********

**СОДЕРЖАНИЕ**

[1. Введение 4](#_Toc168739724)

[2. Постановка задачи 5](#_Toc168739725)

[3. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии. 5](#_Toc168739726)

[4. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных. 7](#_Toc168739727)

[5. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей. 9](#_Toc168739728)

[6. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен. 12](#_Toc168739729)

[7. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор. 13](#_Toc168739730)

[8. Формирование обучающей и тестовой выборок на основе исходного набора данных. 13](#_Toc168739731)

[9. 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. 14](#_Toc168739732)

[10. 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. 19](#_Toc168739733)

[11. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей. 21](#_Toc168739734)

[12. Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, вияния значений гиперпарметров на качество моделей и т.д. 22](#_Toc168739735)

[13. Заключение 25](#_Toc168739736)

[14. Список литературы 26](#_Toc168739737)

# Введение

В качестве предметной области был выбран набор данных, содержащий данные об грибах и их ядовитости.

Задача данной работы - предсказание съедобный ли гриб на основе нескольких факторов. Решение этой задачи может быть актуальным для любителей тихой охоты, желающих безопасно собирать грибы. Также оно может быть полезным для разработчиков приложений для идентификации грибов, способных предупредить пользователей о потенциальной опасности. Наконец, данная задача может быть интересна для научных исследований, направленных на изучение взаимосвязи между внешними признаками грибов и их токсичностью.

# Постановка задачи

Необходимо решить задачу классификации по прогнозированию к какому классу относится гриб – съедобный или несъедобный, с использованием материалов дисциплины «Технологии машинного обучения».

# Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.

В качестве набора данных мы будем использовать набор данных : Этот набор данных о грибах для двоичной классификации. Он содержит 9 столбцов:

1. Cap Diameter - Диаметр шляпки
2. Cap Shape - Форма шляпки
3. Gill Attachment - Прикрепление пластинки
4. Gill Color - Цвет пластинки
5. Stem Height - Высота ножки
6. Stem Width - Ширина ножки
7. Stem Color - Цвет ножки
8. Season - Время года
9. Class - Целевой класс: съедобно или нет?

Целевой класс содержит два значения — 0 или 1, где 0 относится к съедобному, а 1 — к ядовитому.

import pandas as pd  
  
df = pd.read\_csv("/content/mushroom\_cleaned.csv")  
df

{"summary":"{\n \"name\": \"df\",\n \"rows\": 54035,\n \"fields\": [\n {\n \"column\": \"cap-diameter\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 359,\n \"min\": 0,\n \"max\": 1891,\n \"num\_unique\_values\": 1847,\n \"samples\": [\n 11,\n 1677,\n 44\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-shape\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 6,\n 5\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-attachment\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 0,\n 4\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 11,\n \"num\_unique\_values\": 12,\n \"samples\": [\n 4,\n 6,\n 10\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-height\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.6509686159221912,\n \"min\": 0.0004256969608359,\n \"max\": 3.8353196773222433,\n \"num\_unique\_values\": 1455,\n \"samples\": [\n 0.004404685938308,\n 1.826781637620518,\n 1.396199434131862\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-width\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 782,\n \"min\": 0,\n \"max\": 3569,\n \"num\_unique\_values\": 3510,\n \"samples\": [\n 3028,\n 1505,\n 3516\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 12,\n \"num\_unique\_values\": 13,\n \"samples\": [\n 7,\n 4,\n 11\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"season\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.3055935859378124,\n \"min\": 0.0273721330556057,\n \"max\": 1.804272708628173,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0.9431945538974952,\n 0.0273721330556057,\n 1.804272708628173\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"class\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n \"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe","variable\_name":"df"}

df.head()

{"summary":"{\n \"name\": \"df\",\n \"rows\": 54035,\n \"fields\": [\n {\n \"column\": \"cap-diameter\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 359,\n \"min\": 0,\n \"max\": 1891,\n \"num\_unique\_values\": 1847,\n \"samples\": [\n 11,\n 1677,\n 44\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-shape\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 6,\n 5\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-attachment\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 0,\n 4\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 11,\n \"num\_unique\_values\": 12,\n \"samples\": [\n 4,\n 6,\n 10\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-height\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.6509686159221912,\n \"min\": 0.0004256969608359,\n \"max\": 3.8353196773222433,\n \"num\_unique\_values\": 1455,\n \"samples\": [\n 0.004404685938308,\n 1.826781637620518,\n 1.396199434131862\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-width\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 782,\n \"min\": 0,\n \"max\": 3569,\n \"num\_unique\_values\": 3510,\n \"samples\": [\n 3028,\n 1505,\n 3516\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 12,\n \"num\_unique\_values\": 13,\n \"samples\": [\n 7,\n 4,\n 11\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"season\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.3055935859378124,\n \"min\": 0.0273721330556057,\n \"max\": 1.804272708628173,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0.9431945538974952,\n 0.0273721330556057,\n 1.804272708628173\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"class\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n \"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe","variable\_name":"df"}

df.columns

Index(['cap-diameter', 'cap-shape', 'gill-attachment', 'gill-color',  
 'stem-height', 'stem-width', 'stem-color', 'season', 'class'],  
 dtype='object')

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

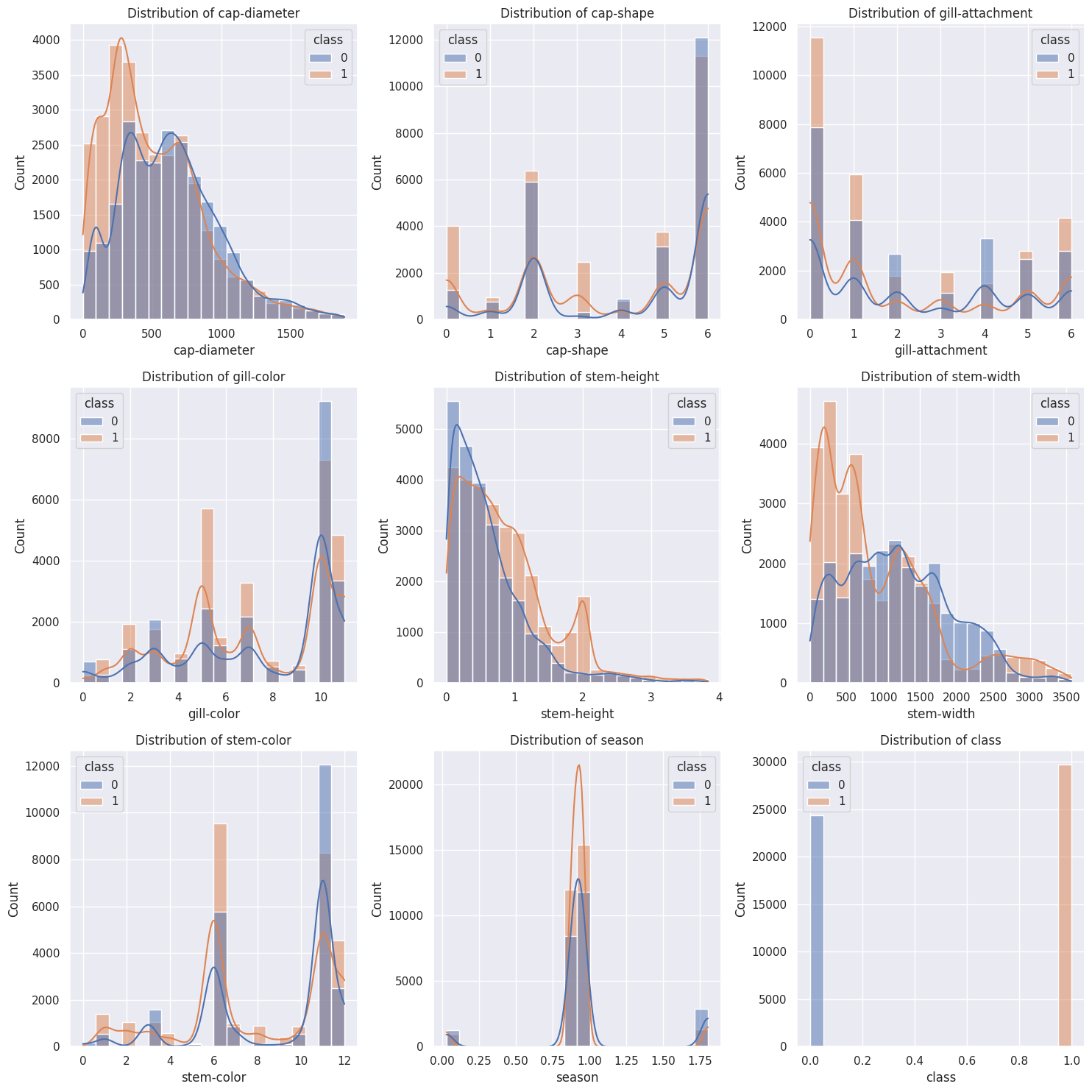
df.isnull().sum()

cap-diameter 0  
cap-shape 0  
gill-attachment 0  
gill-color 0  
stem-height 0  
stem-width 0  
stem-color 0  
season 0  
class 0  
dtype: int64

Представленный набор данных не содержит пропусков

import numpy as np  
  
import seaborn as sns  
import matplotlib.pylab as plt  
%matplotlib inline  
  
from scipy import stats

sns.set()  
  
# Create subplots with 3 columns and 3 rows  
fig, axes = plt.subplots(3, 3, figsize=(15, 15))  
  
# Flatten the axes array for easy iteration  
axes = axes.flatten()  
  
# Iterate over each column in the DataFrame  
for i, column in enumerate(df.columns):  
 # Create a histogram plot for the current column with hue  
 sns.histplot(data=df, x=column, hue='class', kde=True, bins=20, ax=axes[i])  
  
 # Set title for the plot  
 axes[i].set\_title(f'Distribution of {column}')  
  
# Adjust layout to prevent overlap of titles  
plt.tight\_layout()  
  
# Display the plot  
plt.show()



sns.pairplot(df)

<seaborn.axisgrid.PairGrid at 0x7853c9e01960>



df['class'].value\_counts()

class  
1 29675  
0 24360  
Name: count, dtype: int64

Дисбаланса классов нет. Кодирование категориальных признаков производить не нужно, так как они уже закодированы числами

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

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()  
data = df.copy()  
  
x = df.drop(columns=['class'])  
scale\_cols = list(x.columns)

sc\_data = sc.fit\_transform(df[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] = sc\_data[:,i]  
  
data.head()

{"summary":"{\n \"name\": \"data\",\n \"rows\": 54035,\n \"fields\": [\n {\n \"column\": \"cap-diameter\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 359,\n \"min\": 0,\n \"max\": 1891,\n \"num\_unique\_values\": 1847,\n \"samples\": [\n 11,\n 1677,\n 44\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-shape\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 6,\n 5\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-attachment\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2,\n \"min\": 0,\n \"max\": 6,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 2,\n 0,\n 4\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 11,\n \"num\_unique\_values\": 12,\n \"samples\": [\n 4,\n 6,\n 10\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-height\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.6509686159221912,\n \"min\": 0.0004256969608359,\n \"max\": 3.8353196773222433,\n \"num\_unique\_values\": 1455,\n \"samples\": [\n 0.004404685938308,\n 1.826781637620518,\n 1.396199434131862\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-width\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 782,\n \"min\": 0,\n \"max\": 3569,\n \"num\_unique\_values\": 3510,\n \"samples\": [\n 3028,\n 1505,\n 3516\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-color\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 3,\n \"min\": 0,\n \"max\": 12,\n \"num\_unique\_values\": 13,\n \"samples\": [\n 7,\n 4,\n 11\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"season\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.3055935859378124,\n \"min\": 0.0273721330556057,\n \"max\": 1.804272708628173,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0.9431945538974952,\n 0.0273721330556057,\n 1.804272708628173\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"class\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n \"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-diameter\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.19031399433756777,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 1847,\n \"samples\": [\n 0.005817028027498678,\n 0.886832363828662\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-shape\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.36008421656155676,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 0.3333333333333333,\n 1.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-attachment\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.3714701341477766,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 0.3333333333333333,\n 0.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-color\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.29093325883510085,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 12,\n \"samples\": [\n 0.36363636363636365,\n 0.5454545454545454\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-height\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.1697487907764383,\n \"min\": 0.0,\n \"max\": 1.0000000000000002,\n \"num\_unique\_values\": 1455,\n \"samples\": [\n 0.0010375747016341539,\n 0.4762467880500736\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-width\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.2191247060781074,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 3510,\n \"samples\": [\n 0.8484169235079855,\n 0.42168674698795183\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-color\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.27183981359849757,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 13,\n \"samples\": [\n 0.5833333333333333,\n 0.3333333333333333\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"season\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.1719812521526938,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0.5154044257916827,\n 0.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe","variable\_name":"data"}

corr\_cols\_1 = list(df.columns)  
corr\_cols\_1

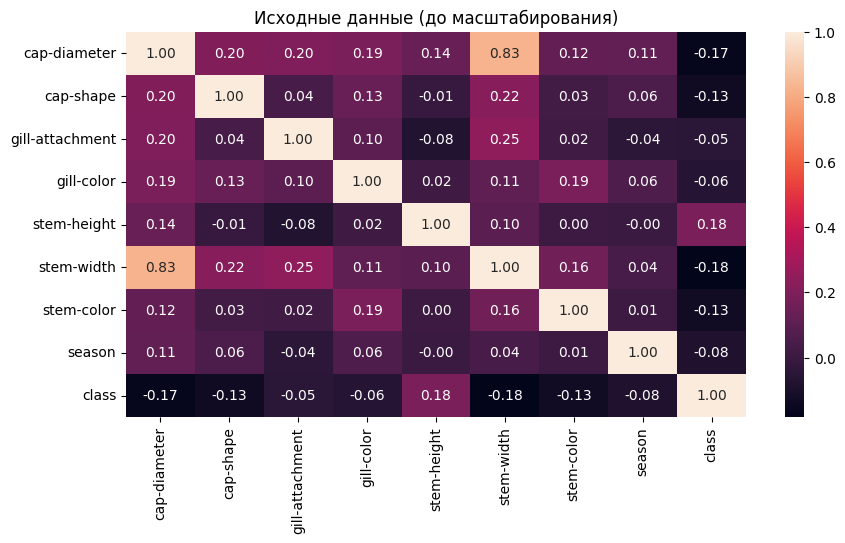
['cap-diameter',  
 'cap-shape',  
 'gill-attachment',  
 'gill-color',  
 'stem-height',  
 'stem-width',  
 'stem-color',  
 'season',  
 'class']

corr\_cols\_2 = [y+'\_scaled' for y in x.columns] + ['class']  
corr\_cols\_2

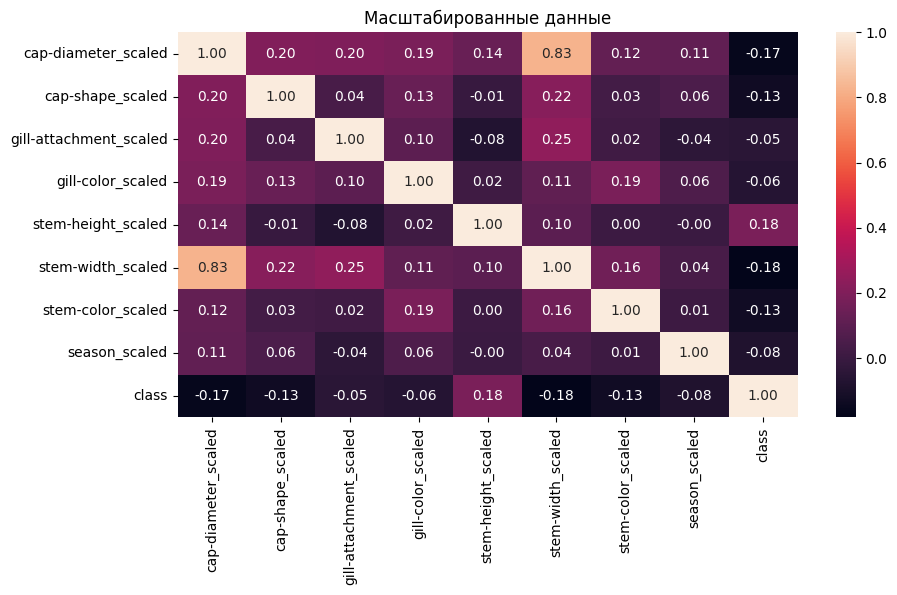
['cap-diameter\_scaled',  
 'cap-shape\_scaled',  
 'gill-attachment\_scaled',  
 'gill-color\_scaled',  
 'stem-height\_scaled',  
 'stem-width\_scaled',  
 'stem-color\_scaled',  
 'season\_scaled',  
 'class']

# Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.

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:

# Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.

Так как наша цель определить относится ли гриб к ядовитым, то у разных ошибок - разный вес. не так важно пропустить съедобный гриб и отнести его к ядовитым, нежели отнести ядовитый к съедобным. Получаем, что метрика accuracy нам не подходит, так как не учитывает вес ошибки. Возьмём метрику recall, так как она учитывает количество False Negative(на ядовитый гриб моделька выдала результат съедобный). Также можем взять метрику precision, чтобы увидеть долю истинных срабатываний от общего числа срабатываний.

**Выбор наиболее подходящих моделей для решения задачи классификации**

Для задачи классификации будем использовать следующие модели:

1. Логистическая регрессия
2. Метод ближайших соседей
3. Решающее дерево
4. Случайный лес
5. Градиентный бустинг

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

from sklearn.model\_selection import train\_test\_split  
  
y = data['class']  
x = data[[y+'\_scaled' for y in x.columns]]  
x

{"summary":"{\n \"name\": \"x\",\n \"rows\": 54035,\n \"fields\": [\n {\n \"column\": \"cap-diameter\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.19031399433756777,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 1847,\n \"samples\": [\n 0.005817028027498678,\n 0.886832363828662,\n 0.023268112109994712\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"cap-shape\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.36008421656155676,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 0.3333333333333333,\n 1.0,\n 0.8333333333333333\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-attachment\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.3714701341477766,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 0.3333333333333333,\n 0.0,\n 0.6666666666666666\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"gill-color\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.29093325883510085,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 12,\n \"samples\": [\n 0.36363636363636365,\n 0.5454545454545454,\n 0.9090909090909092\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-height\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.1697487907764383,\n \"min\": 0.0,\n \"max\": 1.0000000000000002,\n \"num\_unique\_values\": 1455,\n \"samples\": [\n 0.0010375747016341539,\n 0.4762467880500736,\n 0.3639667078982679\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-width\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.2191247060781074,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 3510,\n \"samples\": [\n 0.8484169235079855,\n 0.42168674698795183,\n 0.9851499019333148\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"stem-color\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.27183981359849757,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 13,\n \"samples\": [\n 0.5833333333333333,\n 0.3333333333333333,\n 0.9166666666666666\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"season\_scaled\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.1719812521526938,\n \"min\": 0.0,\n \"max\": 1.0,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0.5154044257916827,\n 0.0,\n 1.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe","variable\_name":"x"}

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x,y,test\_size = 0.2)

# 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.

from sklearn.linear\_model import LinearRegression, LogisticRegression  
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
  
# Модели  
clas\_models = {'LogR': LogisticRegression(),  
 'KNN\_5':KNeighborsClassifier(n\_neighbors=5),  
 # 'SVC':SVC(probability=True),  
 'Tree':DecisionTreeClassifier(),  
 'RF':RandomForestClassifier(),  
 'GB':GradientBoostingClassifier()}

from sklearn.metrics import roc\_curve, roc\_auc\_score  
  
# 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")

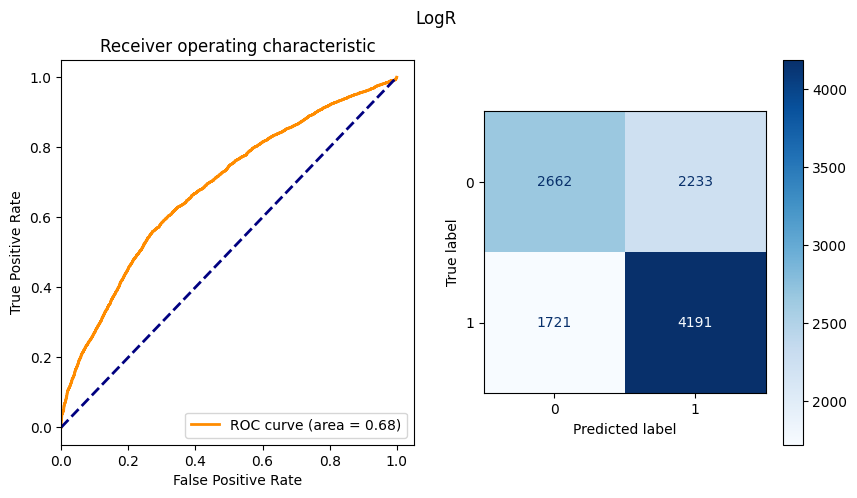
class MetricLogger:  
  
 def \_\_init\_\_(self):  
 self.df = pd.DataFrame(  
 {'metric': pd.Series([], dtype='str'),  
 'alg': pd.Series([], dtype='str'),  
 'value': pd.Series([], dtype='float')})  
  
 def add(self, metric, alg, value):  
 """  
 Добавление значения  
 """  
 # Удаление значения 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 = pd.concat([self.df, pd.DataFrame(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()  
  
# Сохранение метрик

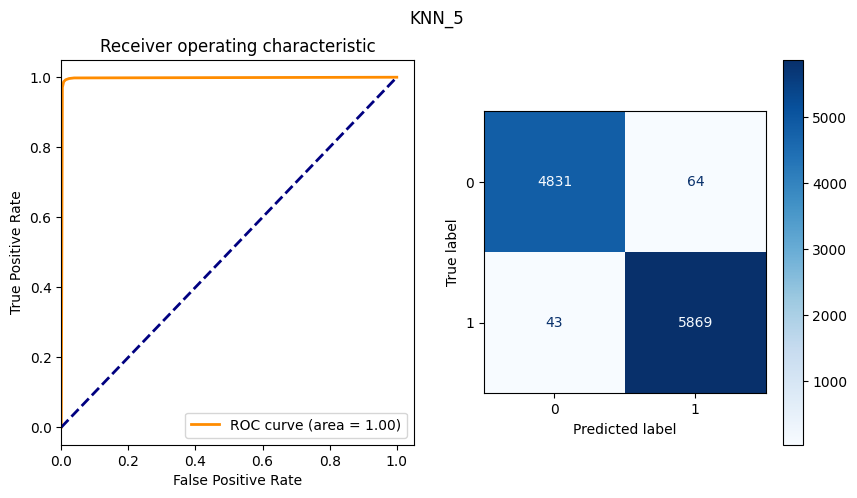
!pip install --upgrade scikit-learn

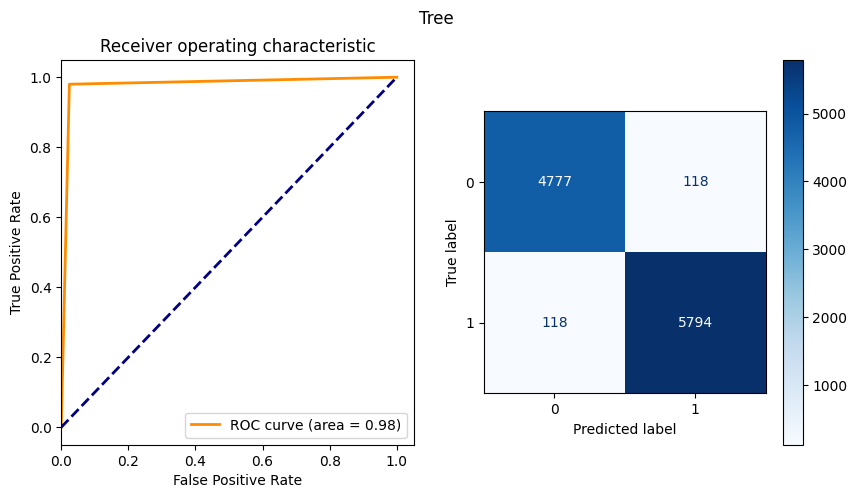
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)  
Collecting scikit-learn  
 Downloading scikit\_learn-1.5.0-cp310-cp310-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (13.3 MB)  
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 36.1 MB/s eta 0:00:00  
ent already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)  
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)  
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)  
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)  
Installing collected packages: scikit-learn  
 Attempting uninstall: scikit-learn  
 Found existing installation: scikit-learn 1.2.2  
 Uninstalling scikit-learn-1.2.2:  
 Successfully uninstalled scikit-learn-1.2.2  
Successfully installed scikit-learn-1.5.0

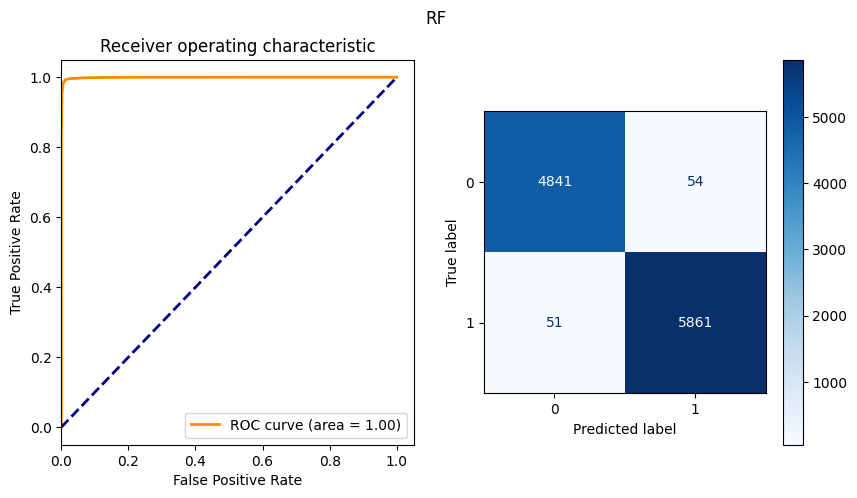
from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import ConfusionMatrixDisplay  
import matplotlib.pyplot as plt1  
  
  
  
def clas\_train\_model(model\_name, model, clasMetricLogger):  
 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])  
  
  
 y\_pred = model.predict(X\_test)  
 cm = confusion\_matrix(Y\_test.values, y\_pred)  
  
# Create the display object  
 cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['0','1'],)  
 cm\_display.plot(ax=ax[1], cmap=plt.cm.Blues)  
 fig.suptitle(model\_name)  
 plt.show()

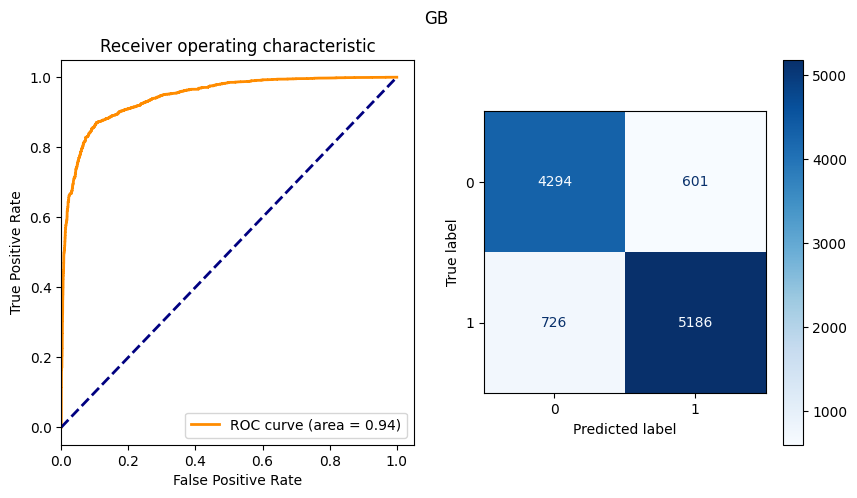
clasMetricLogger = MetricLogger()  
  
for model\_name, model in clas\_models.items():  
 clas\_train\_model(model\_name, model, clasMetricLogger)











# 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.

X\_train.shape

(43228, 8)

n\_range\_list = list(range(0,50,5))  
n\_range\_list[0] = 1

n\_range = np.array(n\_range\_list)  
tuned\_parameters = [{'n\_neighbors': n\_range}]  
tuned\_parameters

[{'n\_neighbors': array([ 1, 5, 10, 15, 20, 25, 30, 35, 40, 45])}]

from sklearn.model\_selection import GridSearchCV

clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring='roc\_auc')  
clf\_gs.fit(X\_train, Y\_train)

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid=[{'n\_neighbors': array([ 1, 5, 10, 15, 20, 25, 30, 35, 40, 45])}],  
 scoring='roc\_auc')

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

KNeighborsClassifier(n\_neighbors=20)

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

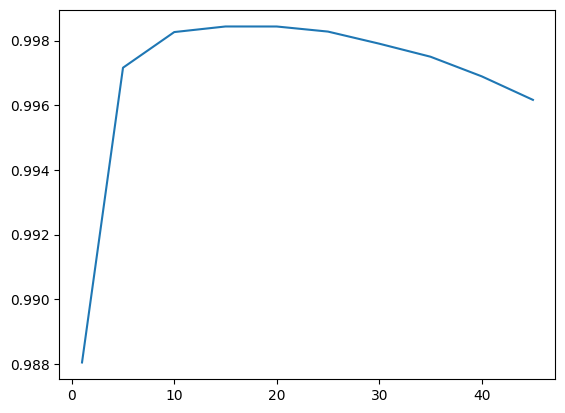
{'n\_neighbors': 20}

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

{"type":"string"}

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

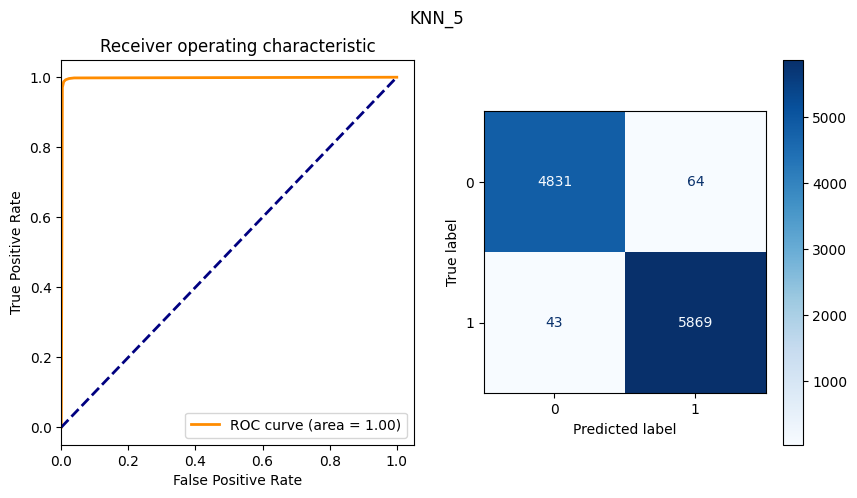
[<matplotlib.lines.Line2D at 0x7fce7845a140>]

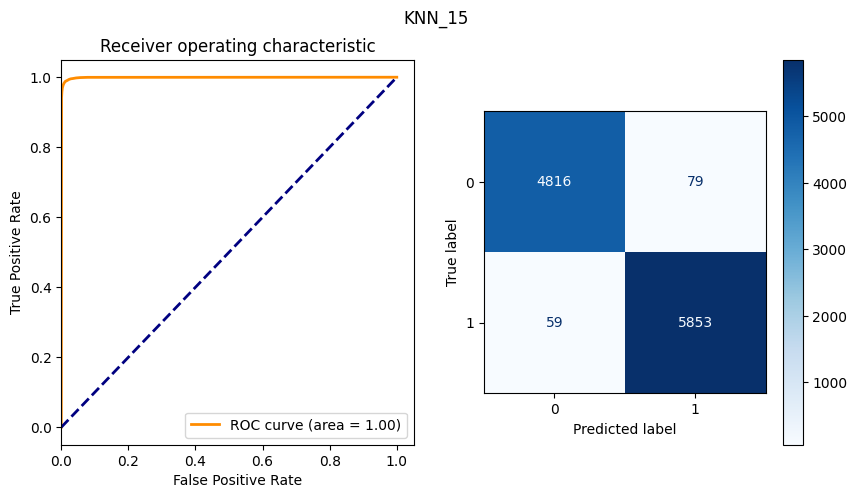


# Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

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)



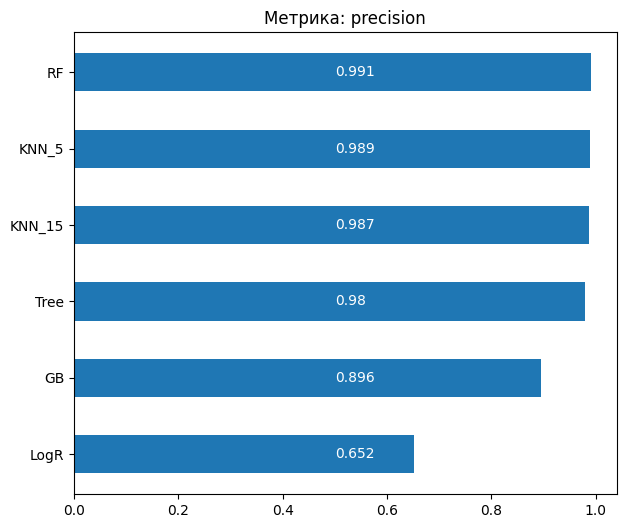


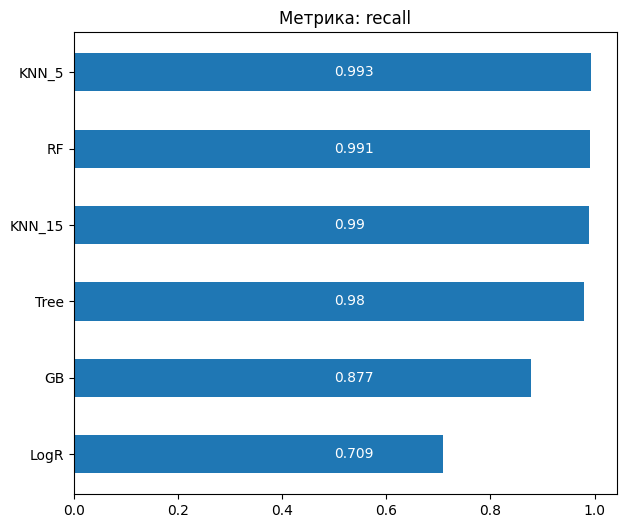
# Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

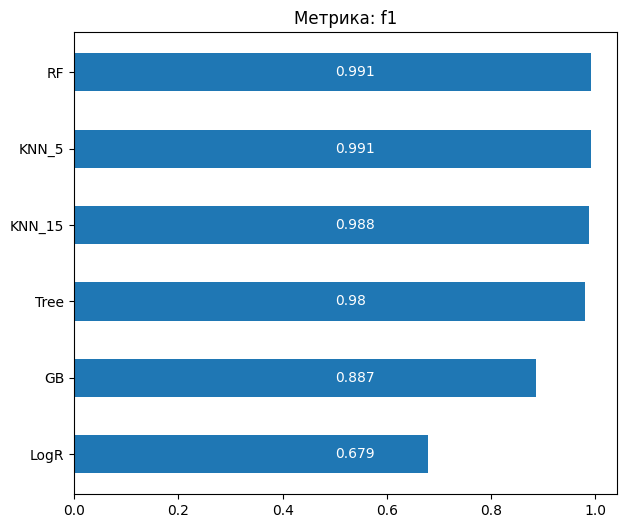
# Метрики качества модели  
clas\_metrics = clasMetricLogger.df['metric'].unique()  
clas\_metrics

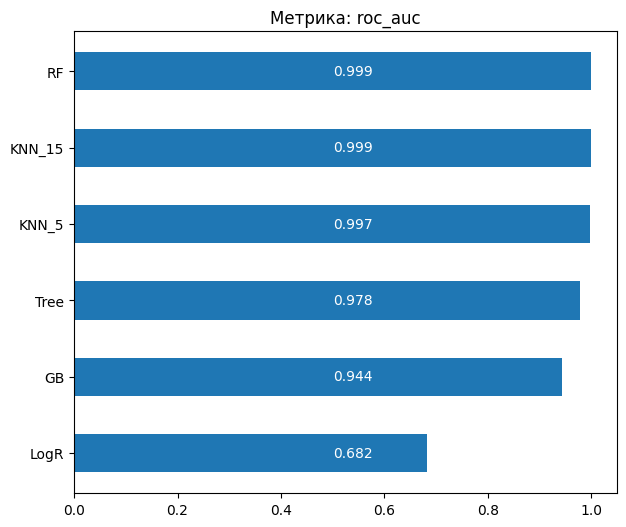
array(['precision', 'recall', 'f1', 'roc\_auc'], dtype=object)

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









Лучше всего себя показали модели RandomForest и KNN\_5 (k Nearest Neighbor)

# Заключение

В работе был проведен разведочный анализ данных с обработкой данных с неинформативными признаками, пропусков и модификацией структуры и самих данных. Также было проведено кодирование категориальных признаков, масштабирование данных и сравнение масштабированных данных с исходными. Был выполнен корреляционный анализ и на его основании были выбраны модели для решения задачи классификации. Исходные данные были разделены на тестовую и обучающую выборку, на основе этих выборок были обучены выбранные модели. Также была построена наиболее оптимальная модель. Все модели подверглись сравнению для определения наилучшего качества решения задачи классификации, для этого использовались несколько метрик классификации.

# Список литературы

1. Kaggle: Your home for Data Science [Электронный ресурс]. URL: https://www.kaggle.com/
2. sckit-learn: machine learning in Python [Электронный ресурс]. URL: https://scikit-learn.org/stable/
3. Matplolib - visualization via Python [Электронный ресурс]. URL: https://matplotlib.org/
4. Методические указания по разработке НИРС [Электронный ресурс]. URL: https://github.com/ugapanyuk/ml\_course\_2024/wiki/TMO\_NIRS
5. Репозиторий курсов "Технологии машинного обучения", бакалавриат, 6 семестр [Электронный ресурс]. URL: https://github.com/ugapanyuk/ml\_course\_2024/wiki/COURSE\_TMO