Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчет по домашнему заданию

«Опорный пример для выполнения проекта по анализу данных»
по курсу
“Методы машинного обучения”

Выполнил:
Али Диб А.Ж.
Студент группы ИУ5-22М

Москва, 2020

1) Текстовое описание набора данных

В качестве набора данных мы будем использовать набор данных для выявления рака моло
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Датасет содержит следующие колонки:

- **radius** (среднее расстояние от центра до точек по периметру)
- **texture** (стандартное отклонение значений оттенков серого)
- **perimeter**
- **area**
- **smothness** (локальное изменение длины радиуса)
- **compactness** ($\frac{\text{периметр}^2}{\text{площадь}-1,0}$)

7. **concavity** (выраженность вогнутых участков контура)
8. **concave points** (количество вогнутых частей контура)
9. **symmetry**
10. **fractal dimension** («приближение береговой линии» - 1)

Среднее значение, стандартная ошибка и «наихудшее» или наибольшее (среднее из трех са
были рассчитаны для каждого изображения, что дало 30 признаков. Например, поле 3 - сре
наихудший радиус.

Импорт библиотек

Импортируем библиотеки с помощью команды import. Как правило, все команды import ра

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
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
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_l
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_grap
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
#from gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
from sklearn.datasets import *
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Future
  import pandas.util.testing as tm
```

Загрузка данных

Загрузим файлы датасета в помощью sklearn датасет.

```
cancer = load_breast_cancer()
```

```
for x in cancer:
    print(x)
```

```
data
target
target_names
DESCR
feature_names
filename
```

```
cancer['feature_names']
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
cancer['data'].shape
```

```
(569, 30)
```

```
data1=pd.DataFrame(data=np.c_[cancer['data'],cancer['target']],
                   columns = list(cancer['feature_names']) + ['target'])
```

```
data1
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	m conc poi
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10
...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00

569 rows × 31 columns

2) Основные характеристики датасета

```
# Первые 5 строк датасета
data1.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mea concav point
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

```
# Размер датасета - 8143 строк, 7 колонок
data1.shape
```

```
(569, 31)
```

```
total_count = data1.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 569
```

```
# Список колонок
data1.columns
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

```
# Список колонок с типами данных
data1.dtypes
```

```
mean radius                float64
mean texture               float64
mean perimeter             float64
mean area                  float64
mean smoothness            float64
mean compactness           float64
mean concavity             float64
mean concave points        float64
mean symmetry              float64
mean fractal dimension     float64
radius error               float64
texture error              float64
perimeter error            float64
area error                 float64
smoothness error           float64
compactness error          float64
concavity error            float64
concave points error       float64
symmetry error             float64
fractal dimension error    float64
worst radius               float64
worst texture              float64
worst perimeter            float64
worst area                 float64
worst smoothness           float64
worst compactness          float64
worst concavity            float64
worst concave points       float64
worst symmetry             float64
worst fractal dimension    float64
target                     float64
dtype: object
```

```
# Проверим наличие пустых значений
```

```
# Количество пустых значений - все значения заполнены
data1.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

Мы заметили, что все данные имеют тип float64, поэтом категориальную кодировку символов. И мы также обра данных нет пропусков, поэтому нам не нужно анализир пропуски в данных.

```
# Основные статистические характеристки набора данных
data1.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	conc
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0

```
#Определим уникальные значения для целевого признака
data1['target'].unique()
```

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

Целевой признак является бинарным и содержит только значения 0 и 1.

3) Визуальное исследование датасета

Для визуального исследования могут быть использованы различные виды диаграмм, буде
диаграмм, которые используются достаточно часто.

Будет использовано две библиотеки:

- **Matplotlib**
- **Seaborn**

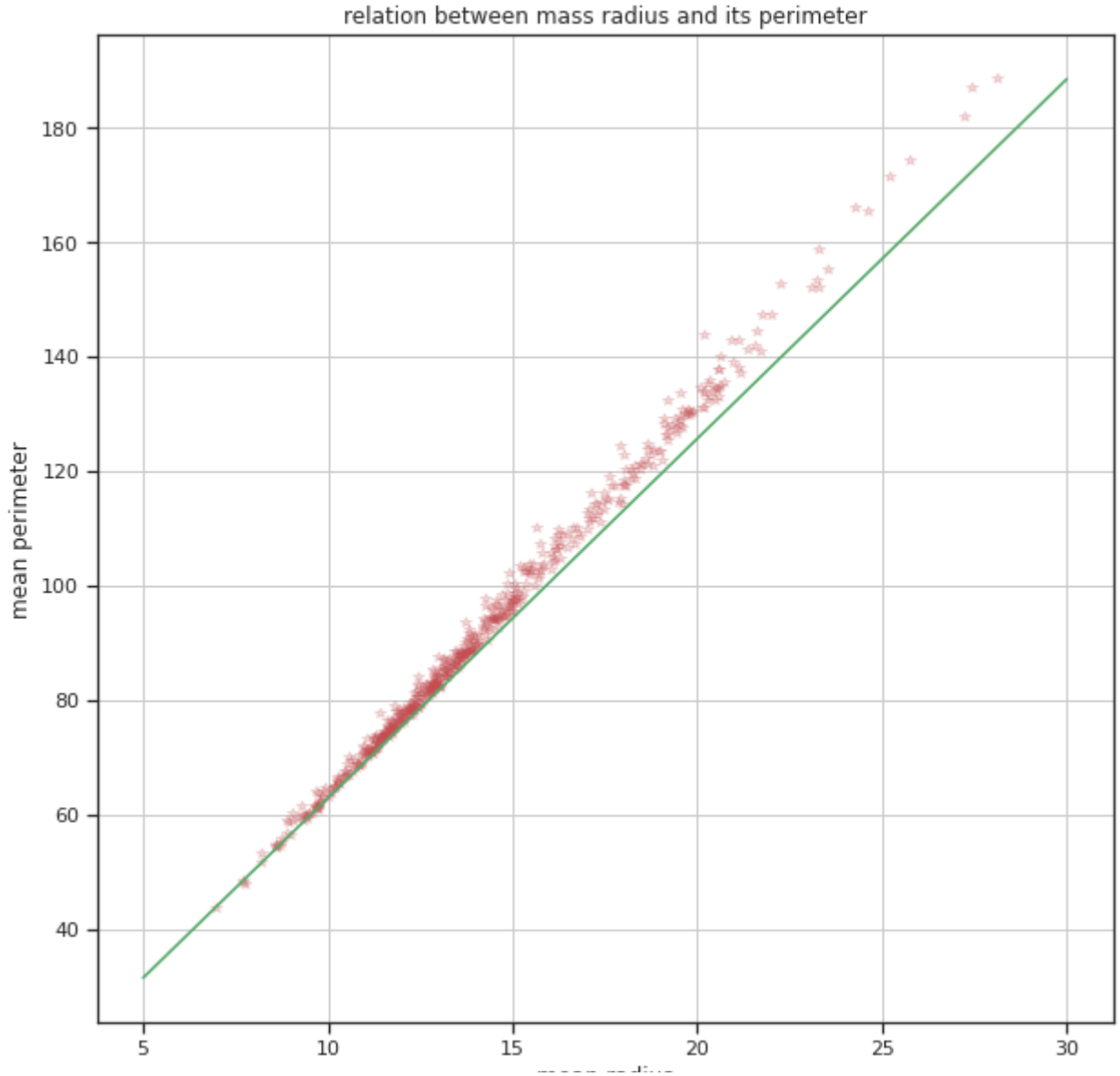
Диаграмма рассеяния

Позволяет построить распределение двух колонок данных и визуально обнаружить налич
значения упорядочены (например, по времени).

Matplotlib

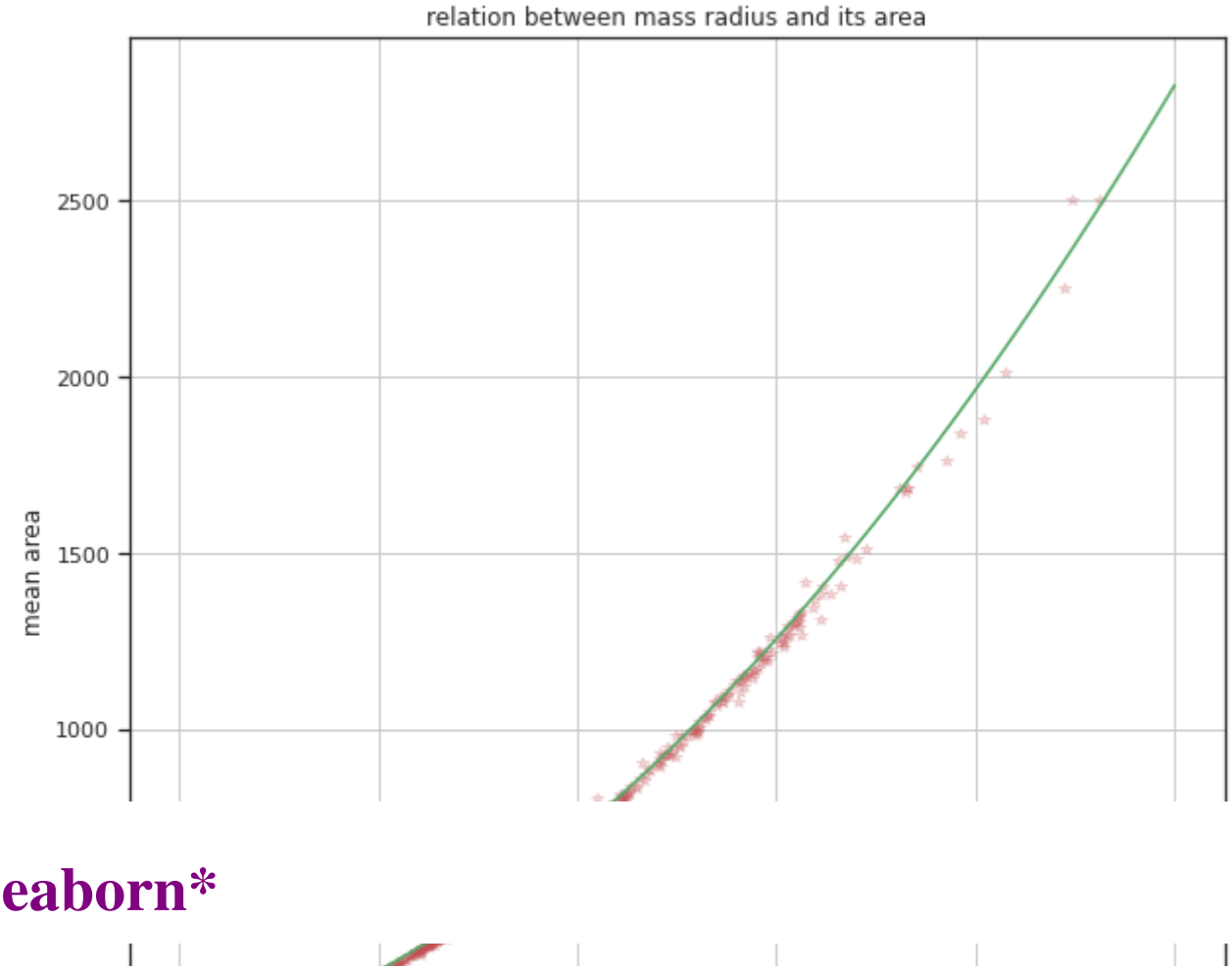
```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
ax.grid()
plt.plot(data1['mean radius'],data1['mean perimeter'],'*r',alpha=0.2)
ax.set_xlabel('mean radius')
ax.set_ylabel('mean perimeter')
ax.set_title('relation between mass radius and its perimeter')
xxx=np.linspace(5,30)
yyy=2*np.pi*xxx
plt.plot(xxx,yyy,'g')
```

```
[<matplotlib.lines.Line2D at 0x7f1efc317e10>]
```




```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
ax.grid()
plt.plot(data1['mean radius'],data1['mean area'],'*r',alpha=0.2)
ax.set_xlabel('mean radius')
ax.set_ylabel('mean area')
ax.set_title('relation between mass radius and its area')
xxx=np.linspace(5,30)
yyy=np.pi*xxx**2
plt.plot(xxx,yyy,'g')
```

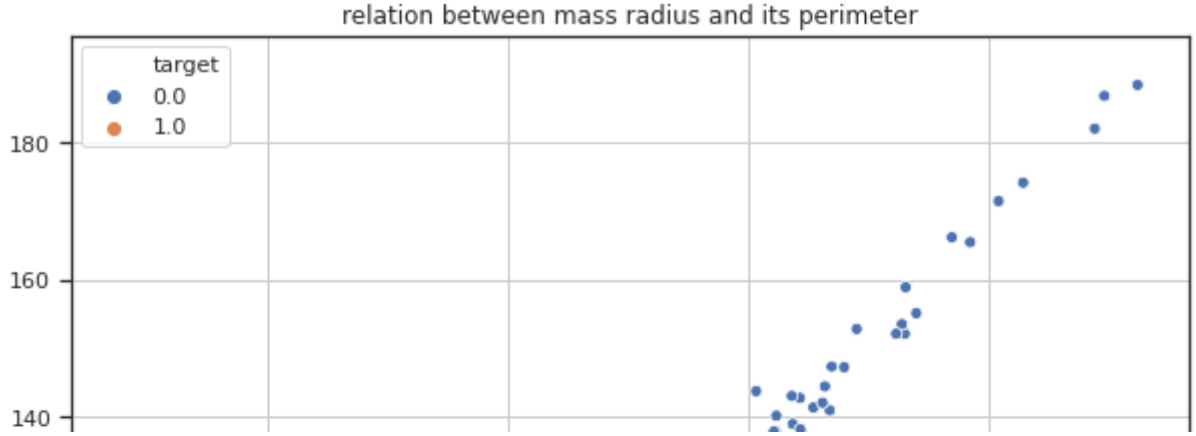
```
[<matplotlib.lines.Line2D at 0x7f1efc2c3f28>]
```



Seaborn

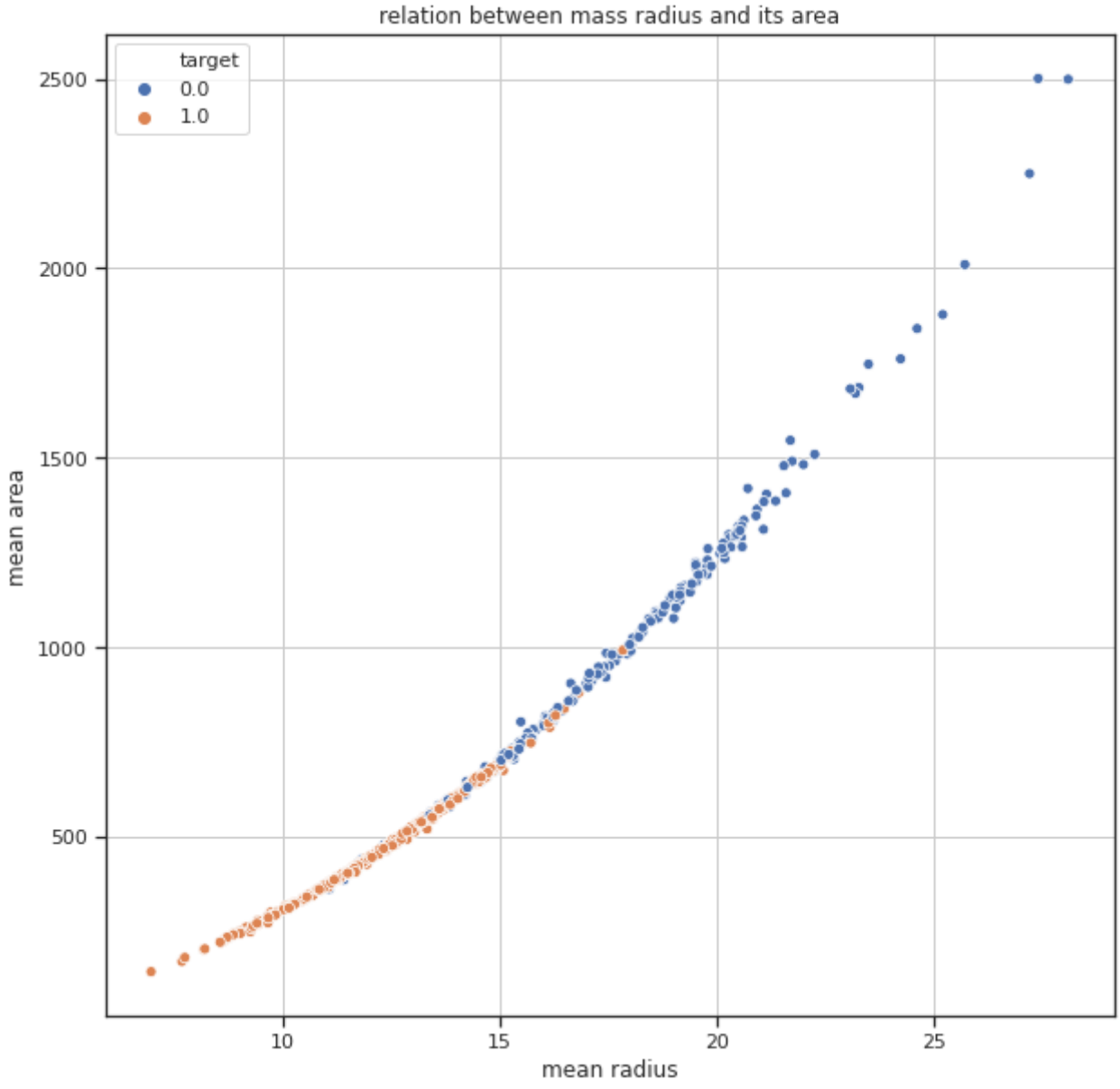
```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
ax.grid()
sns.scatterplot(data1['mean radius'],data1['mean perimeter'],color='r',hue=data1['t
ax.set_title('relation between mass radius and its perimeter')
```

```
Text(0.5, 1.0, 'relation between mass radius and its perimeter')
```




```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
ax.grid()
sns.scatterplot(data1['mean radius'],data1['mean area'],color='r',hue=data1['target
ax.set_title('relation between mass radius and its area')
```

```
Text(0.5, 1.0, 'relation between mass radius and its area')
```



Можно видеть что между полями mean radius и mean perimeter пристутствует почти линейн
Можно видеть что между полями mean radius и mean area пристутствует почти параболичес

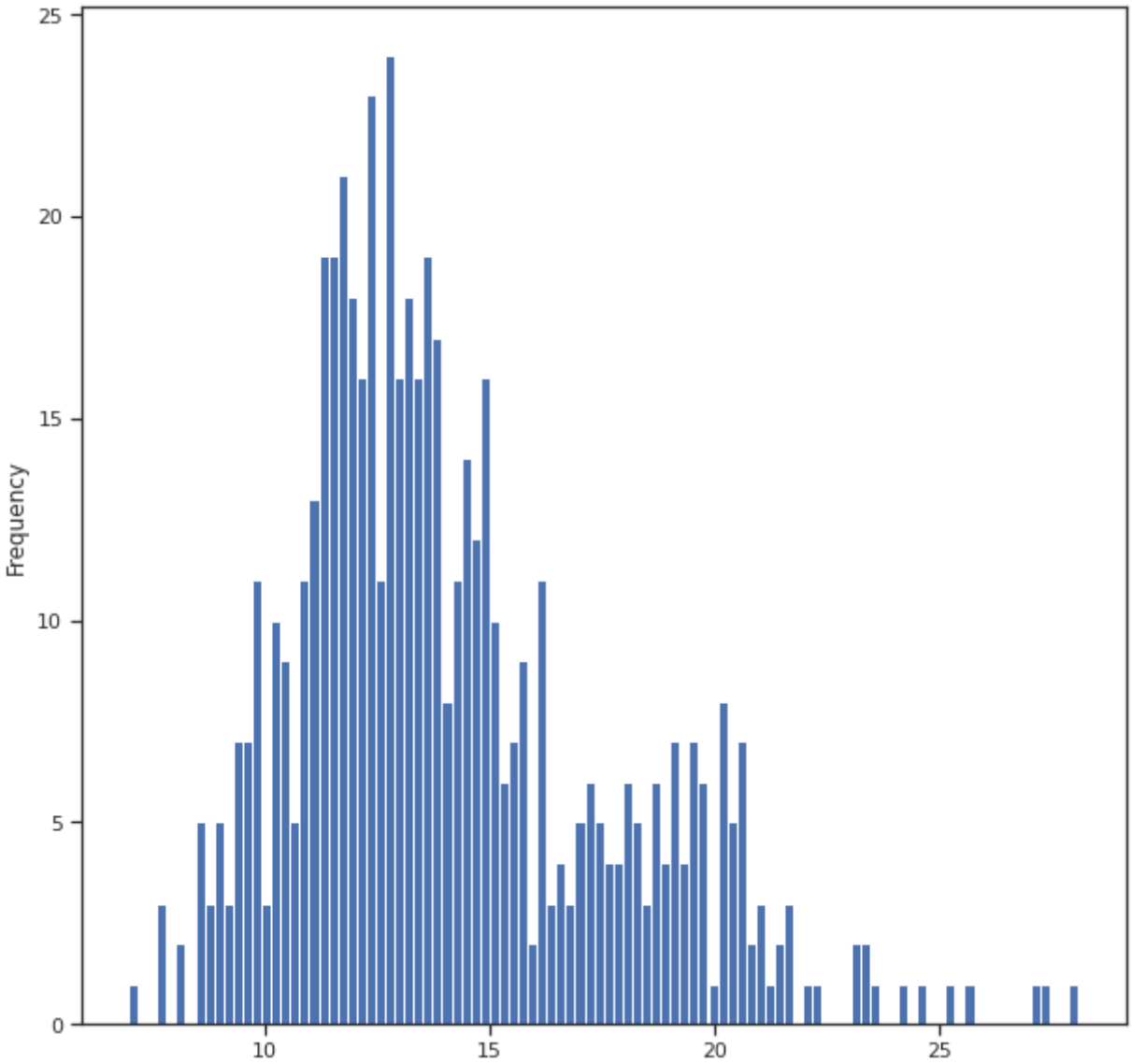
Гистограмма

Позволяет оценить плотность вероятности распределения данных.

Matplotlib

```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
data1['mean radius'].plot.hist(bins=100)
```

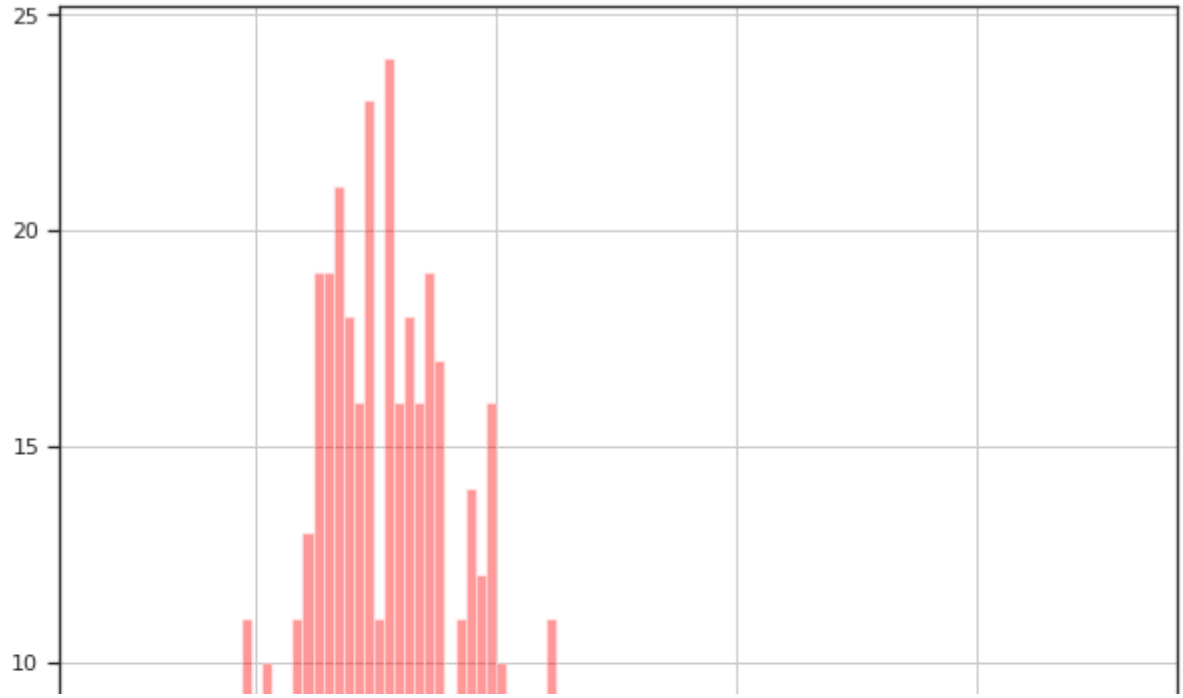
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1efc36b518>
```



Seaborn

```
fig=plt.figure(1,figsize=(10,10))
ax=fig.gca()
ax.grid()
sns.distplot(data1['mean radius'],rug=False,kde=False,color='red',bins=100)
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
sns.kdeplot(data1['mean radius'], shade=True,bw=0.01);
ax.set_title('график плотности')
```

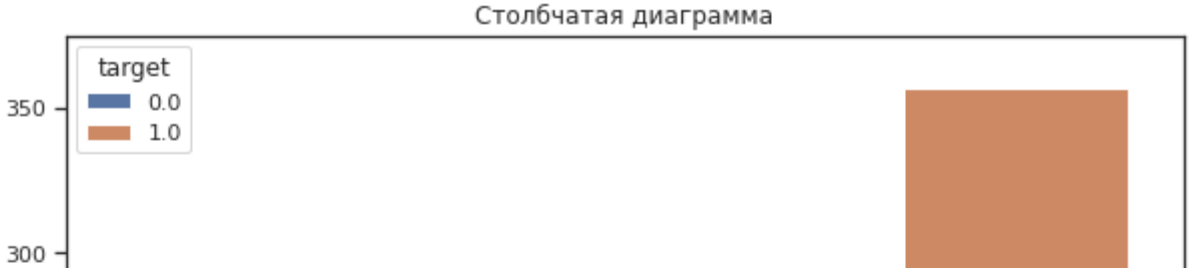
```
Text(0.5, 1.0, 'график плотности')
```



Столбчатая диаграмма

```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
sns.countplot(x=data1['target'],hue=data1['target'])
ax.set_title('Столбчатая диаграмма')
```

```
Text(0.5, 1.0, 'Столбчатая диаграмма')
```



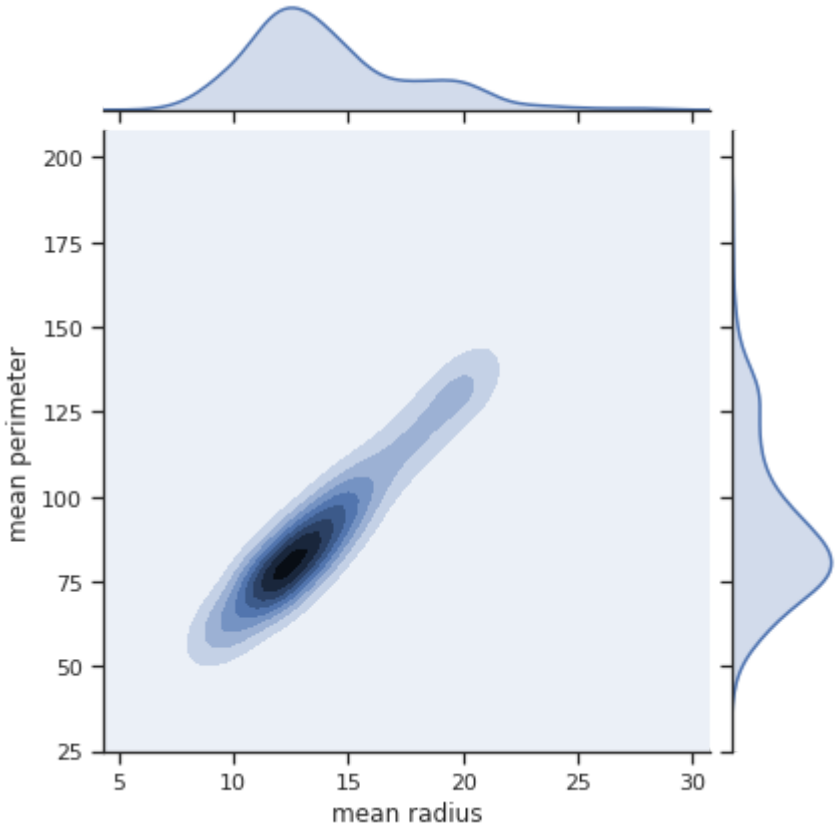
Этот график показывает, когда у человека есть опухоль, какой процент является злокачест независимо от характеристик опухоли.

Joinplot

Комбинация гистограмм и диаграмм рассеивания.

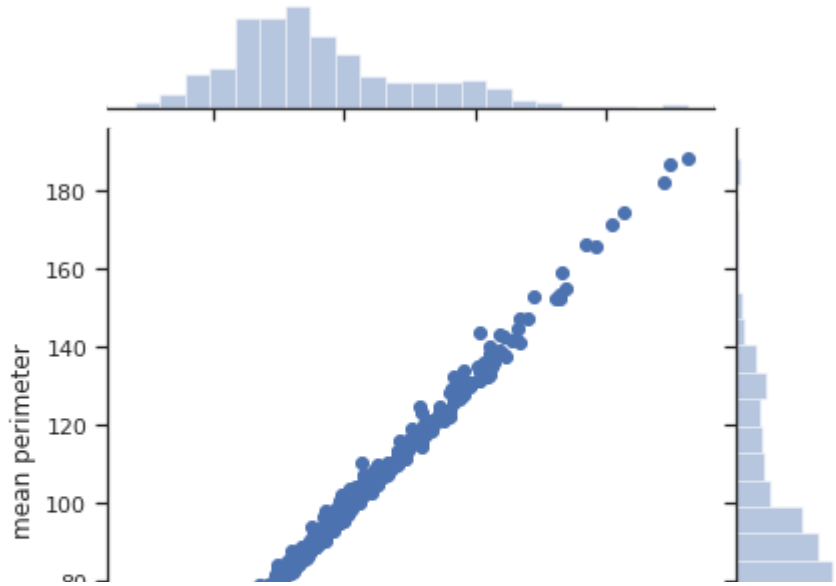
```
sns.jointplot(data1['mean radius'],data1['mean perimeter'],kind='kde')
```

```
<seaborn.axisgrid.JointGrid at 0x7f1ef9107198>
```



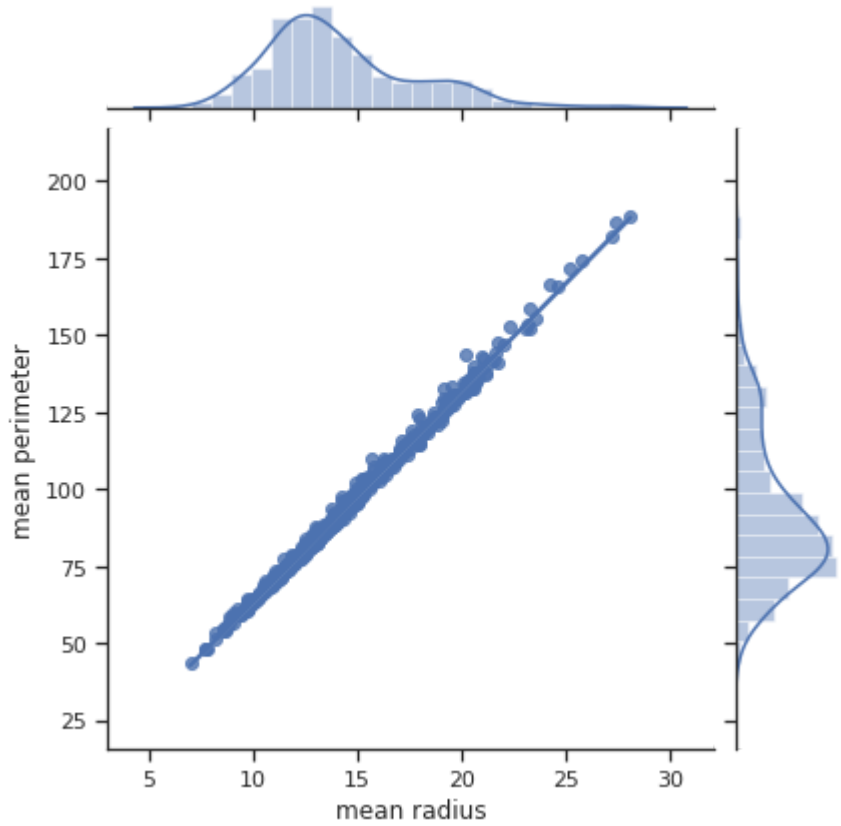

```
sns.jointplot(data1['mean radius'],data1['mean perimeter'],kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7f1ef911dcc0>
```



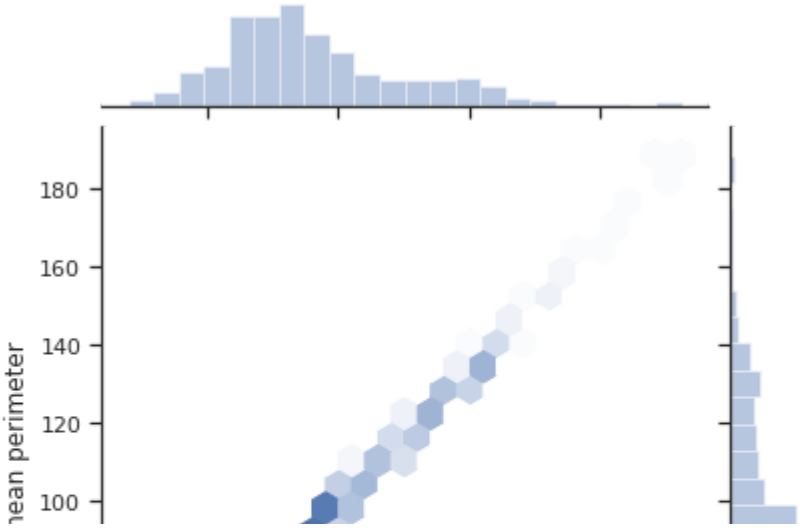

```
sns.jointplot(data1['mean radius'],data1['mean perimeter'],kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f1ef756f908>
```




```
sns.jointplot(data1['mean radius'],data1['mean perimeter'],kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x7f1ef745e710>
```



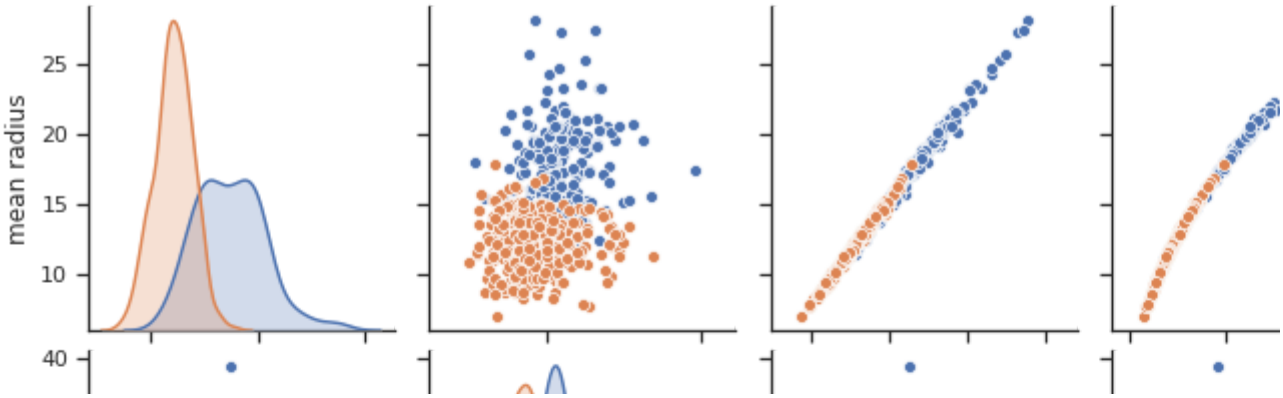
Парные диаграммы

Комбинация гистограмм и диаграмм рассеивания для всего набора данных.

Выводится матрица графиков. На пересечении строки и столбца, которые соответстуют дву
рассеивания. В главной диагонали матрицы строятся гистограммы распределения соответ

```
sns.pairplot(data1[['mean radius','mean texture','mean perimeter','mean area','targ
```

```
<seaborn.axisgrid.PairGrid at 0x7f1ef72d5860>
```

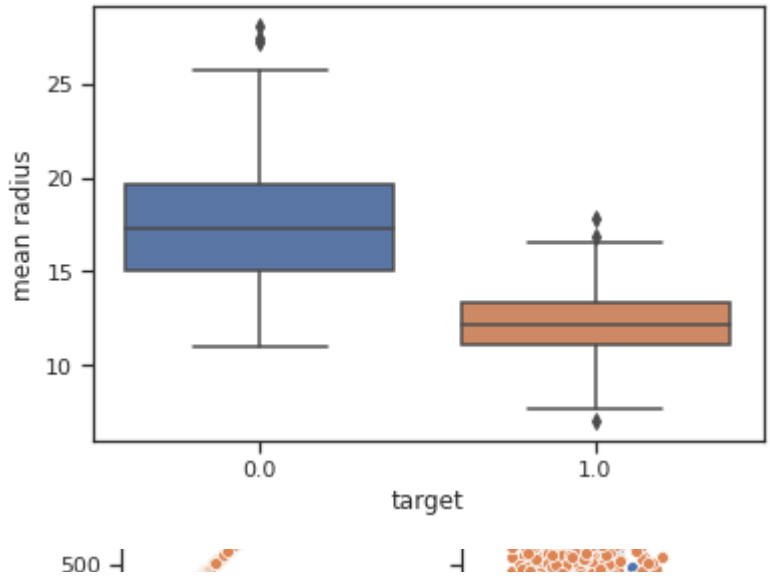


Ящик с усами

Отображает одномерное распределение вероятности.

```
sns.boxplot(x='target', y='mean radius', data=data1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef6c4c0b8>
```



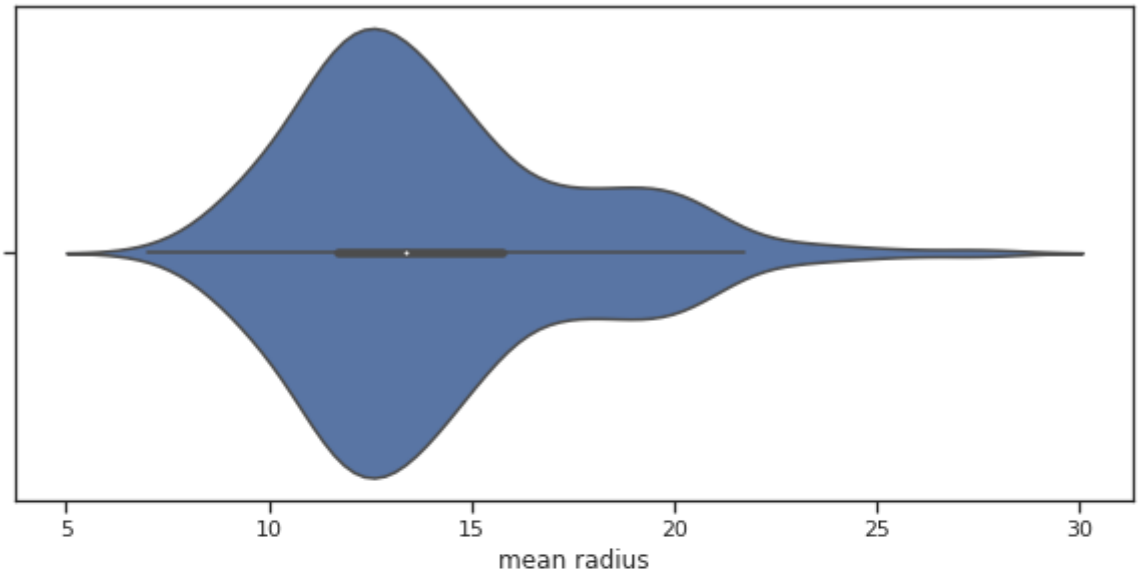
мы можем видеть, что медиана "mean radius" доброкачественной опухоли меньше, чем мед

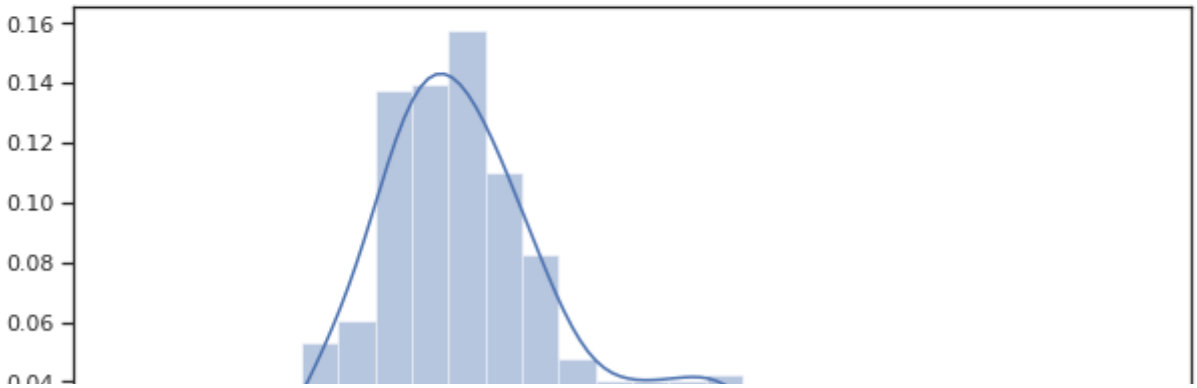
Violin plot

Похоже на предыдущую диаграмму, но по краям отображаются распределения плотности.

```
fig, ax = plt.subplots(2, 1, figsize=(10,10))
sns.violinplot(ax=ax[0], x=data1['mean radius'])
sns.distplot(data1['mean radius'], ax=ax[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef6c14c18>
```






```
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
sns.violinplot(x=data1['target'], y=data1['mean radius'], data=data1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef6c0ce80>
```

4) Информация о корреляции признаков


```
data1.corr()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	co
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	

В нашем случае набор данных имеет большое количество признаков, поэтому анализ числ
становится неудобным.

Чтобы визуализировать матрицу корреляции, мы будем использовать тепловую карту тепл
корреляции в разных цветах.

error

```
# Числовые колонки для масштабирования
scale_cols = ['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
```

```
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension']
```

compactness

```
sc1 = StandardScaler()
sc1_data = sc1.fit_transform(data1[scale_cols])
```

worst

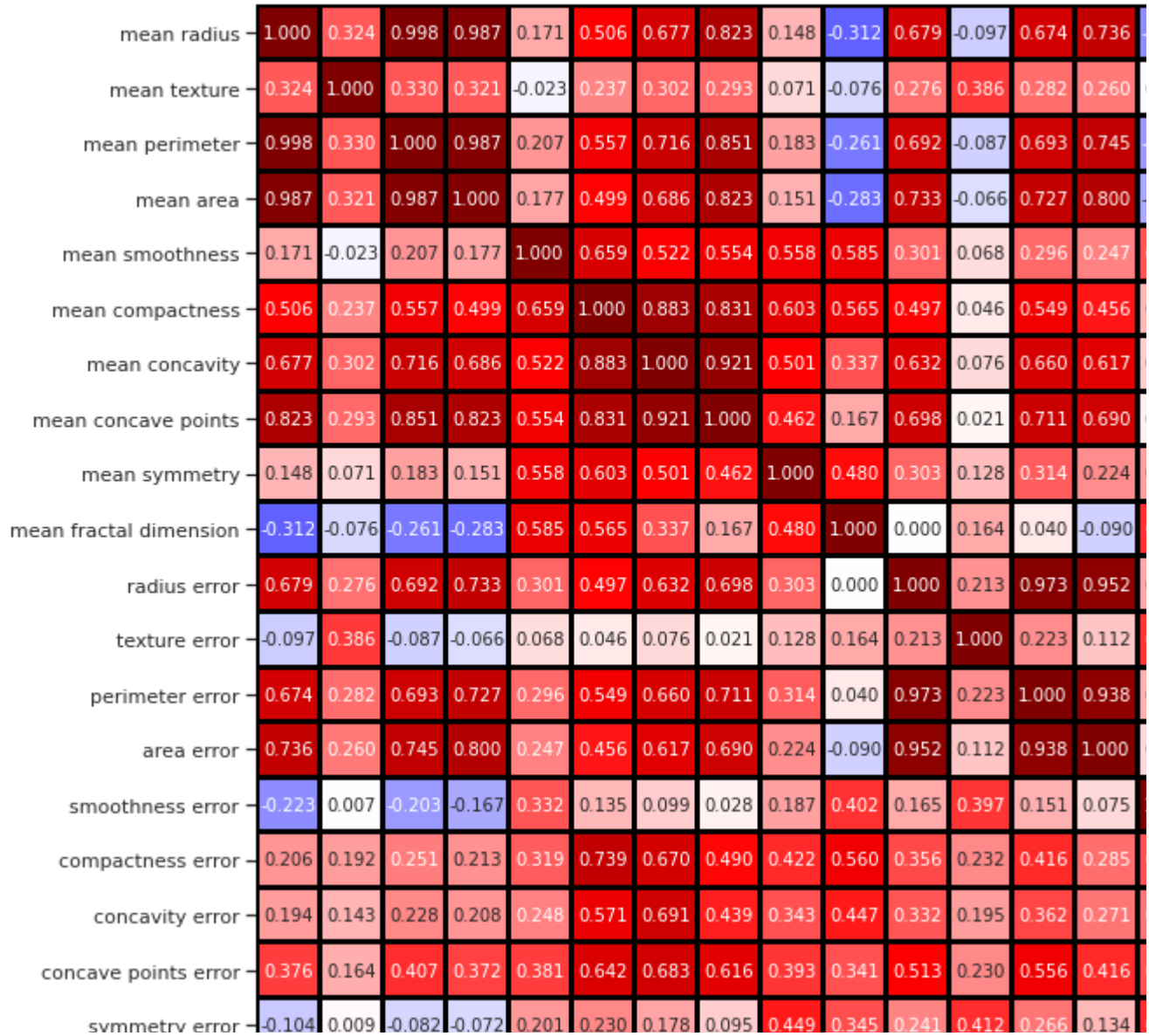
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]


data_scaled=pd.DataFrame(data=np.c_[sc1_data,cancer['target']],
                  columns = scale_cols_postfix + ['target'])
```

worst fractal 0.007066 0.119205 0.051019 0.003738 0.499316 0.687382

Корреляционные матрицы для исходных данных

```
fig=plt.figure(figsize=(25,18))
ax=fig.gca()
sns.heatmap(data1.corr(), annot = True, vmin=-1, vmax=1, center= 0, cmap= 'seismic'
```

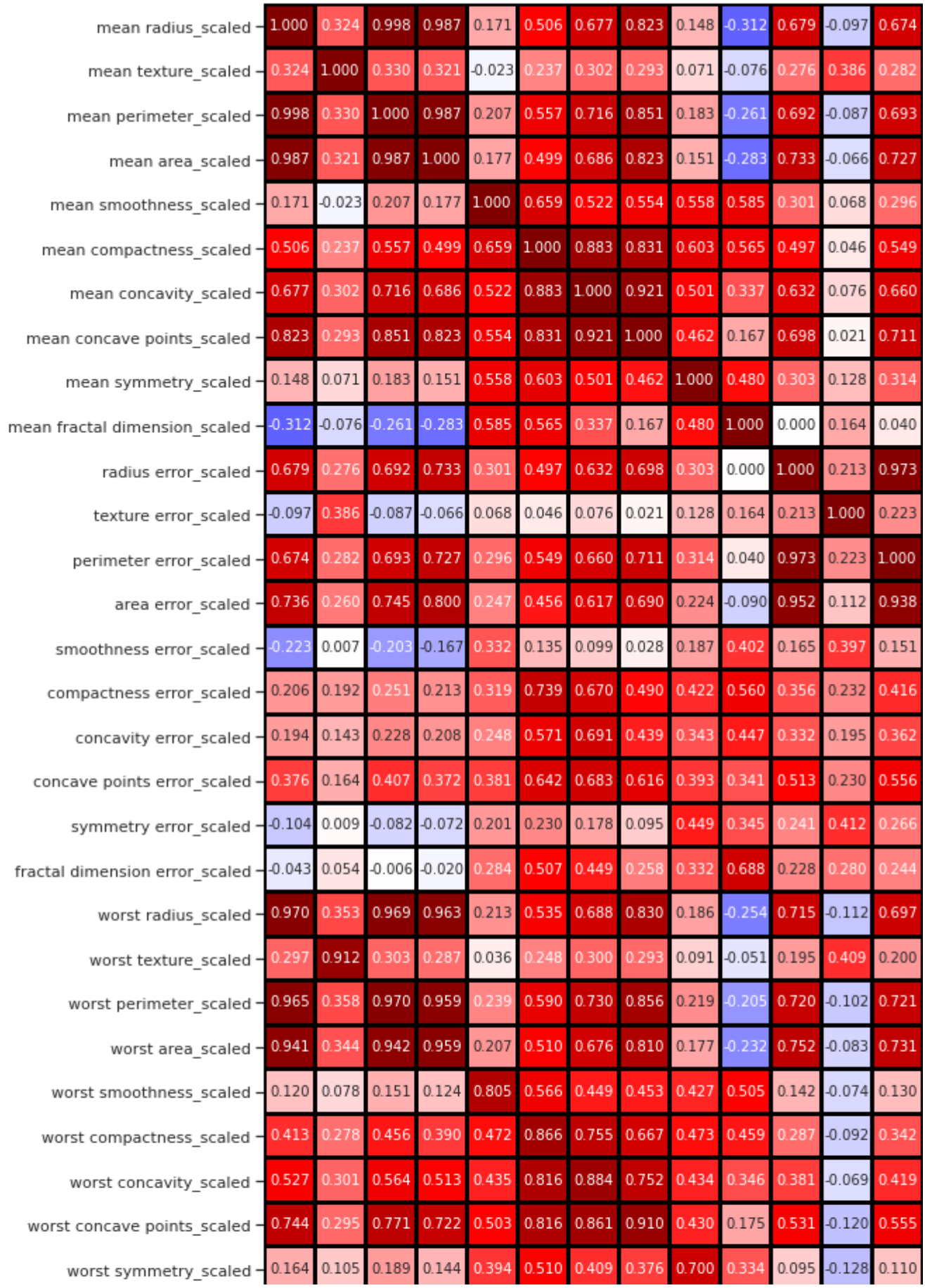
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef6c26b70>
```



Корреляционные матрицы для масштабированных данных


```
fig=plt.figure(figsize=(25,18))
ax=fig.gca()
sns.heatmap(data_scaled.corr(), annot = True, vmin=-1, vmax=1, center= 0, cmap= 'se
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef6af7940>
```



Корреляционная матрица содержит коэффициенты корреляции между всеми парами призн

Корреляционная матрица симметрична относительно главной диагонали. На главной диаго признака самого с собой).

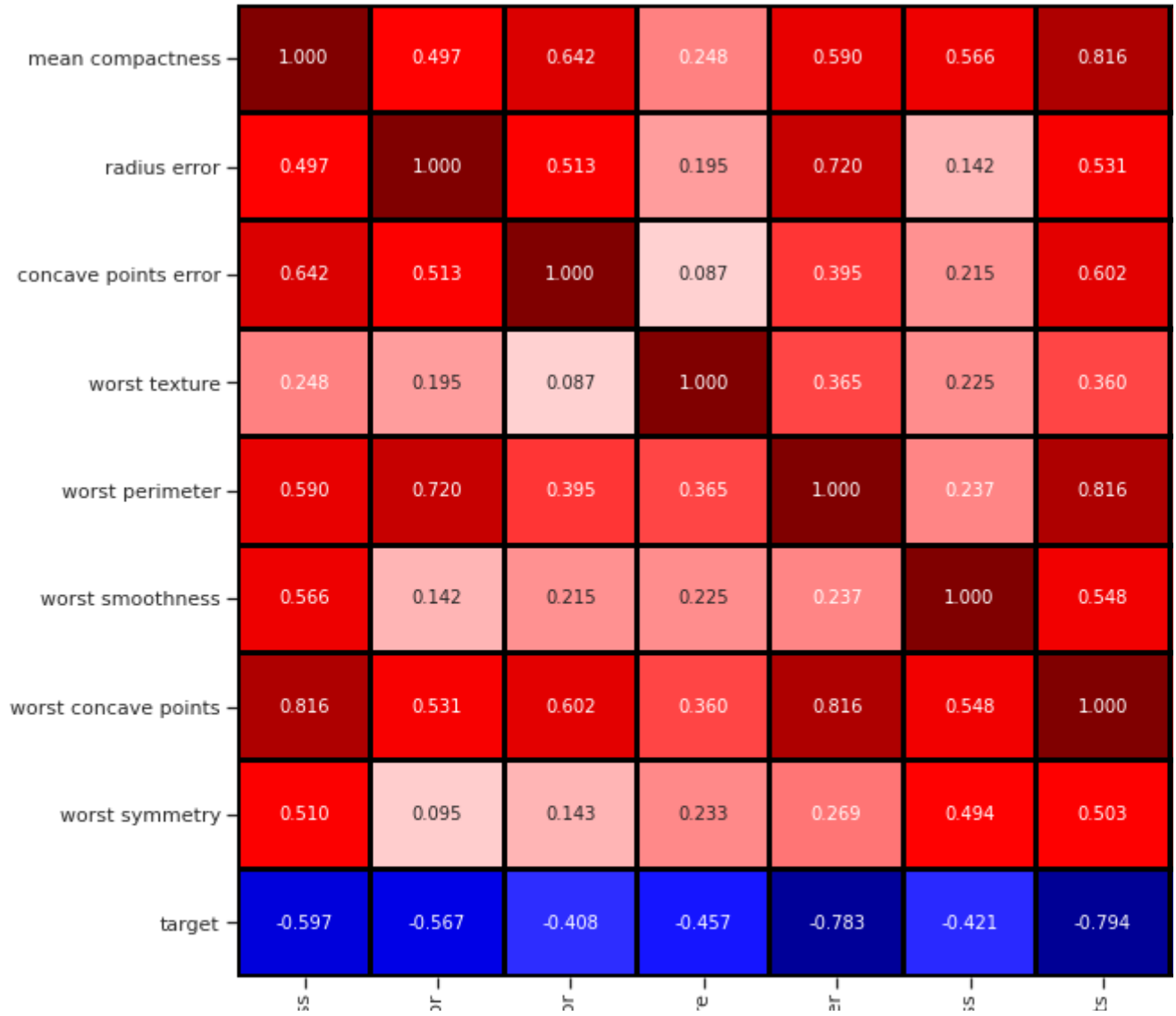
На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак наиболее сильно коррелирует с "worst concave points" (-0.794) и "wors обязательно следует оставить в модели.
- Целевой признак отчасти коррелирует с "mean compactness" (-0.597),"radius error" (-0.56 texture" (-0.547) ,"worst smoothness" (-0.421),"worst symmetry" (-0.416). Этот признак стои
- Целевой признак слабо коррелирует с "mean smoothness" (-0.359) , "mean symmetry" (-0 "texture error" (0.008) , "smothness error" (0.067) , "Compactness error" (-0.293) , "concavity e "fractal dimension error" (-0.078) и "worst fractal dimension" (-0.324). Скорее всего эти при возможно они только ухудшат качество модели.
- "mean radius" и "mean perimeter" очень сильно коррелируют между собой (0.998). Это н величина производная от mean radius. Поэтому из этих признаков в модели можно ос и то же самое между "mean radius" и ("mean area" (0.987), "worst radius" (0.969), "worst pe
- "mean texture" и "worst texture" очень сильно коррелируют между собой (0.912). Это неу производная от mean texture. Поэтому из этих признаков в модели можно оставлять т
- "radius error" и "perimeter error" очень сильно коррелируют между собой (0.973). Это неу производная от radius error. Поэтому из этих признаков в модели можно оставлять тол и то же самое между "radius error" и "area error" (0.952)
- "mean concavity" и "worst concave points" очень сильно коррелируют между собой (0.86 concavity величина производная от worst concave points. Поэтому из этих признаков в и то же самое между "worst concave points" и ("mean concave points" (0.910), "worst conc
- "mean compactness" и "worst compactness" очень сильно коррелируют между собой (0. compactness величина производная от worst compactness. Поэтому из этих признаков
- Также можно сделать вывод, что выбирая из признаков ("mean radius","mean perimeter "worst area") лучше выбрать "worst perimeter", потому что он сильнее коррелирован с ц зависимые признаки сильно коррелированы с целевым, то оставляют именно тот при сильнее.

Матрица корреляции после удаления столбцов, которые не име модели

```
fig=plt.figure(figsize=(15,10))
ax=fig.gca()
sns.heatmap(data1[['mean compactness','radius error','concave points error','worst
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ef4f8afd0>
```



4) Выбор метрик для последующей оценки качества мо

Мы будем использовать метрику для точности, f1_score, отзыва и roc_auc. И мы увидим кри
путаницы.

```
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)].ind
        # Добавление нового значения
```

```
        # 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()
```

5)Выбор признаков, подходящих для построения модел

Как мы уже говорили, мы будем использовать mean compactness, radius error, concave points smoothness, worst concave points, и worst symmetry за модель. Все остальные признаки мал

```
X_train, X_test, y_train, y_test = train_test_split(data1[['mean compactness','radi
                                                           'worst smoothness','wors
```

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

```
((28, 9), (541, 9), (28,), (541,))
```

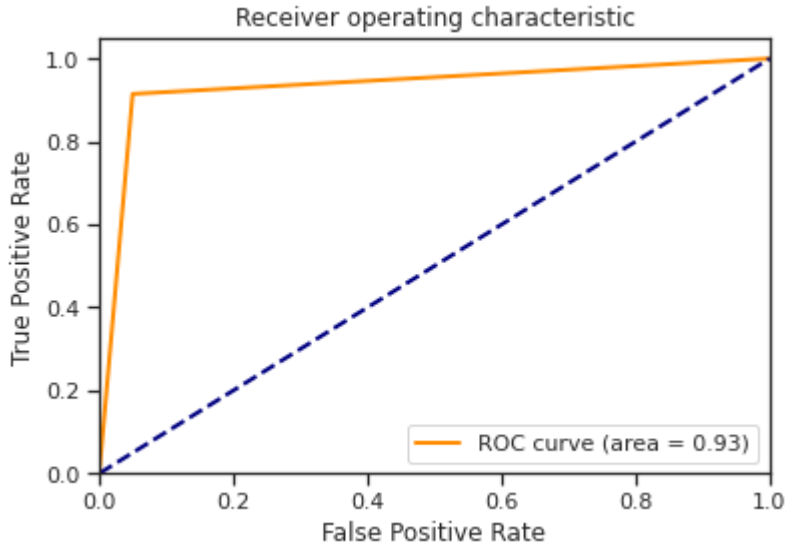
6) Выбор моделей для решения задачи

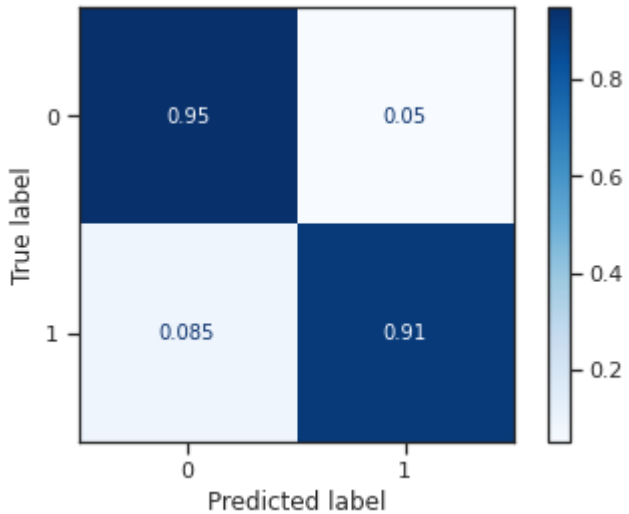
Мы выберем многие модели для классификации, а затем сравним их производительность будем использовать K-Neighbours Classifier с пятью соседями, классификатор дерева решен классификатор повышения градиента, классификатор мешков, классификатор дополнител повышения Ada.

```
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier(),
               'Bag':BaggingClassifier(),
               'ExTree':ExtraTreesClassifier(),
               'Adab':AdaBoostClassifier()}


# Сохранение метрик
clasMetricLogger = MetricLogger()


# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, 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
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()


def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, y_train)
    Y_pred = model.predict(X_test)
    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)

    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)

    print('****************************************************')
    print(model)
    print('****************************************************')

    draw_roc_curve(y_test.values, Y_pred)

    plot_confusion_matrix(model,X_test, y_test.values,
                          display_labels=['0','1'],
```

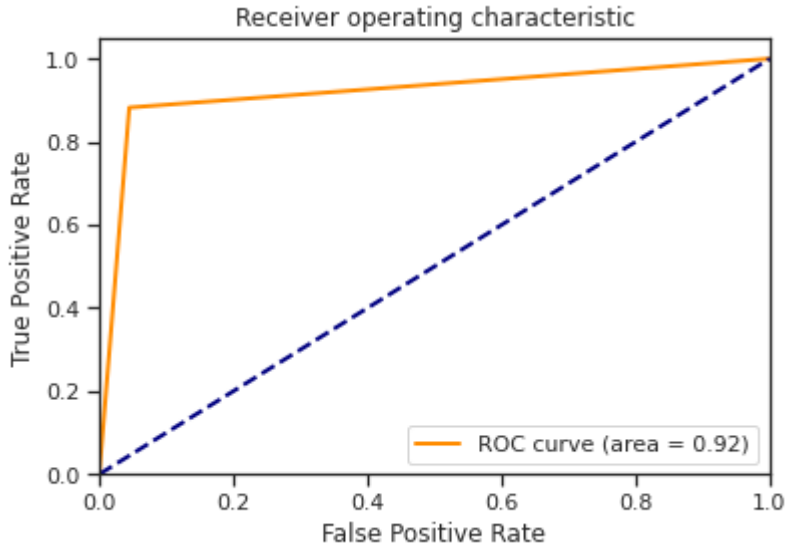
```
                        cmap=plt.cm.Blues, normalize='true')
    plt.show()


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

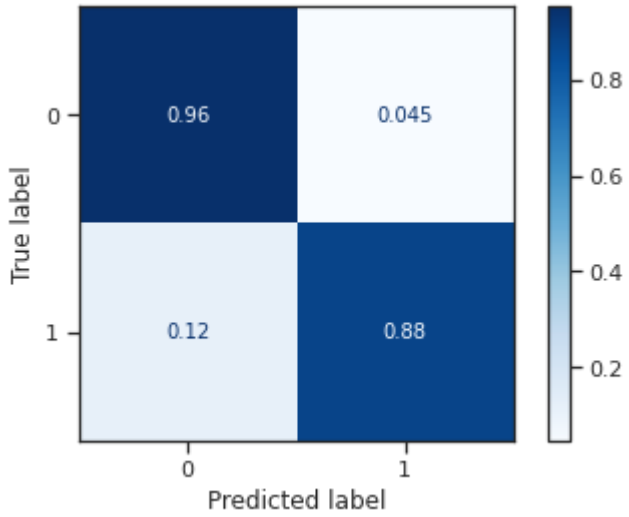
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



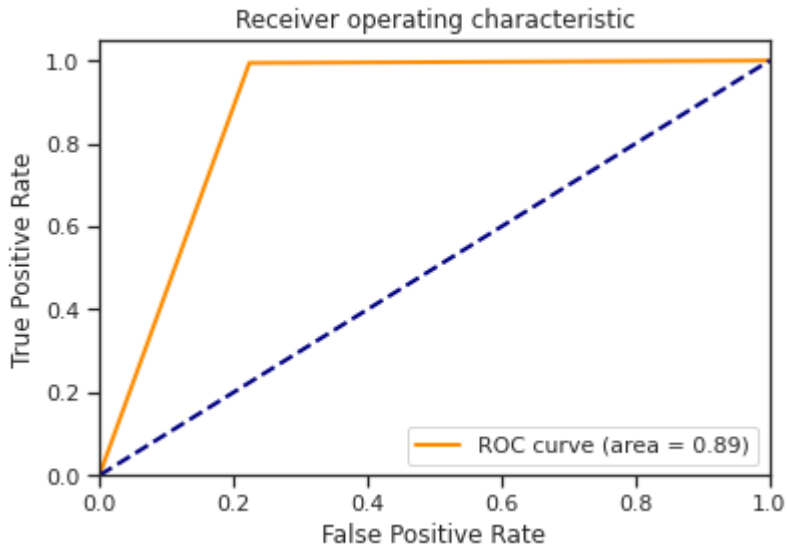


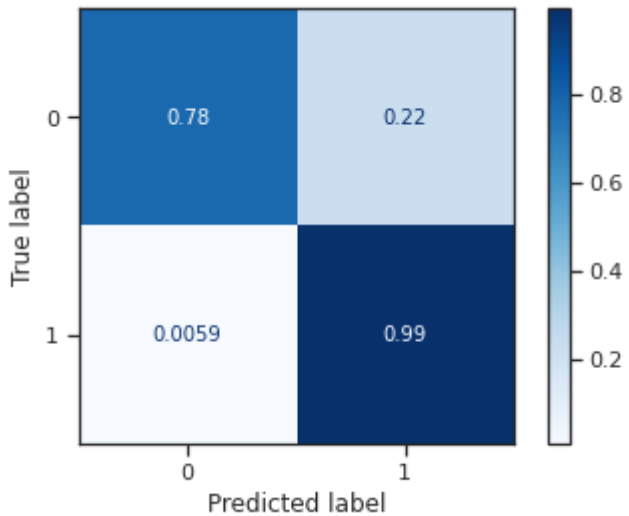

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
**************************************************
```





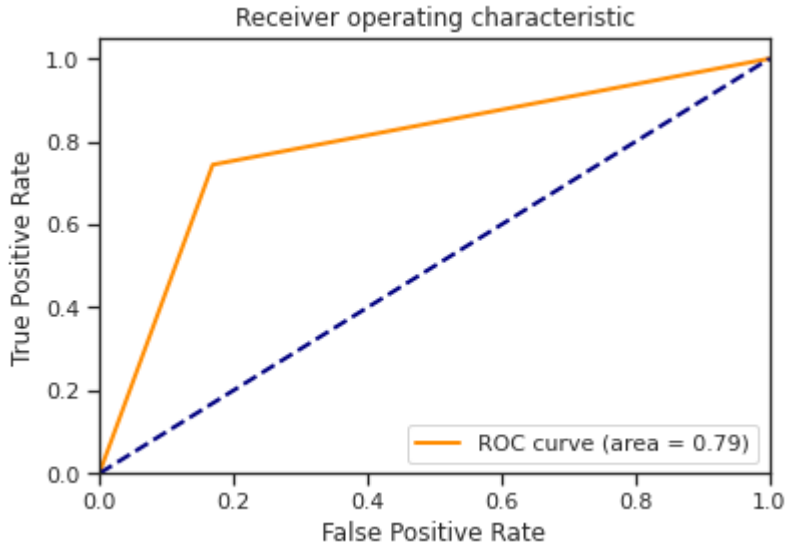

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```

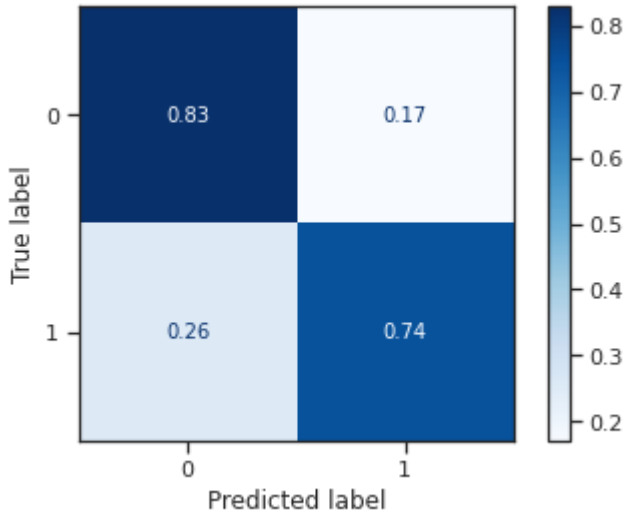





```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```

****

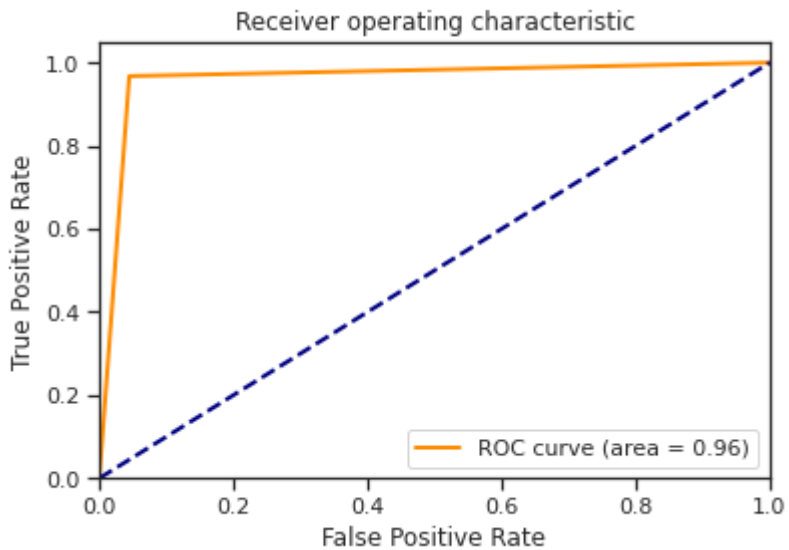


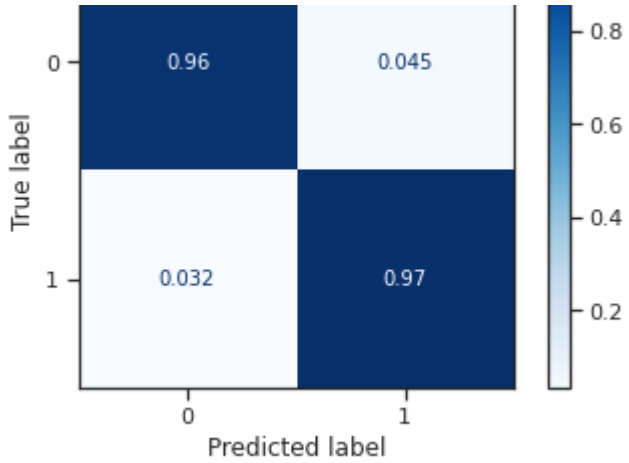


****

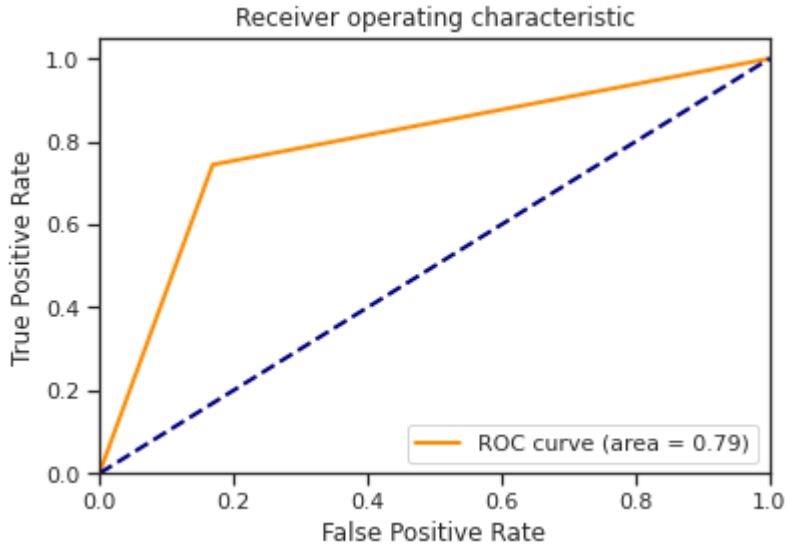
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

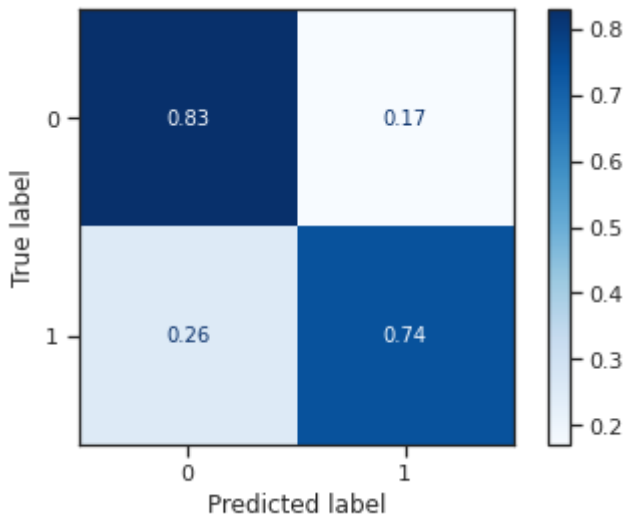
****





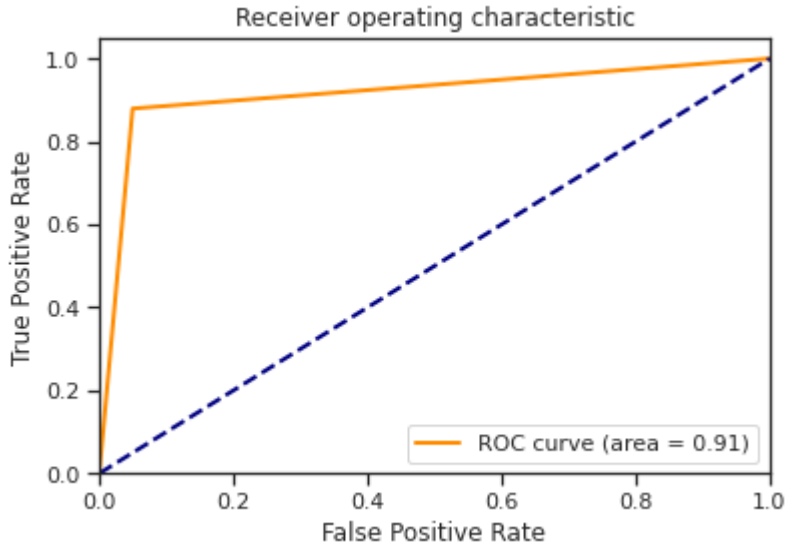

```
******************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
******************************************************
```

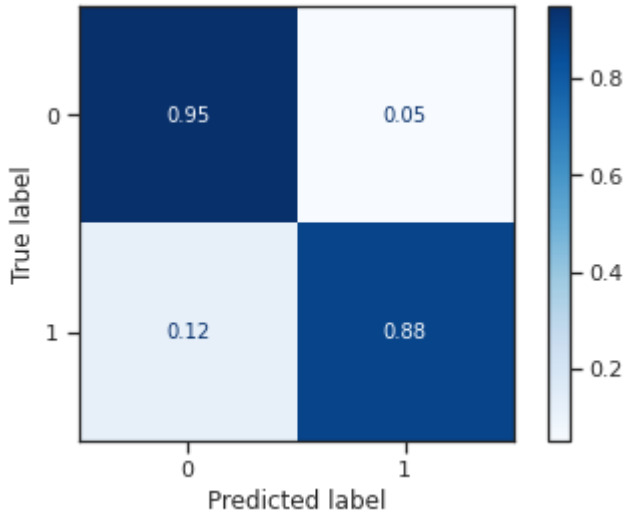




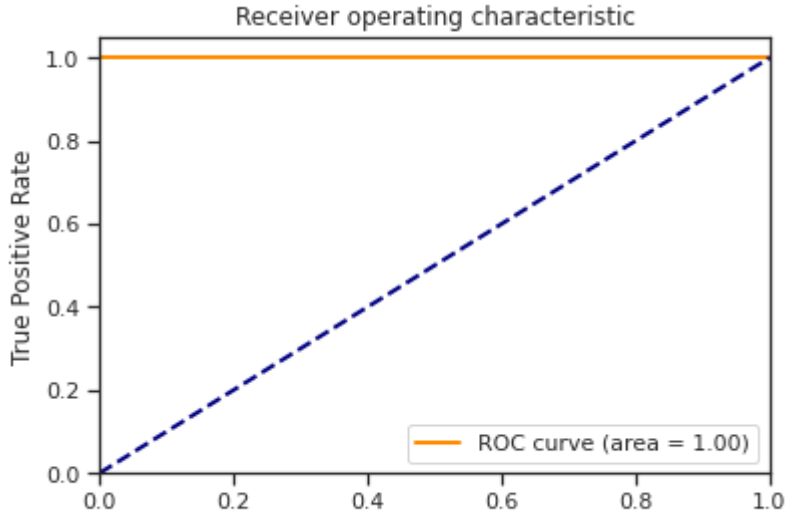

```
******************************************************
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=Fals
                  max_features=1.0, max_samples=1.0, n_estimators=10,
```

```
                       n_jobs=None, oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
**********************************************************
```






```
**********************************************************
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0
                     warm_start=False)
**********************************************************
```

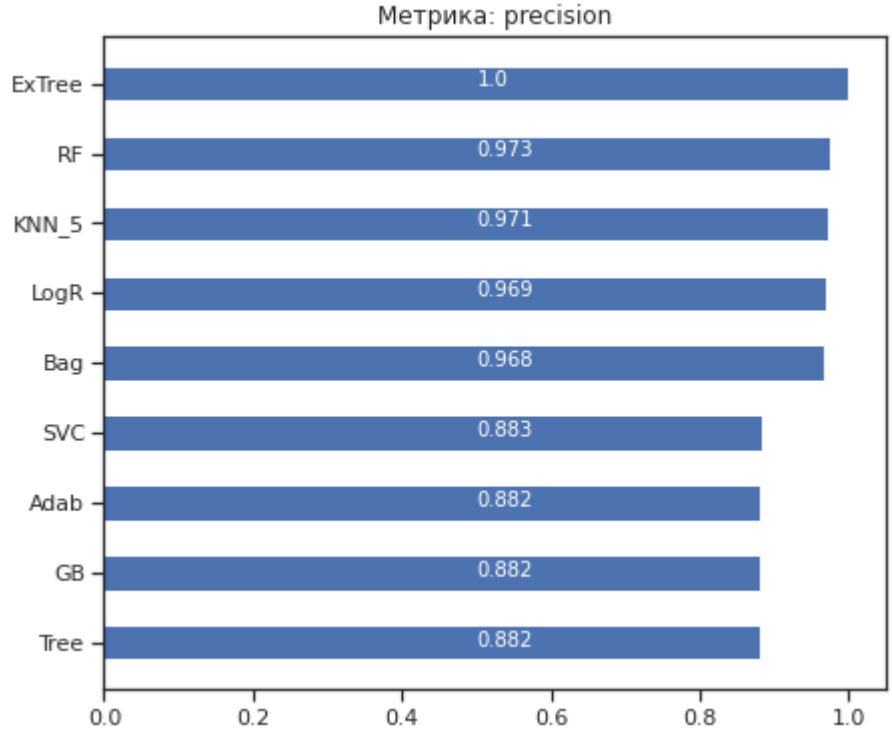


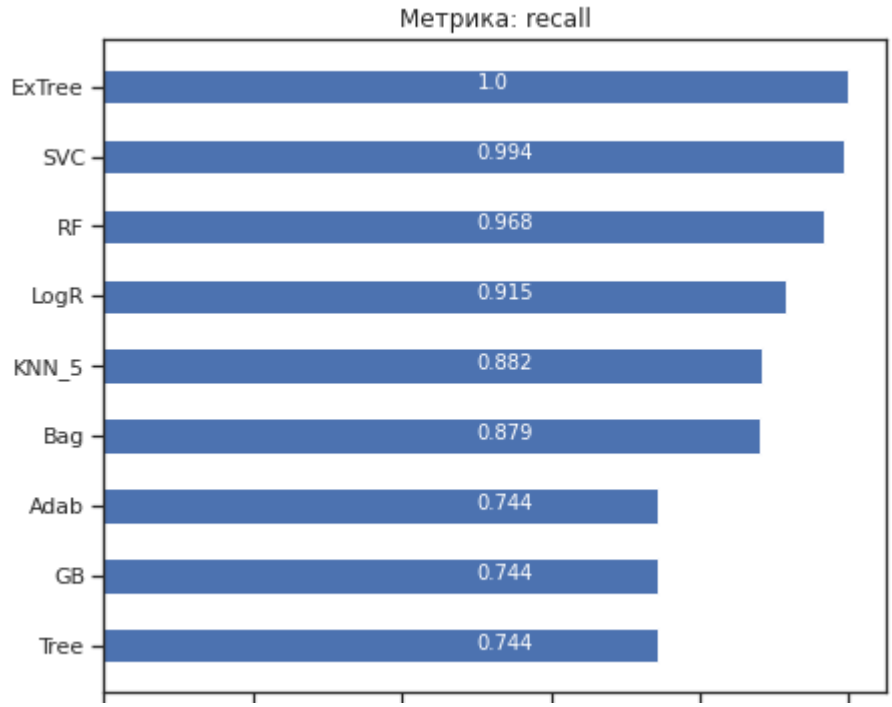
Мы видим, что RandomForestClassifier и ExtraTreesClassifier имеют наилучшую производите
кривой ROC. Мы также увидит меру других показателей по всем классификаторам, так что

```
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

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

```
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```





В соответствии с precision: ExtraTreesClassifier имеет наилучшие значения, потом BaggingCla
Согласно recall: ExtraTreesClassifier имеет наилучшие значения, потом SVC, потом RandomFo
Score»: наилучший результат имеют ExtraTreesClassifier, потом RandomForestClassifier, потом

7)Подбор гиперпараметров для выбранных моделей.

Мы будем изменять количество соседей, которое имеет классификатор k-соседей. и увидет
поиска по сетке

```
n_range = np.array(range(1,22,1))
tuned_parameters = [{'n_neighbors': n_range}]
```

```
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
        18, 19, 20, 21])}]
```

Метрика: roc_auc

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

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8
        18, 19, 20, 21])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
clf_gs.cv_results_
```

```
{'mean_fit_time': array([0.00310426, 0.0018362 , 0.00173659, 0.00171471, 0.002
       0.00186629, 0.00178323, 0.00172935, 0.0017169 , 0.00179191,
       0.00165772, 0.00164089, 0.00162826, 0.00163569, 0.00164137,
       0.00168605, 0.00163426, 0.00164008, 0.00164742, 0.0018301 ,
       0.00204983]),
 'mean_score_time': array([0.0036181 , 0.00236197, 0.00221562, 0.00217724, 0.0
       0.00231009, 0.00225224, 0.00219083, 0.00220208, 0.00230036,
       0.00220351, 0.00217261, 0.0021749 , 0.00221562, 0.00223656,
       0.00219994, 0.00216308, 0.00216784, 0.00219531, 0.00236921,
       0.00279932]),
 'mean_test_score': array([1.        , 1.        , 0.9875    , 0.9875    , 0.98
       0.97638889, 0.97638889, 0.96388889, 0.96388889, 0.96388889,
       0.96388889, 0.975     , 0.975     , 0.9875    , 1.        ,
       0.95      , 0.91666667, 0.86666667, 0.86666667, 0.86666667,
       0.81666667]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
                    17, 18, 19, 20, 21],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14},
  {'n_neighbors': 15},
  {'n_neighbors': 16},
  {'n_neighbors': 17},
  {'n_neighbors': 18},
  {'n_neighbors': 19},
  {'n_neighbors': 20},
```

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

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

```
clf_gs.best_score_
```

```
1.0
```

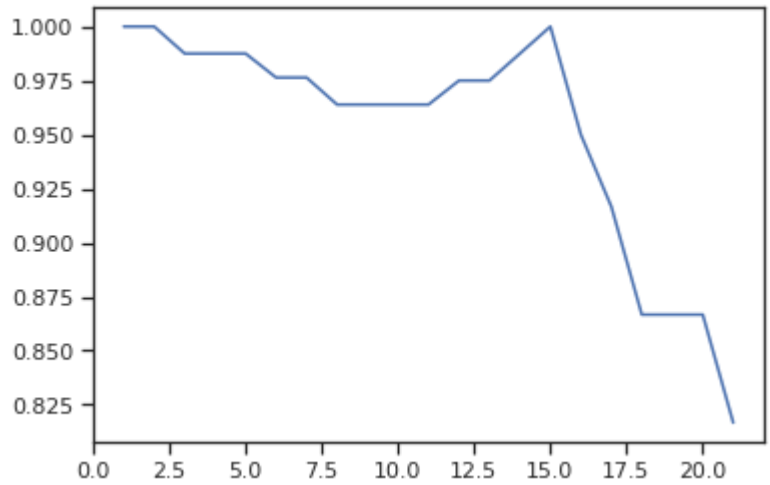
```
# Лучшее значение параметров
clf_gs.best_params_
```

```
{'n_neighbors': 1}
```

5.10739855e-04 1.33457035e-04 5.67525568e-05 2.22727365e-05

```
# 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 0x7f1ef466fc88>]
```




```
clas_models_grid = {'KNN_1':clf_gs.best_estimator_}


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

```
    ****************************************************
    KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                         metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                         weights='uniform')
    ****************************************************
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
clf = ExtraTreesClassifier()
scoring = 'accuracy'
score = cross_val_score(clf, X_train, y_train, cv=k_fold, n_jobs=1, scoring=scoring
print(score)
round(np.mean(score)*100, 2)
```

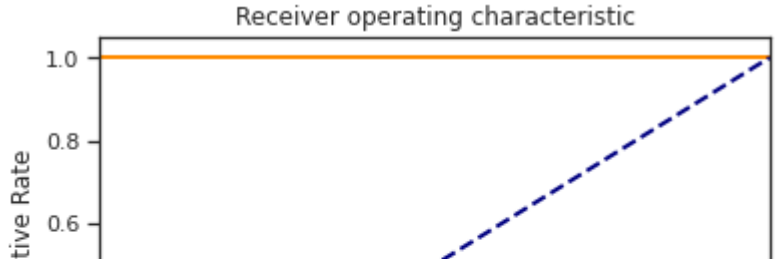
```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
100.0
```

8)Решение задачи классификации

Итак, наконец, мы используем классификатор LogisticRegression и видим его производител тестовыми данными.

```
clas_models_grid = {'Tree':ExtraTreesClassifier()}

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

```
**************************************************
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0
                     warm_start=False)
**************************************************
```



9)Вывод

Таким образом, в нашей модели мы хорошо изучили данные и решили, какой из ее столбцо (после обработки всего объекта и пропущенных данных из него). Затем мы использовали н оценки их работы с нашими данными. Наилучшая производительность была у классификат можем использовать нашу модель для прогнозирования, у кого рак, а у кого нет, и мы може она имеет 100% точность и 100% recall.