()

## Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

### Основные характеристики датасетов

Так как пропуски обозначены знаком "?" необходимо их заменить предварительно на null

train = train.replace(to\_replace='?', value=None)  
test\_1 = test\_1.replace('?', 'None')

# Первые 5 строк датасета  
train.head()

test\_1.head()

# Размер обучающего датасета - 32536 строк, 15 колонок  
# Размер тестового датасета - 16275 строк, 15 колонок  
train.shape, test\_1.shape,

((32537, 15), (16276, 15))

# Список колонок  
train.columns

Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',  
 'marital-status', 'occupation', 'relationship', 'race', 'sex',  
 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',  
 'income'],  
 dtype='object')

# Список колонок с типами данных  
# убедимся что типы данных одинаковы в обучающей и тестовых выборках  
train.dtypes

age int64  
workclass object  
fnlwgt int64  
education object  
education-num int64  
marital-status object  
occupation object  
relationship object  
race object  
sex object  
capital-gain int64  
capital-loss int64  
hours-per-week int64  
native-country object  
income object  
dtype: object

test\_1.dtypes

age int64  
workclass object  
fnlwgt int64  
education object  
education-num int64  
marital-status object  
occupation object  
relationship object  
race object  
sex object  
capital-gain int64  
capital-loss int64  
hours-per-week int64  
native-country object  
income object  
dtype: object

# Проверим наличие пустых значений  
train.isnull().sum()

age 0  
workclass 1836  
fnlwgt 0  
education 0  
education-num 0  
marital-status 0  
occupation 1843  
relationship 0  
race 0  
sex 0  
capital-gain 0  
capital-loss 0  
hours-per-week 0  
native-country 582  
income 0  
dtype: int64

test\_1.isnull().sum()

age 0  
workclass 0  
fnlwgt 0  
education 0  
education-num 0  
marital-status 0  
occupation 0  
relationship 0  
race 0  
sex 0  
capital-gain 0  
capital-loss 0  
hours-per-week 0  
native-country 0  
income 0  
dtype: int64

**Вывод. Представленный набор данных содержит пропуски** Заменим пропуски с использованием стратегии *most\_frequent*

train = train.apply(lambda x: x.fillna(x.value\_counts().index[0]))  
test\_1 = test\_1.apply(lambda x: x.fillna(x.value\_counts().index[0]))

train.isnull().sum()

age 0  
workclass 0  
fnlwgt 0  
education 0  
education-num 0  
marital-status 0  
occupation 0  
relationship 0  
race 0  
sex 0  
capital-gain 0  
capital-loss 0  
hours-per-week 0  
native-country 0  
income 0  
dtype: int64

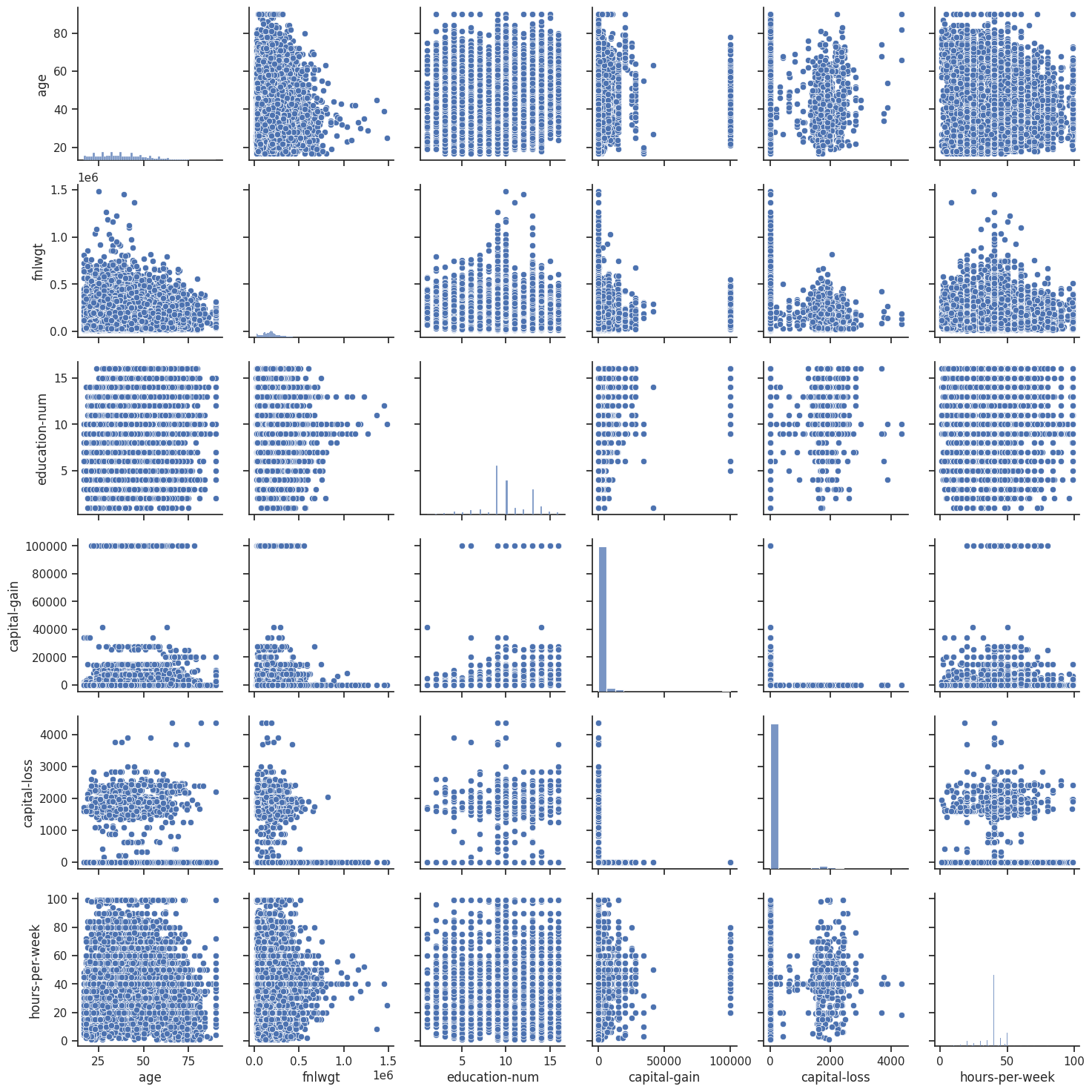
test\_1.isnull().sum()

age 0  
workclass 0  
fnlwgt 0  
education 0  
education-num 0  
marital-status 0  
occupation 0  
relationship 0  
race 0  
sex 0  
capital-gain 0  
capital-loss 0  
hours-per-week 0  
native-country 0  
income 0  
dtype: int64

### Построение графиков для понимания структуры данных

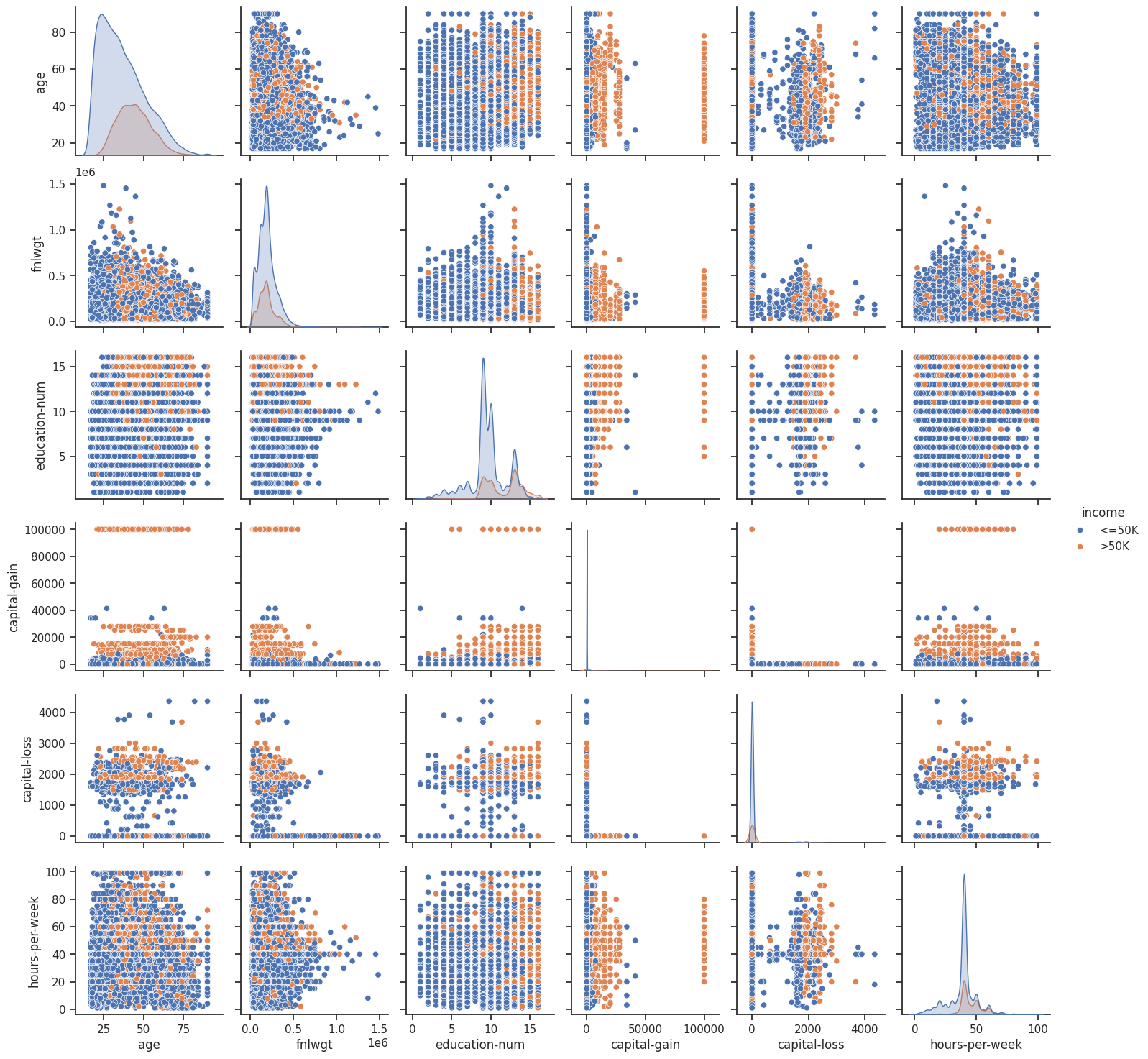
# Парные диаграммы  
sns.pairplot(train)

<seaborn.axisgrid.PairGrid at 0x7c3b1b0a7ee0>



sns.pairplot(train, hue="income")

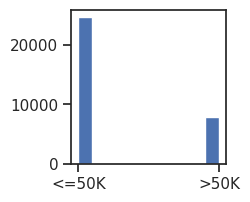
<seaborn.axisgrid.PairGrid at 0x7c3b19c90ca0>



# Убедимся, что целевой признак  
# для задачи бинарной классификации содержит только >50K и <=50K  
train['income'].unique()

array(['<=50K', '>50K'], dtype=object)

# Оценим дисбаланс классов для Occupancy  
fig, ax = plt.subplots(figsize=(2, 2))  
plt.hist(train['income'])  
plt.show()



train['income'].value\_counts()

income  
<=50K 24698  
>50K 7839  
Name: count, dtype: int64

# посчитаем дисбаланс классов  
total = train.shape[0]  
class\_0, class\_1 = train['income'].value\_counts()  
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'  
 .format(round(class\_0 / total, 4) \* 100, round(class\_1 / total, 4) \* 100))

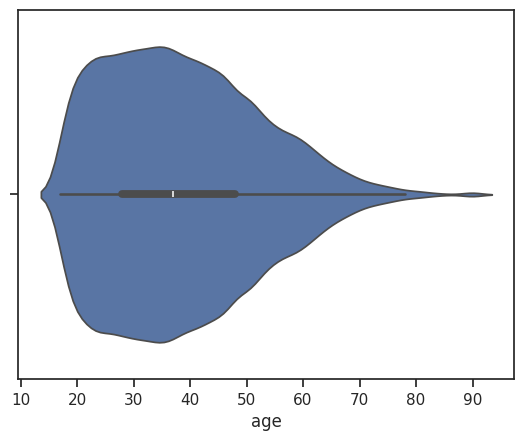
Класс 0 составляет 75.91%, а класс 1 составляет 24.09%.

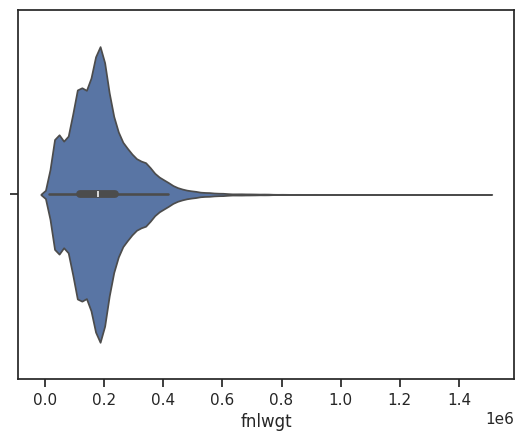
**Вывод. Дисбаланс классов присутствует, но является приемлемым.**

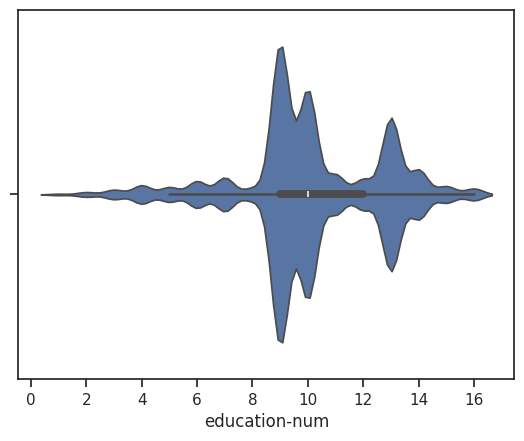
train.columns

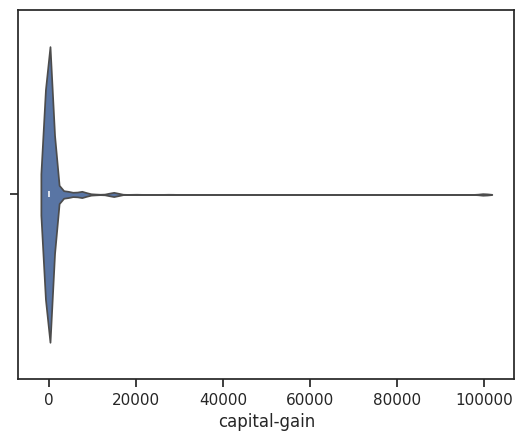
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',  
 'marital-status', 'occupation', 'relationship', 'race', 'sex',  
 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',  
 'income'],  
 dtype='object')

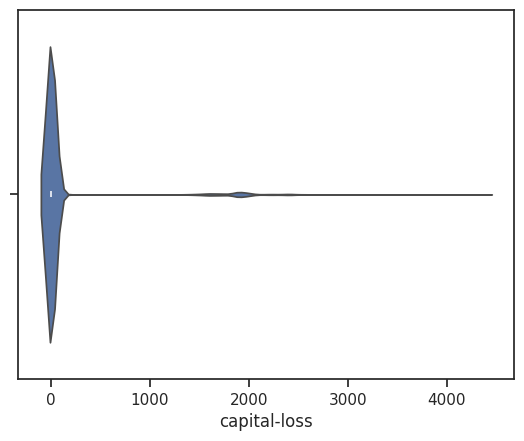
# Скрипичные диаграммы для числовых колонок  
for col in ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'income']:  
 sns.violinplot(x=train[col])  
 plt.show()

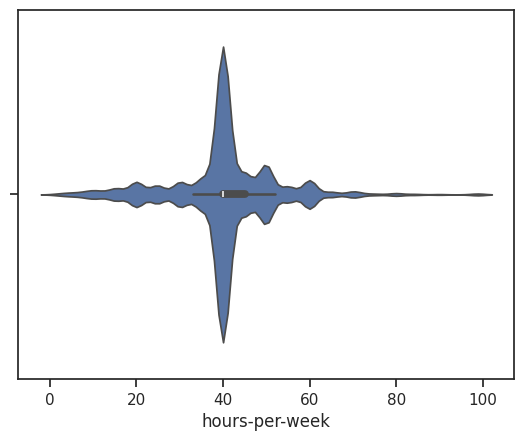


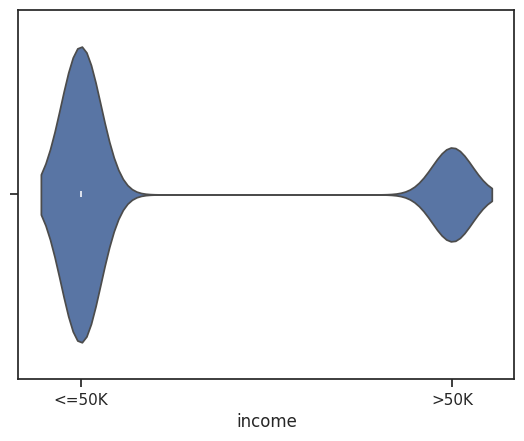












## Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

train.dtypes

age int64  
workclass object  
fnlwgt int64  
education object  
education-num int64  
marital-status object  
occupation object  
relationship object  
race object  
sex object  
capital-gain int64  
capital-loss int64  
hours-per-week int64  
native-country object  
income object  
dtype: object

Для построения моделей будем использовать все признаки.

Категориальные признаки присутствуют, необходимо их закодировать.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

Выполним масштабирование данных. Для этого необходимо объединить обучающую и тестовые выборки.

## Кодирование категориальных признаков

from sklearn.preprocessing import LabelEncoder  
  
le = LabelEncoder()  
train['workclass'] = le.fit\_transform(train['workclass'])  
train['education'] = le.fit\_transform(train['education'])  
train['marital-status'] = le.fit\_transform(train['marital-status'])  
train['occupation'] = le.fit\_transform(train['occupation'])  
train['relationship'] = le.fit\_transform(train['relationship'])  
train['race'] = le.fit\_transform(train['race'])  
train['sex'] = le.fit\_transform(train['sex'])  
train['native-country'] = le.fit\_transform(train['native-country'])  
train['income'] = le.fit\_transform(train['income'])  
  
test\_1['workclass'] = le.fit\_transform(test\_1['workclass'])  
test\_1['education'] = le.fit\_transform(test\_1['education'])  
test\_1['marital-status'] = le.fit\_transform(test\_1['marital-status'])  
test\_1['occupation'] = le.fit\_transform(test\_1['occupation'])  
test\_1['relationship'] = le.fit\_transform(test\_1['relationship'])  
test\_1['race'] = le.fit\_transform(test\_1['race'])  
test\_1['sex'] = le.fit\_transform(test\_1['sex'])  
test\_1['native-country'] = le.fit\_transform(test\_1['native-country'])  
test\_1['income'] = le.fit\_transform(test\_1['income'])  
  
train.head()

# Создадим вспомогательные колонки,  
# чтобы наборы данных можно было разделить.  
train['dataset'] = 'TRAIN'  
test\_1['dataset'] = 'TEST1'

# Колонки для объединения  
join\_cols = ['age', 'workclass', 'education', 'fnlwgt',  
 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain',  
 'capital-loss',  
 'hours-per-week', 'native-country', 'income', 'dataset']

data\_all = pd.concat([train[join\_cols], test\_1[join\_cols]])

# Проверим корректность объединения  
assert data\_all.shape[0] == train.shape[0] + test\_1.shape[0]

data\_all.head()

# Числовые колонки для масштабирования  
scale\_cols = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'income']

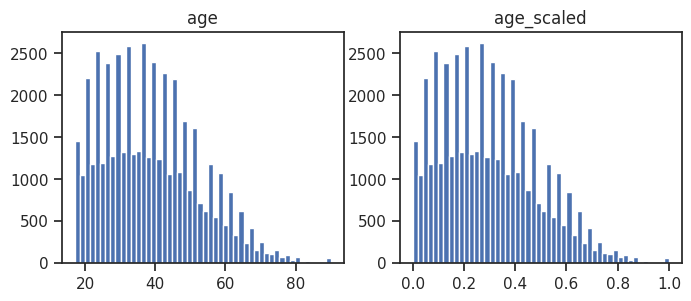
sc1 = MinMaxScaler()  
sc1\_data = sc1.fit\_transform(data\_all[scale\_cols])

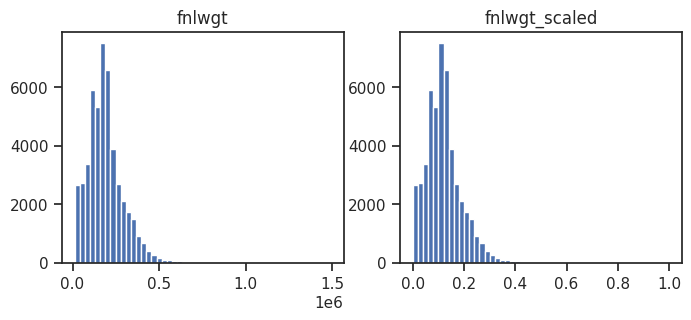
# Добавим масштабированные данные в набор данных  
for i in range(len(scale\_cols)):  
 col = scale\_cols[i]  
 new\_col\_name = col + '\_scaled'  
 data\_all[new\_col\_name] = sc1\_data[:, i]

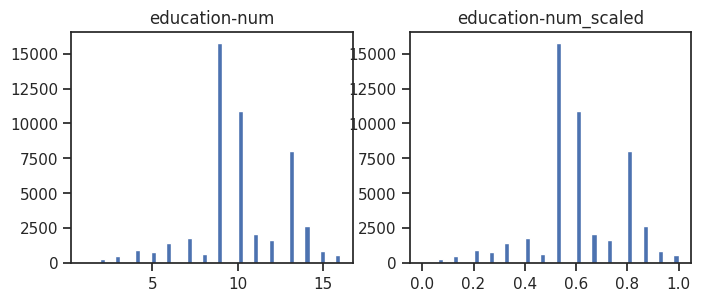
data\_all.head()

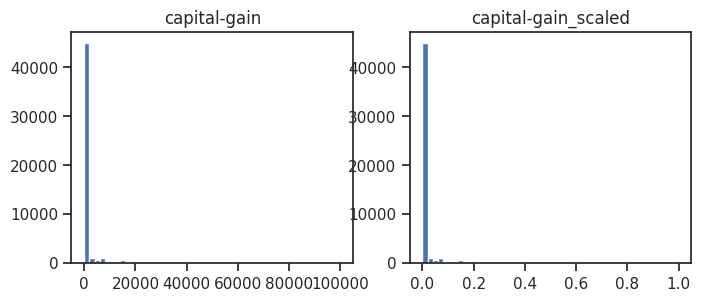
{"type":"dataframe","variable\_name":"data\_all"}

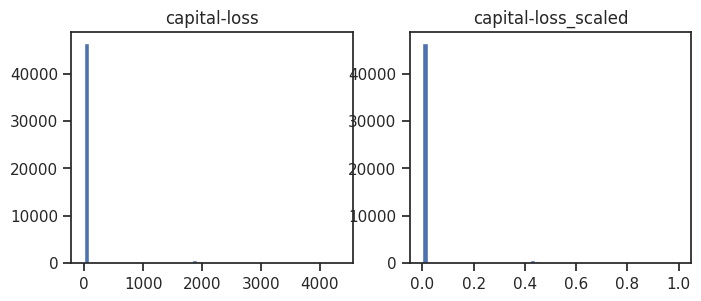
# Проверим, что масштабирование не повлияло на распределение данных  
for col in scale\_cols:  
 col\_scaled = col + '\_scaled'  
  
 fig, ax = plt.subplots(1, 2, figsize=(8, 3))  
 ax[0].hist(data\_all[col], 50)  
 ax[1].hist(data\_all[col\_scaled], 50)  
 ax[0].title.set\_text(col)  
 ax[1].title.set\_text(col\_scaled)  
 plt.show()

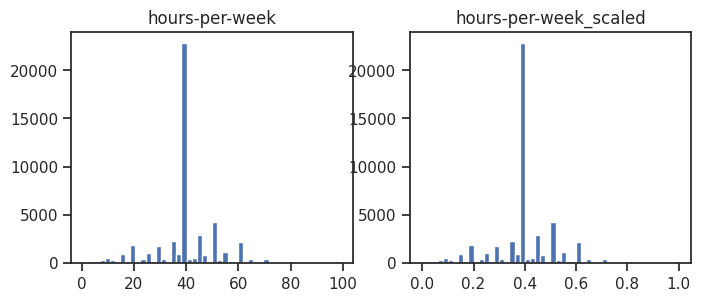


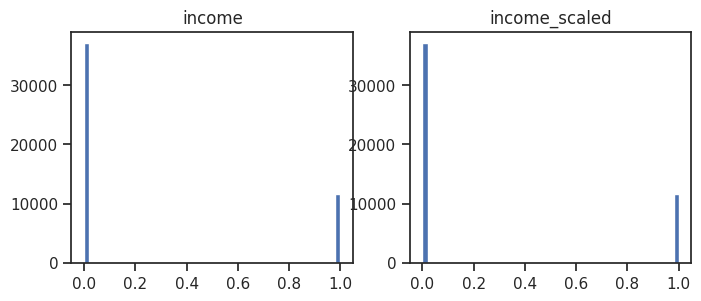












## Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

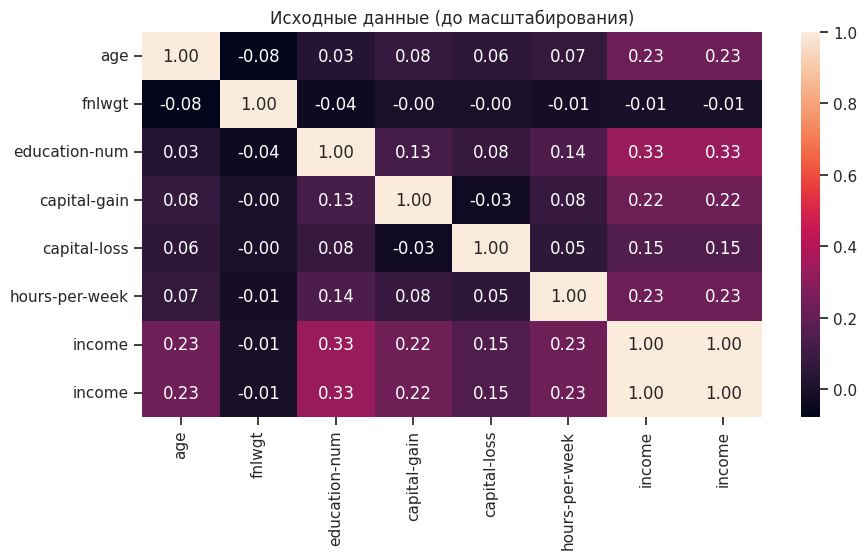
# Воспользуемся наличием тестовых выборок,  
# включив их в корреляционную матрицу  
corr\_cols\_1 = scale\_cols + ['income']  
corr\_cols\_1

['age',  
 'fnlwgt',  
 'education-num',  
 'capital-gain',  
 'capital-loss',  
 'hours-per-week',  
 'income',  
 'income']

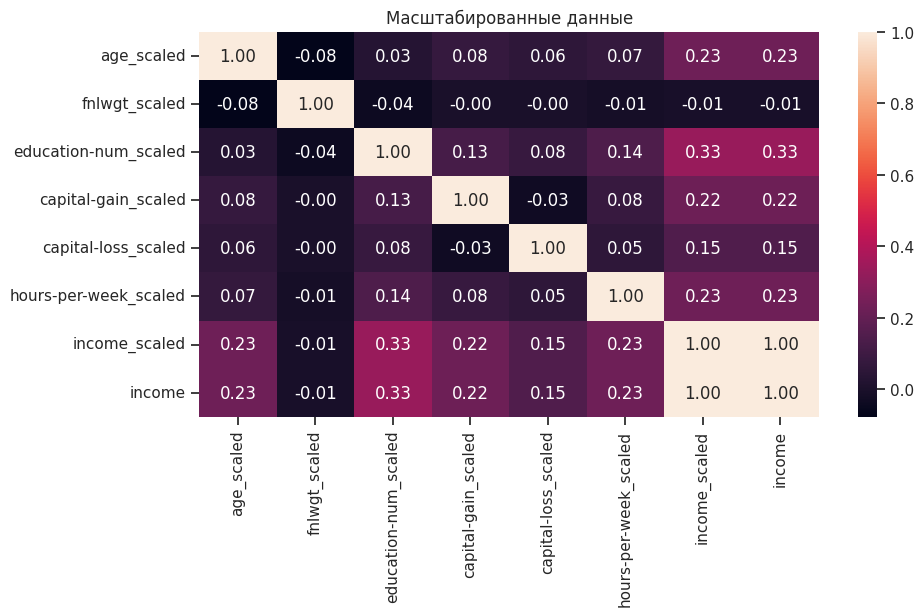
scale\_cols\_postfix = [x + '\_scaled' for x in scale\_cols]  
corr\_cols\_2 = scale\_cols\_postfix + ['income']  
corr\_cols\_2

['age\_scaled',  
 'fnlwgt\_scaled',  
 'education-num\_scaled',  
 'capital-gain\_scaled',  
 'capital-loss\_scaled',  
 'hours-per-week\_scaled',  
 'income\_scaled',  
 'income']

fig, ax = plt.subplots(figsize=(10, 5))  
sns.heatmap(data\_all[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\_all[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 "income" наиболее сильно коррелирует с Классом Образования (0.33), возрастом (0.23) и временем работы в неделю газа (0.23) и потерей/получением капитала(0,22 и 0,15 соответственно). Эти признаки обязательно следует оставить в модели классификации.
* Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

## Выбор метрик для последующей оценки качества моделей.

### В качестве метрик для решения задачи классификации будем использовать:

Метрики, формируемые на основе матрицы ошибок:

#### Метрика precision:

Можно переводить как точность, но такой перевод совпадает с переводом метрики "accuracy".

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

Используется функция [precision\_score.](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score)

#### Метрика recall (полнота):

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

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

#### Метрика -мера

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

где определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при :

Для вычисления используется функция [f1\_score.](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score)

#### Метрика ROC AUC

Основана на вычислении следующих характеристик:

- True Positive Rate, откладывается по оси ординат. Совпадает с recall.

- 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)

### 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()

## 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

## Формирование обучающей и тестовой выборок на основе исходного набора данных.

# На основе масштабированных данных выделим  
# обучающую и тестовую выборки с помощью фильтра  
train\_data\_all = data\_all[data\_all['dataset'] == 'TRAIN']  
test\_data\_all = data\_all[data\_all['dataset'] == 'TEST1']  
train\_data\_all.shape, test\_data\_all.shape

((32537, 23), (16276, 23))

#Вспомогательная ячейка  
data\_all.head()

{"type":"dataframe","variable\_name":"data\_all"}

# Признаки для задачи классификации  
task\_clas\_cols = ['age\_scaled', 'workclass', 'education', 'fnlwgt\_scaled',  
 'education-num\_scaled', 'marital-status', 'occupation', 'relationship', 'race', 'sex',  
 'capital-gain\_scaled', 'capital-loss\_scaled',  
 'hours-per-week\_scaled', 'native-country']

# Выборки для задачи классификации  
clas\_X\_train = train\_data\_all[task\_clas\_cols]  
clas\_X\_test = test\_data\_all[task\_clas\_cols]  
clas\_Y\_train = train\_data\_all['income']  
clas\_Y\_test = test\_data\_all['income']  
clas\_X\_train.shape, clas\_X\_test.shape, clas\_Y\_train.shape, clas\_Y\_test.shape

((32537, 14), (16276, 14), (32537,), (16276,))

## 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.

### Решение задачи классификации

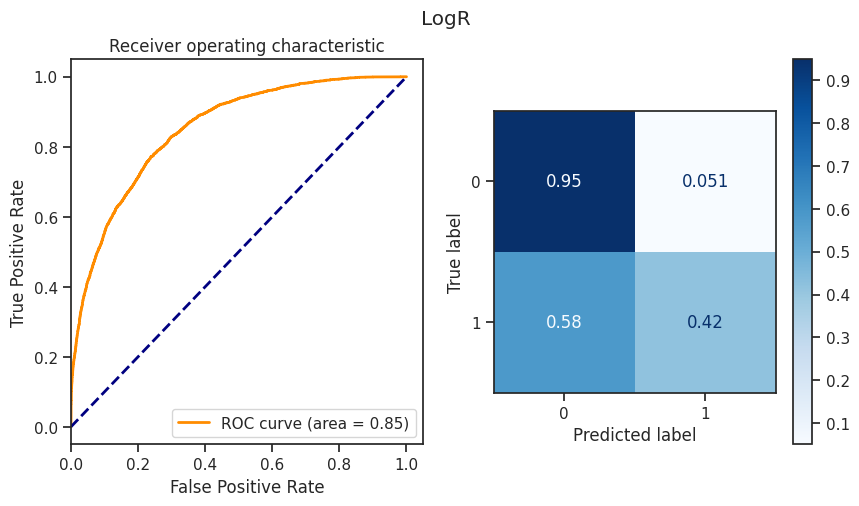
# Модели  
clas\_models = {'LogR': LogisticRegression(max\_iter=1000),  
 'KNN\_5': KNeighborsClassifier(n\_neighbors=5),  
 'SVC': SVC(probability=True),  
 'Tree': DecisionTreeClassifier(),  
 'RF': RandomForestClassifier(),  
 'GB': GradientBoostingClassifier()}

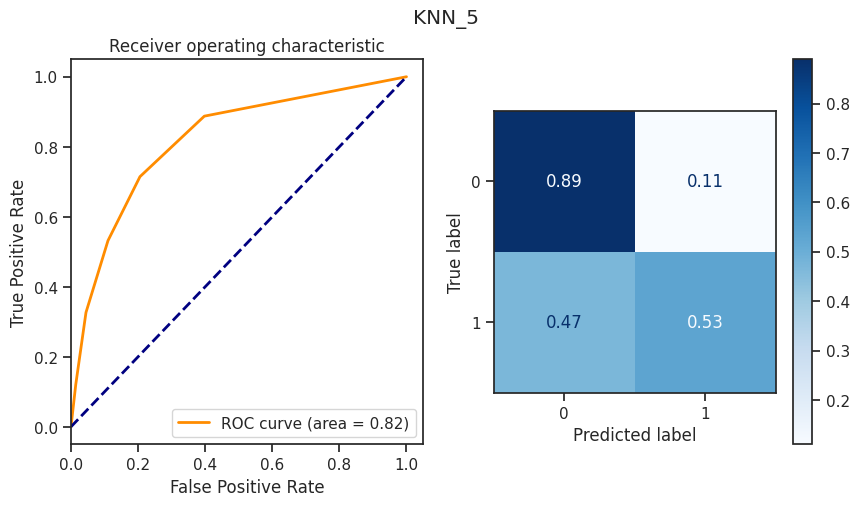
# Сохранение метрик  
clasMetricLogger = MetricLogger()

# 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")

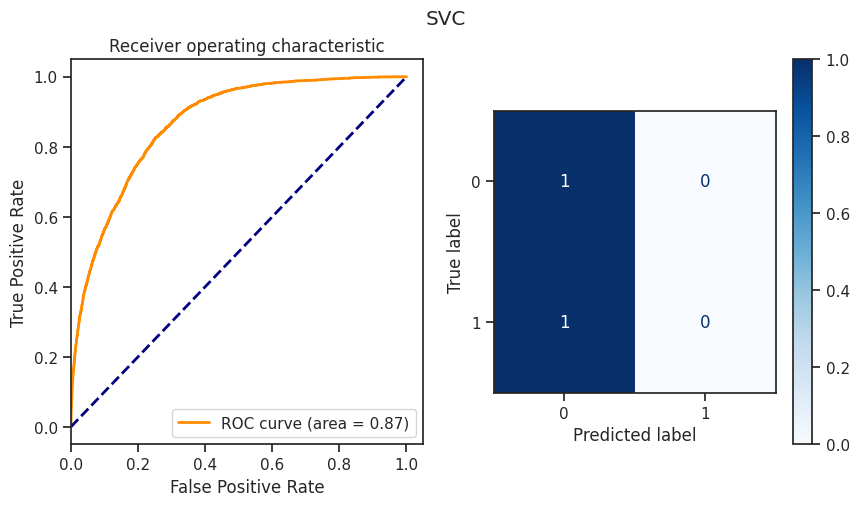
from sklearn.metrics import ConfusionMatrixDisplay  
  
  
def clas\_train\_model(model\_name, model, clasMetricLogger):  
 model.fit(clas\_X\_train, clas\_Y\_train)  
 # Предсказание значений  
 Y\_pred = model.predict(clas\_X\_test)  
 # Предсказание вероятности класса "1" для roc auc  
 Y\_pred\_proba\_temp = model.predict\_proba(clas\_X\_test)  
 Y\_pred\_proba = Y\_pred\_proba\_temp[:, 1]  
  
 precision = precision\_score(clas\_Y\_test.values, Y\_pred)  
 recall = recall\_score(clas\_Y\_test.values, Y\_pred)  
 f1 = f1\_score(clas\_Y\_test.values, Y\_pred)  
 roc\_auc = roc\_auc\_score(clas\_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(clas\_Y\_test.values, Y\_pred\_proba, ax[0])  
 ConfusionMatrixDisplay.from\_estimator(model, clas\_X\_test, clas\_Y\_test.values, ax=ax[1],  
 display\_labels=['0', '1'],  
 cmap=plt.cm.Blues, normalize='true')  
 fig.suptitle(model\_name)  
 plt.show()

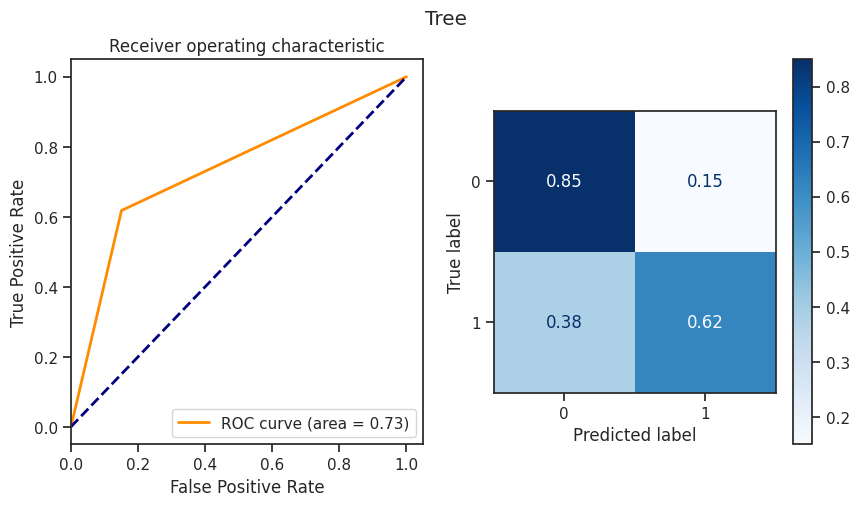
for model\_name, model in clas\_models.items():  
 clas\_train\_model(model\_name, model, clasMetricLogger)

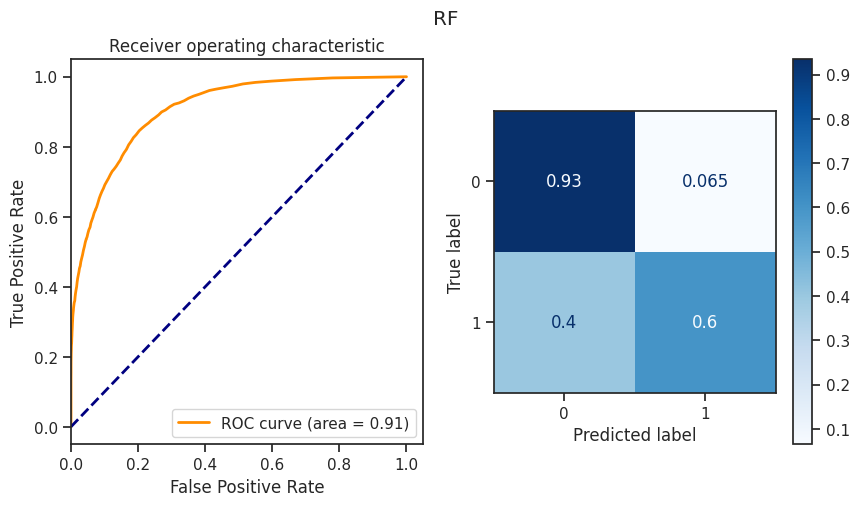


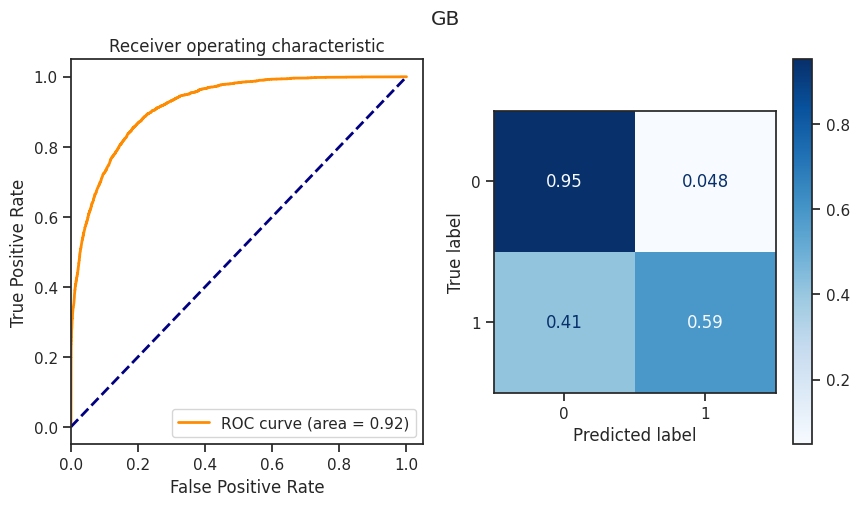


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))









## Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.

### Пример для задачи классификации

clas\_X\_train.shape

(32537, 14)

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(clas\_X\_train, clas\_Y\_train)

CPU times: user 9min 2s, sys: 10.8 s, total: 9min 13s  
Wall time: 9min 16s

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=50)

# Лучшее значение параметров  
clf\_gs.best\_params\_

{'n\_neighbors': 50}

clf\_gs\_best\_params\_txt = str(clf\_gs.best\_params\_['n\_neighbors'])  
clf\_gs\_best\_params\_txt

{"type":"string"}

n\_range\_list = list(range(20, 50, 1))  
n\_range\_list[0] = 1  
n\_range = np.array(n\_range\_list)  
tuned\_parameters = [{'n\_neighbors': n\_range}]

%%time  
clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring='roc\_auc')  
clf\_gs.fit(clas\_X\_train, clas\_Y\_train)

CPU times: user 2min 53s, sys: 218 ms, total: 2min 53s  
Wall time: 2min 55s

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid=[{'n\_neighbors': array([ 1, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])}],  
 scoring='roc\_auc')

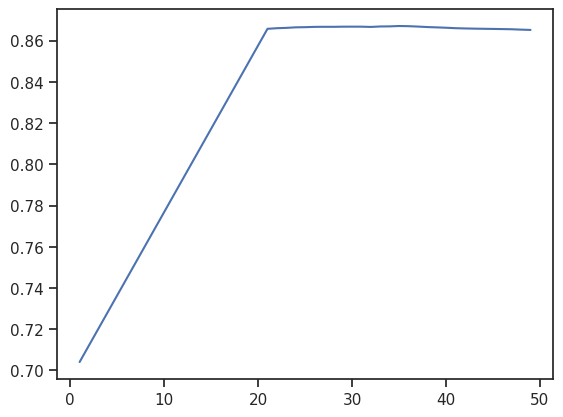
clf\_gs.best\_estimator\_

KNeighborsClassifier(n\_neighbors=35)

clf\_gs.best\_params\_  
clf\_gs\_best\_params\_txt = str(clf\_gs.best\_params\_['n\_neighbors'])

# Изменение качества на тестовой выборке в зависимости от К-соседей  
plt.plot(n\_range, clf\_gs.cv\_results\_['mean\_test\_score'])

[<matplotlib.lines.Line2D at 0x7c3b0f9c0310>]



# Подбор Гиперпараметра для модели Логистической Регрессии(алгоритм)

#  
solver = ['liblinear', 'newton-cg', 'sag', 'saga', 'lbfgs']  
tuned\_parameters = [{'solver': solver}]  
tuned\_parameters

[{'solver': ['liblinear', 'newton-cg', 'sag', 'saga', 'lbfgs']}]

# from grid\_search\_utils import plot\_grid\_search, table\_grid\_search  
  
clf\_lr = GridSearchCV(LogisticRegression(max\_iter=5000), tuned\_parameters, cv=5, scoring='roc\_auc')  
clf\_lr.fit(clas\_X\_train, clas\_Y\_train)  
plot\_grid\_search(clf\_lr)  
table\_grid\_search(clf\_lr, all\_ranks=True)

clf\_lr.best\_estimator\_

LogisticRegression(max\_iter=5000, solver='newton-cg')

clf\_lr.best\_params\_

{'solver': 'newton-cg'}

# Подбор Гиперпараметра для модели Градиентного Спуска

learning\_rate = {'learning\_rate': [0.01, 0.02, 0.05, 0.1, 0.2, 0.25, 0.30, 0.35, 0.4, 0.45, 0.5]}

clf\_gb = GridSearchCV(GradientBoostingClassifier(), learning\_rate, cv=5, scoring='roc\_auc')  
clf\_gb.fit(clas\_X\_train, clas\_Y\_train)  
plot\_grid\_search(clf\_gb)  
table\_grid\_search(clf\_gb, all\_ranks=True)

# Лучшая модель  
clf\_gb.best\_estimator\_

GradientBoostingClassifier(learning\_rate=0.3)

# Лучшее значение параметров  
clf\_gb.best\_params\_

{'learning\_rate': 0.3}

clf\_gb\_best\_params\_txt = str(clf\_gb.best\_params\_['learning\_rate'])  
clf\_gb\_best\_params\_txt

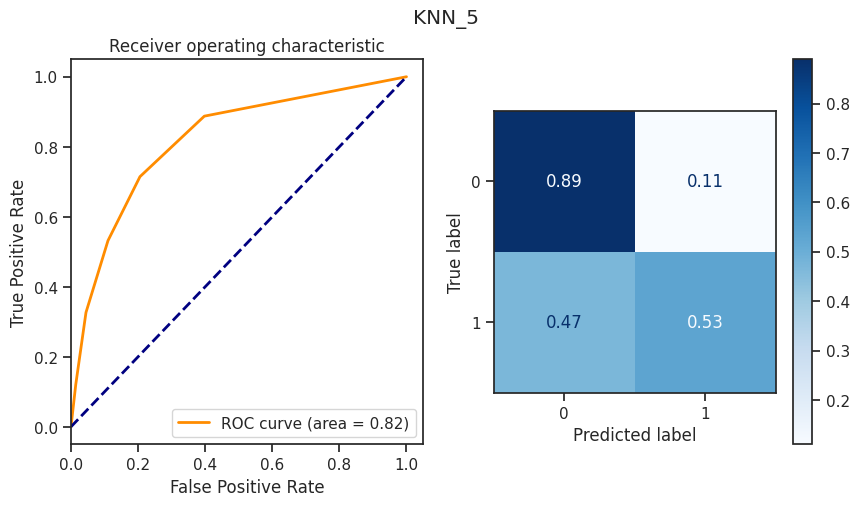
{"type":"string"}

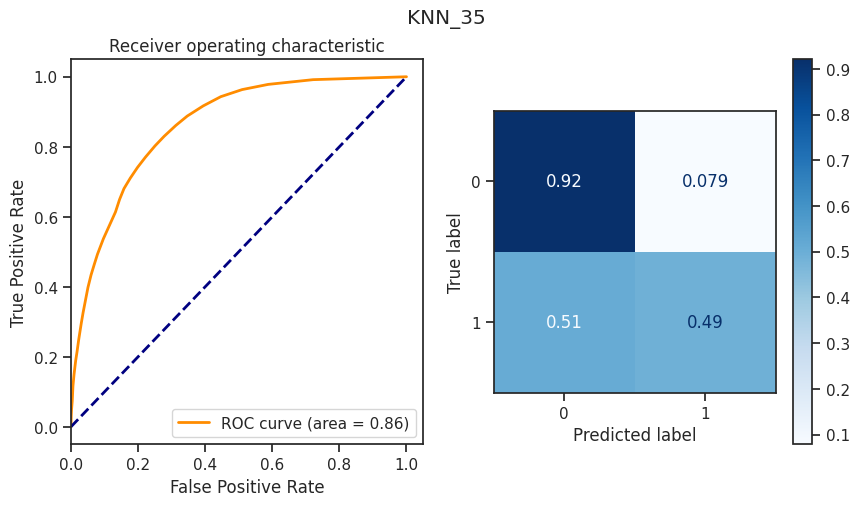
## Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

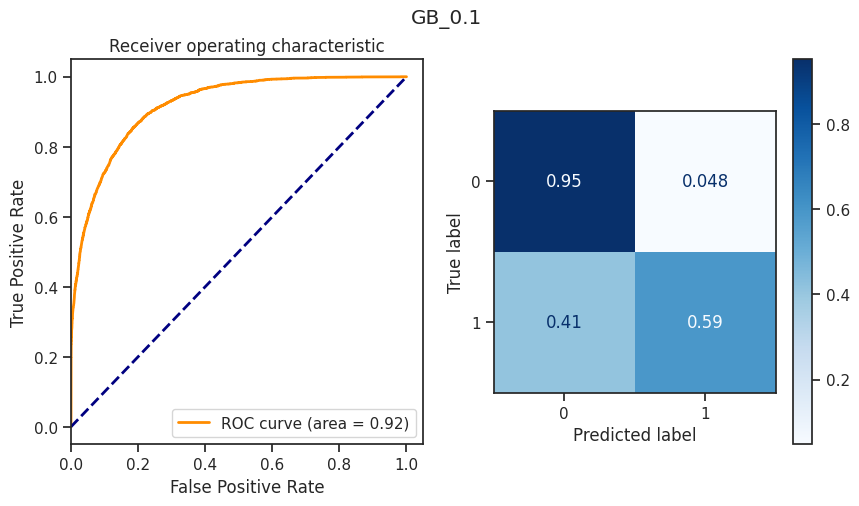
### Решение задачи классификации

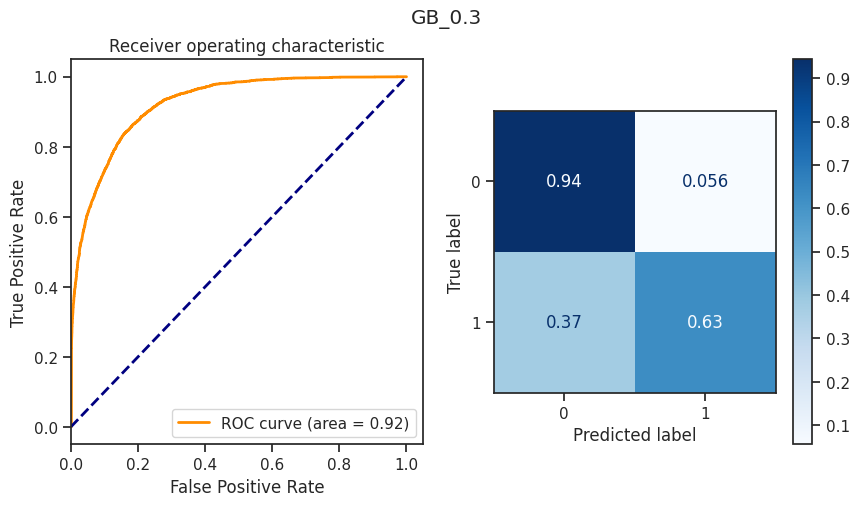
clas\_models\_grid = {'KNN\_5': KNeighborsClassifier(n\_neighbors=5),  
 str('KNN\_' + clf\_gs\_best\_params\_txt): clf\_gs.best\_estimator\_,  
 'GB\_0.1': GradientBoostingClassifier(),  
 str('GB\_' + clf\_gb\_best\_params\_txt): clf\_gb.best\_estimator\_,  
 }

for model\_name, model in clas\_models\_grid.items():  
 clas\_train\_model(model\_name, model, clasMetricLogger)









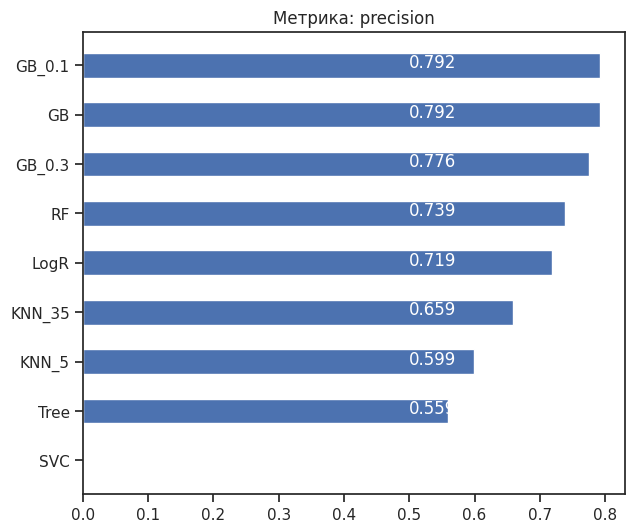
## Формирование выводов о качестве построенных моделей на основе выбранных метрик.

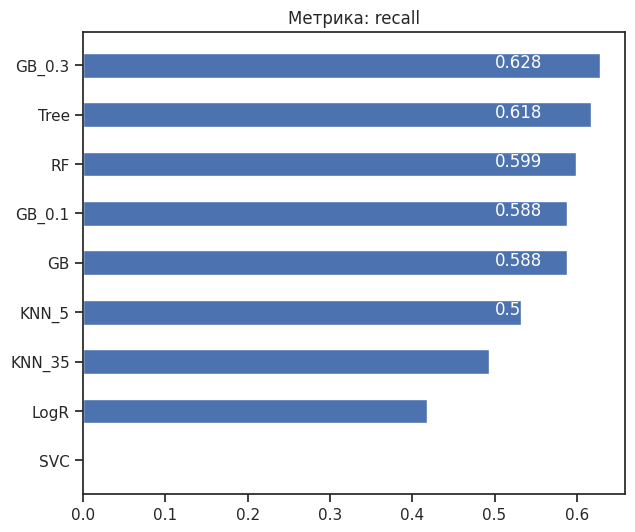
### Решение задачи классификации

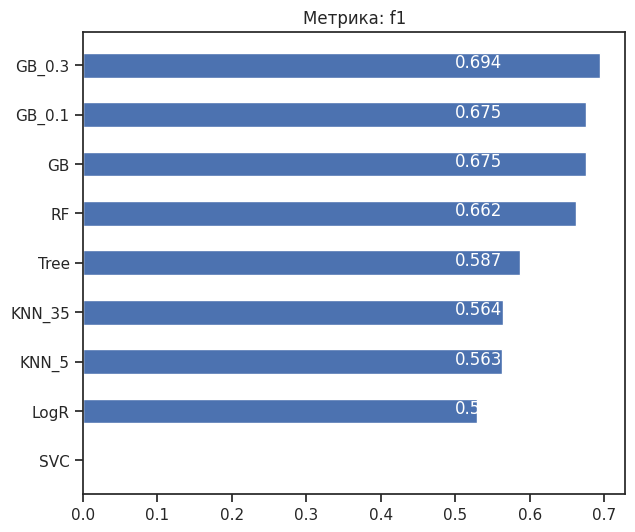
# Метрики качества модели  
clas\_metrics = clasMetricLogger.df['metric'].unique()  
clas\_metrics

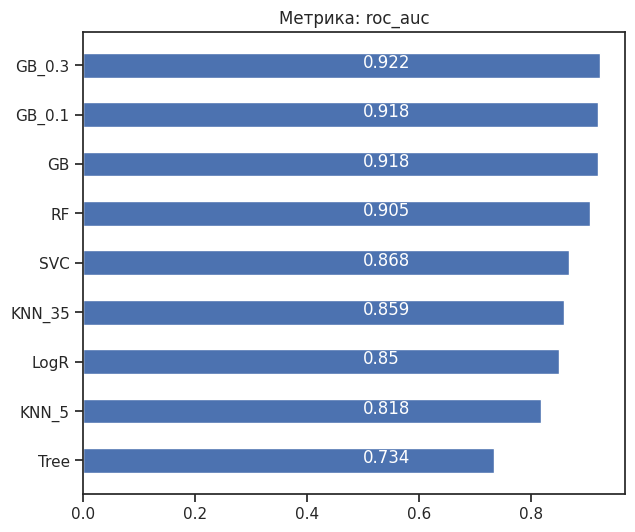
array(['precision', 'recall', 'f1', 'roc\_auc'], dtype=object)

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









**Вывод: на основании трех метрик из четырех используемых, лучшей оказалась модель градиентного спуска c оптимальными параметрами.**