

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



**Отчет по Лабораторной Работе № 7**

**«Решение комплексной задачи машинного обучения»**

**По курсу**

**«Методы машинного обучения»**

Выполнила:  
Шаххуд Ф.М.  
Студентка группы ИУ5И-22М

**Москва, 2020**



```

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()

```

```

data = pd.read_csv('/content/drive/My Drive/titanic/train.csv')
test = pd.read_csv('/content/drive/My Drive/titanic/test.csv')
data.head()

```

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr. Owen Harris	male	22
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38
2	3	1	3	Heikkinen, Miss. Laina	female	26
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35
4	5	0	3	Allen, Mr. William Henry	male	35

```
test.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Par
0	892	3	Kelly, Mr. James	male	34.5	0	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	
3	895	3	Wirz, Mr. Albert	male	27.0	0	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	

```

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

```

```
Всего строк: 891
```

```
data.shape
```

```
(891, 12)
```

```
test.shape
```

```
(418, 11)
```

```
# Список колонок
data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
test.columns
```

```
Index(['PassengerId', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
data.dtypes
```

```
PassengerId    int64
Survived        int64
Pclass          int64
Name            object
Sex             object
Age            float64
SibSp           int64
Parch           int64
Ticket          object
Fare            float64
Cabin           object
Embarked        object
dtype: object
```

```
test.dtypes
```

```
PassengerId    int64
Pclass          int64
Name            object
Sex             object
Age            float64
SibSp           int64
Parch           int64
Ticket          object
Fare            float64
Cabin           object
Embarked        object
dtype: object
```

## ▼ Анализ и заполнение пропусков в данных

## ▼ Обработка пропусков в числовых данных

```
data.isnull().sum()
```

```
data.isnull().sum()
```

```

↳ PassengerId      0
   Survived         0
   Pclass           0
   Name             0
   Sex              0
   Age             177
   SibSp            0
   Parch            0
   Ticket           0
   Fare             0
   Cabin           687
   Embarked         2
   dtype: int64

```

```
test.isnull().sum()
```

```

↳ PassengerId      0
   Pclass           0
   Name             0
   Sex              0
   Age             86
   SibSp            0
   Parch            0
   Ticket           0
   Fare             1
   Cabin           327
   Embarked         0
   dtype: int64

```

```
# Выберем числовые колонки с пропущенными значениями
```

```
# Цикл по колонкам датасета
```

```
num_cols = []
```

```
for col in data.columns:
```

```
    # Количество пустых значений
```

```
    temp_null_count = data[data[col].isnull()].shape[0]
```

```
    dt = str(data[col].dtype)
```

```
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
```

```
        num_cols.append(col)
```

```
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
```

```
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

```
↳ Колонка Age. Тип данных float64. Количество пустых значений 177, 19.87%.
```

```
# Выберем числовые колонки с пропущенными значениями
```

```
# Цикл по колонкам датасета
```

```
num_cols = []
```

```
for col in test.columns:
```

```
    # Количество пустых значений
```

```
    temp_null_count = test[test[col].isnull()].shape[0]
```

```
    dt = str(test[col].dtype)
```

```
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
```

```
        num_cols.append(col)
```

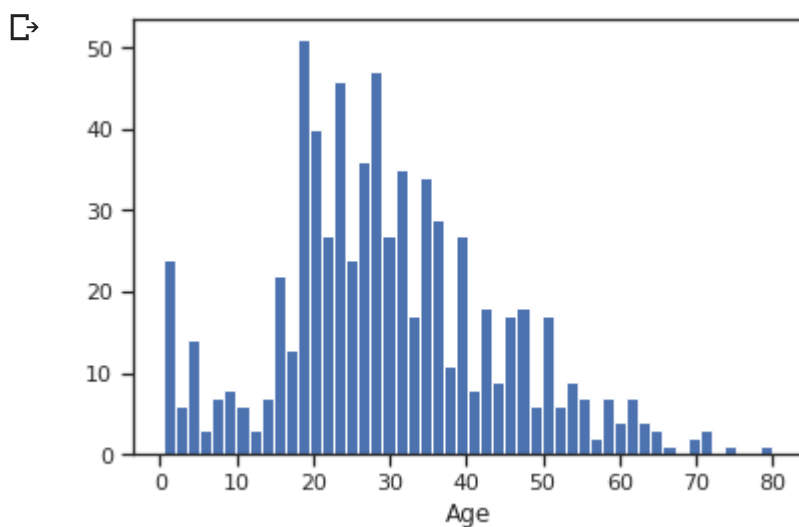
```
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
```

```
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

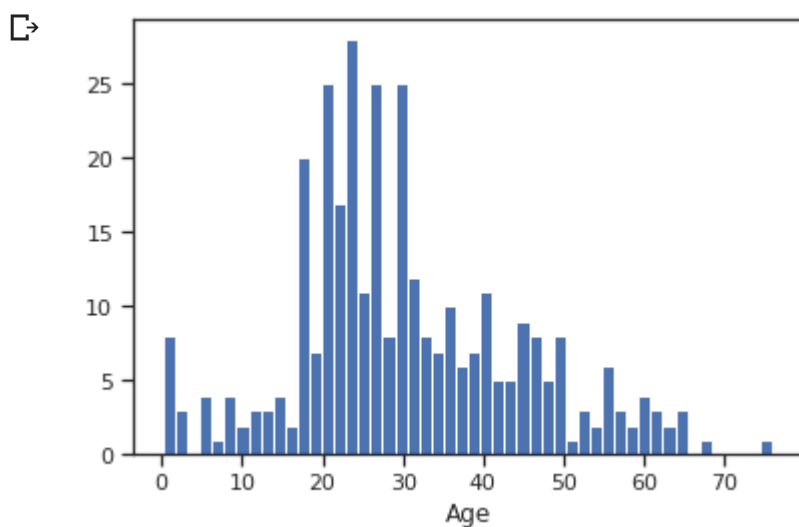
Колонка Age. Тип данных float64. Количество пустых значений 86, 9.65%.  
 Колонка Fare. Тип данных float64. Количество пустых значений 1, 0.11%.

```
# Фильтр по колонкам с пропущенными значениями
data_num = data['Age']
test_num= test['Age']
test_fare=test['Fare']
```

```
# Гистограмма по признакам
plt.hist(data_num, 50)
plt.xlabel('Age')
plt.show()
```



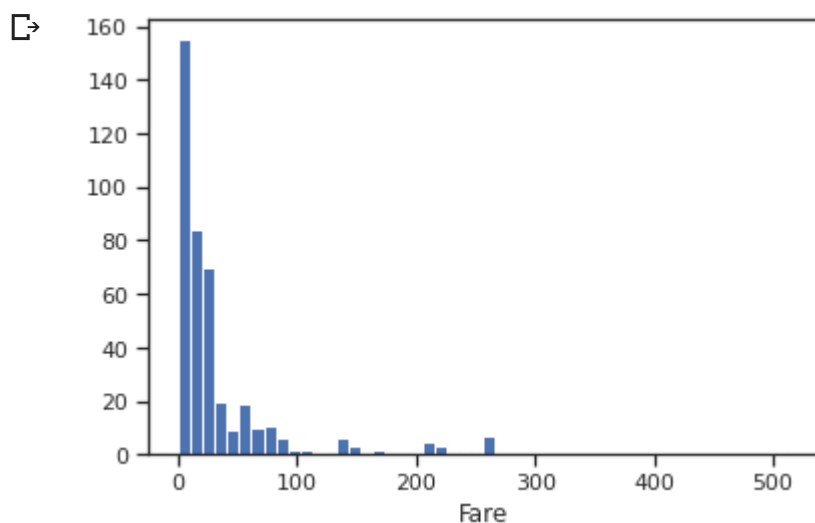
```
plt.hist(test_num, 50)
plt.xlabel('Age')
plt.show()
```



лучший способ уместить недостающие данные в Возрасте и Тарифе - заполнить их медиану  
 невозможно заполнить их 0.

```
data["Age"].fillna(data.groupby("Pclass")["Age"].transform("median"), inplace=True)
test["Age"].fillna(test.groupby("Pclass")["Age"].transform("median"), inplace=True)
```

```
plt.hist(test_fare, 50)
plt.xlabel('Fare')
plt.show()
```



```
test["Fare"].fillna(test.groupby("Pclass")["Fare"].transform("median"), inplace=True)
```

Double-click (or enter) to edit

```
data.isnull().sum()
```

```
PassengerId      0
Survived          0
Pclass            0
Name              0
Sex               0
Age              0
SibSp             0
Parch            0
Ticket           0
Fare              0
Cabin            687
Embarked          2
dtype: int64
```

```
test.isnull().sum()
```

```
PassengerId      0
Pclass            0
Name              0
Sex               0
Age              0
SibSp             0
Parch            0
Ticket           0
Fare              0
Cabin            327
Embarked          0
dtype: int64
```

## ▼ Обработка пропусков в категориальных данных

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

Колонка Cabin. Тип данных object. Количество пустых значений 687, 77.1%.  
Колонка Embarked. Тип данных object. Количество пустых значений 2, 0.22%.

```
cat_colstest = []
for col in test.columns:
    # Количество пустых значений
    temp_null_count = test[test[col].isnull()].shape[0]
    dt = str(test[col].dtype)
    if temp_null_count>0 and (dt=='object'):
        cat_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

Колонка Cabin. Тип данных object. Количество пустых значений 327, 36.7%.

```
data_col=data[cat_cols]
data_col.head()
```

Колонка Cabin Embarked Cabin

	Cabin	Embarked	Cabin
0	NaN	S	NaN
1	C85	C	C85
2	NaN	S	NaN
3	C123	S	C123
4	NaN	S	NaN

```
data_num_Embarked = data_col[['Embarked']]
data_num_Cabin = data_col[['Cabin']]
test_num_Cabin = test[['Cabin']]
```

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

array(['S', 'C', 'Q', nan], dtype=object)



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

```
↳ array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
        'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
        'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
        'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
        'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
        'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
        'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
        'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
        'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
        'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
        'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
        'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
        'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
        'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
        'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
        'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
        'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
        'C148'], dtype=object)
```

```
data['Cabin'].count()
```

```
↳ 204
```

```
print(data[data['Cabin'].str.startswith('A')==True].Age.count())
print(data[data['Cabin'].str.startswith('B')==True].Age.count())
print(data[data['Cabin'].str.startswith('C')==True].Age.count())
print(data[data['Cabin'].str.startswith('D')==True].Age.count())
print(data[data['Cabin'].str.startswith('E')==True].Age.count())
print(data[data['Cabin'].str.startswith('F')==True].Age.count())
```

```
↳ 15
   47
   59
   33
   32
   13
```

Мы запишем признак Embarked, которые отсутствуют, с наиболее частым значением в наборе

```
imp_num = SimpleImputer(strategy='most_frequent')
imp_num_fit = imp_num.fit_transform(data_num_Embarked)
```

```
data.Embarked=imp_num_fit
```

```
data.Cabin.value_counts()
```

```
↳
```

```

B96 B98      4
C23 C25 C27   4
G6           4
E101         3
F2           3
...

```

Так как название салона связано с классом пассажира. Мы будем заполнять пустые значения каждого класса отдельно. Вот почему мы сначала сгруппируемся по классу Pclass, а затем

```

data["Cabin"].fillna(
test["Cabin"].fillna(

```

```

data["Cabin"].fillna(data.groupby("Pclass")["Cabin"].transform(lambda x:x.value_counts().max()))
test["Cabin"].fillna(test.groupby("Pclass")["Cabin"].transform(lambda x:x.value_counts().max()))

```

Убедиться, что у нас нет пропущенных значений

```
data.isnull().sum()
```

```

PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age            0
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin           0
Embarked        0
dtype: int64

```

```
test.isnull().sum()
```

```

PassengerId    0
Pclass          0
Name            0
Sex             0
Age            0
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin           0
Embarked        0
dtype: int64

```

## ▼ Построение графиков

```

def bar_chart(feature):
    survived = data[data['Survived']==1][feature].value_counts()
    dead = data[data['Survived']==0][feature].value_counts()
    df = pd.DataFrame([survived,dead])
    df.index = ['survived', 'dead']

```

```
df.index = [ 'Survived' , 'Dead' ]  
df.plot(kind='bar',stacked=True, figsize=(10,5))
```

```
bar_chart('Sex')
```

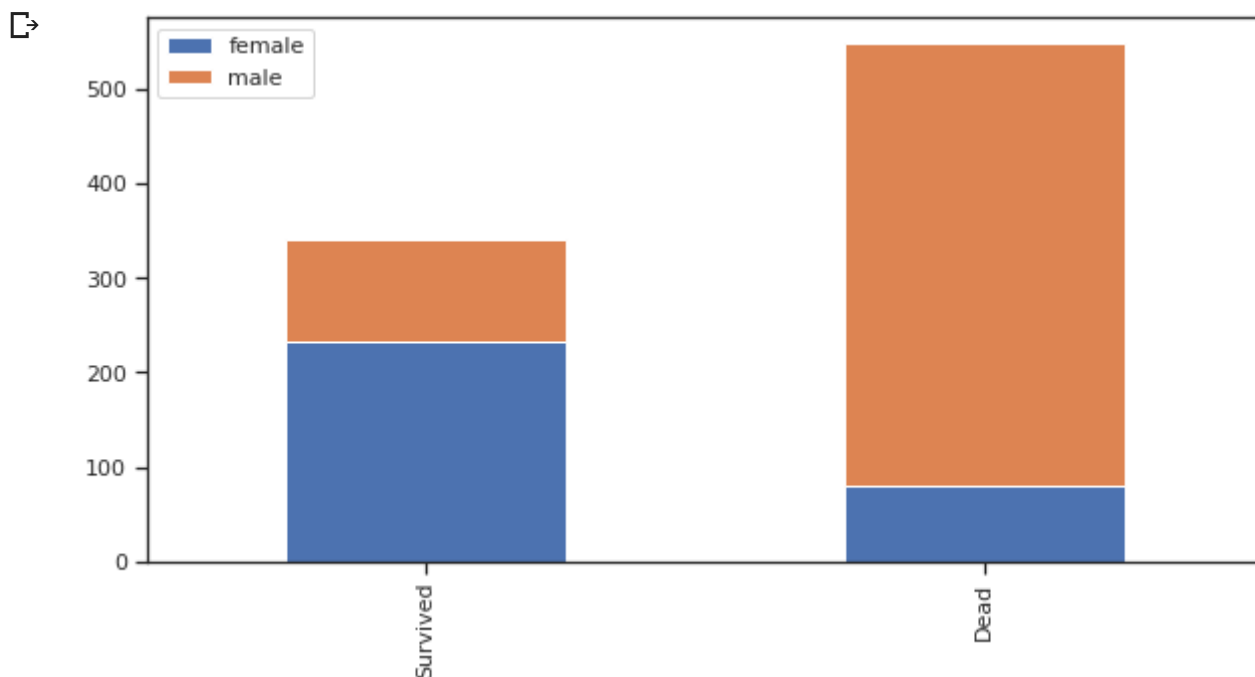


Диаграмма подтверждает, что женщины выжили чаще, чем мужчины

```
bar_chart('Pclass')
```

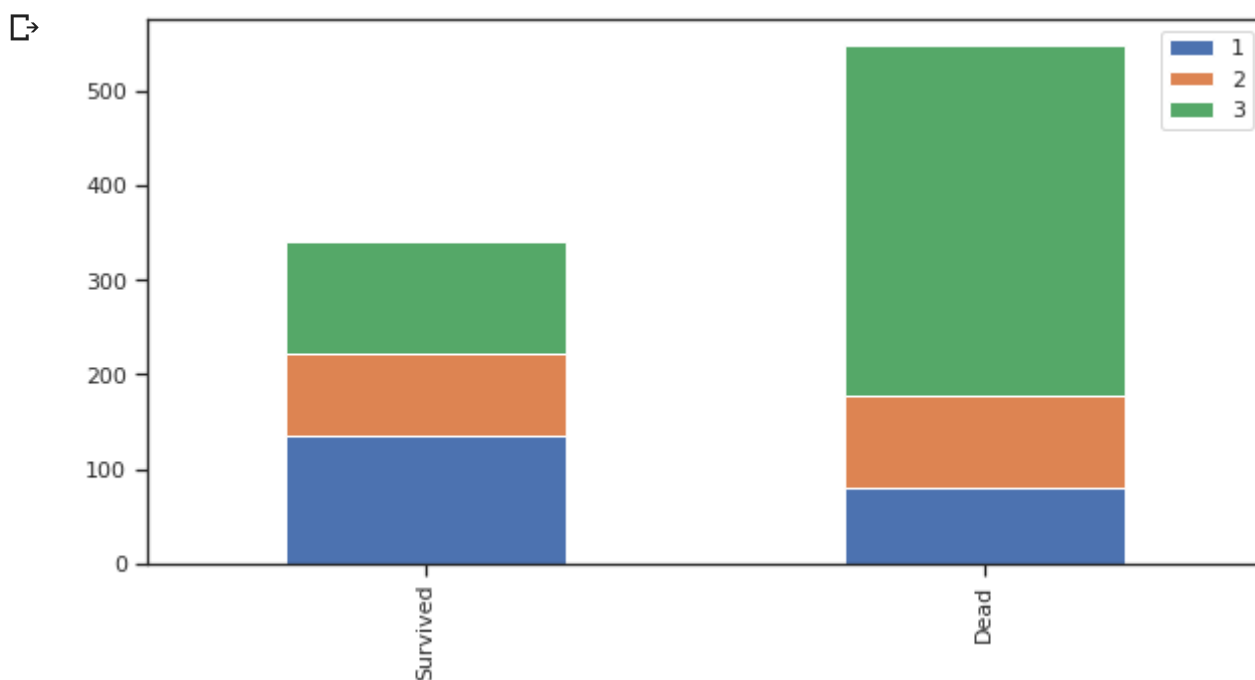


Диаграмма подтверждает, что 1-й класс выжил чаще других.

Диаграмма подтверждает, что 3-й класс скорее мертв, чем другие.

```
bar_chart('Embarked')
```

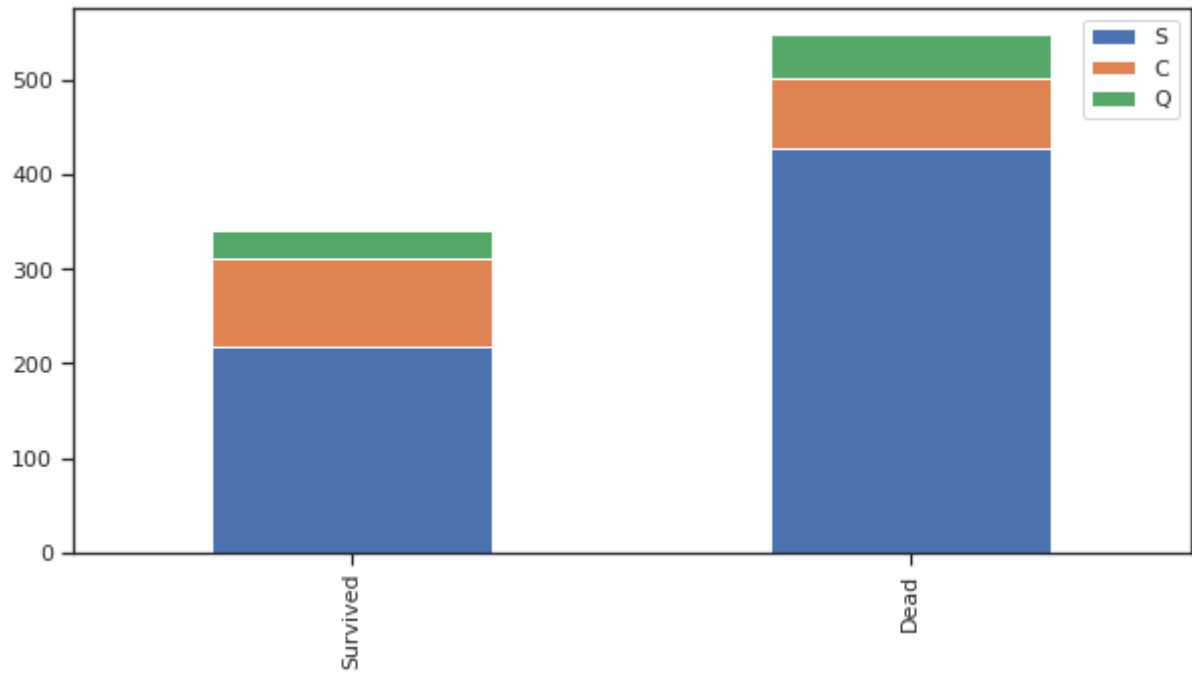


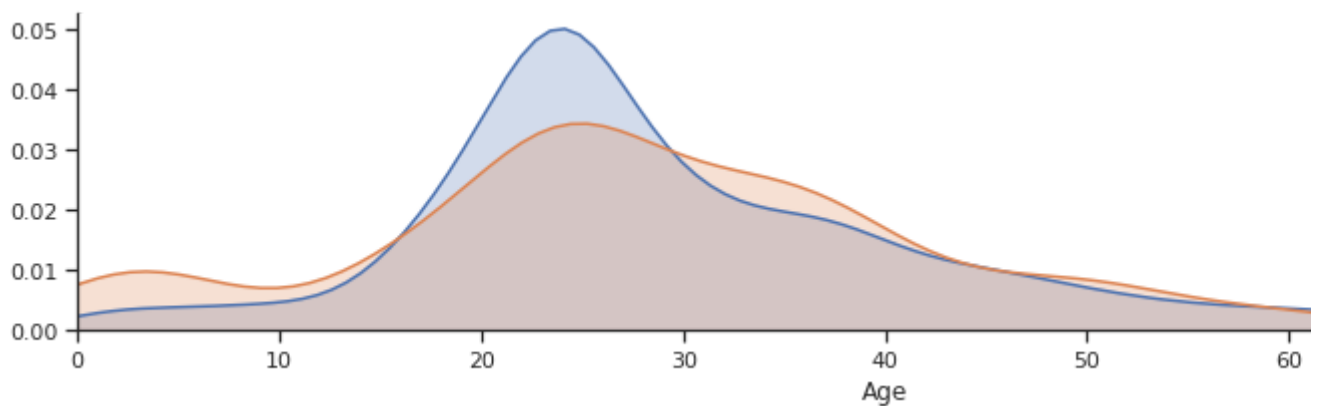
Диаграмма подтверждает, что человек, находящийся на борту C, несколько более вероятно

Диаграмма подтверждает, что человек, находящийся на борту Q, скорее всего мертв.

Диаграмма подтверждает, что человек, находящийся на борту S, скорее всего мертв.

```
facet = sns.FacetGrid(data, hue="Survived", aspect=4)
facet.map(sns.kdeplot, 'Age', shade= True)
facet.set(xlim=(0, data['Age'].max()))
facet.add_legend()

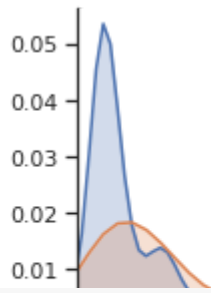
plt.show()
```



```
facet = sns.FacetGrid(data, hue="Survived", aspect=4)
facet.map(sns.kdeplot, 'Fare', shade= True)
facet.set(xlim=(0, data['Fare'].max()))
facet.add_legend()

plt.show()
```





```
sns.pairplot(data[['Pclass', 'Age', 'Fare', 'Parch', 'SibSp', 'Survived']], hue="Survived"
```



<seaborn.axisgrid.PairGrid at 0x7fb4be84dda0>

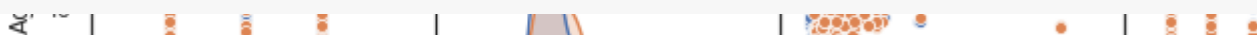


Имя и номер билета не имеют ничего общего с выживанием человека. Вот почему мы можем увидеть такой результат. Но прежде чем мы это сделаем, мы сможем увидеть все специальные детали, касающиеся «Имя».



```
train_test_data = [data, test] # combining train and test dataset
```

```
for dataset in train_test_data:
    dataset['Title'] = dataset['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```



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

```
Mr      517
Miss    182
Mrs     125
Master   40
Dr        7
Rev        6
Major      2
Mlle       2
Col        2
Lady       1
Countess   1
Capt      1
Mme        1
Jonkheer   1
Sir         1
Don         1
Ms          1
Name: Title, dtype: int64
```



```
test['Title'].value_counts()
```

```
Mr      240
Miss     78
Mrs      72
Master   21
Rev        2
Col        2
Ms         1
Dona       1
Dr          1
Name: Title, dtype: int64
```

Поскольку слова «Мистер», «Мисс» и «Миссис» отражают какую-то социальную ценность в тесте, так и в данных, мы добавим столбец, значение которого соответствует этим словам, которые не содержат ни одного из этих слов, будут иметь значение 3.

```
title_mapping = {"Mr": 0, "Miss": 1, "Mrs": 2,
                 "Master": 3, "Dr": 3, "Rev": 3, "Col": 3, "Major": 3, "Mlle": 3, "C
                 "Ms": 3, "Lady": 3, "Jonkheer": 3, "Don": 3, "Dona" : 3, "Mme": 3,
for dataset in train_test_data:
    dataset['Title'] = dataset['Title'].map(title_mapping)
```

## ▼ Кодирование категориальных признаков

Мы будем использовать простую кодировку для наших переменных объекта.

```
LE=LabelEncoder()
```

```
data_Sex=LE.fit_transform(data['Sex'])
data['Sex']=data_Sex
```

```
data_cabin=LE.fit_transform(data['Cabin'])
data['Cabin']=data_cabin
```

```
embarked_mapping = {"S": 0, "C": 1, "Q": 2}
for dataset in [data,test]:
    dataset['Embarked'] = dataset['Embarked'].map(embarked_mapping)
```

```
test_Sex=LE.fit_transform(test['Sex'])
test['Sex']=test_Sex
```

```
test_cabin=LE.fit_transform(test['Cabin'])
test['Cabin']=test_cabin
```

```
test_fare=LE.fit_transform(test['Fare'])
test['Fare']=test_fare
```

## ▼ Удаление столбцов имени и номера билета.

```
data=data.drop(['Name','Ticket'], axis=1)
data.head()
```



```
PassengerId Survived Pclass Sex Age SibSp Parch Fare Cabin Embarked
```

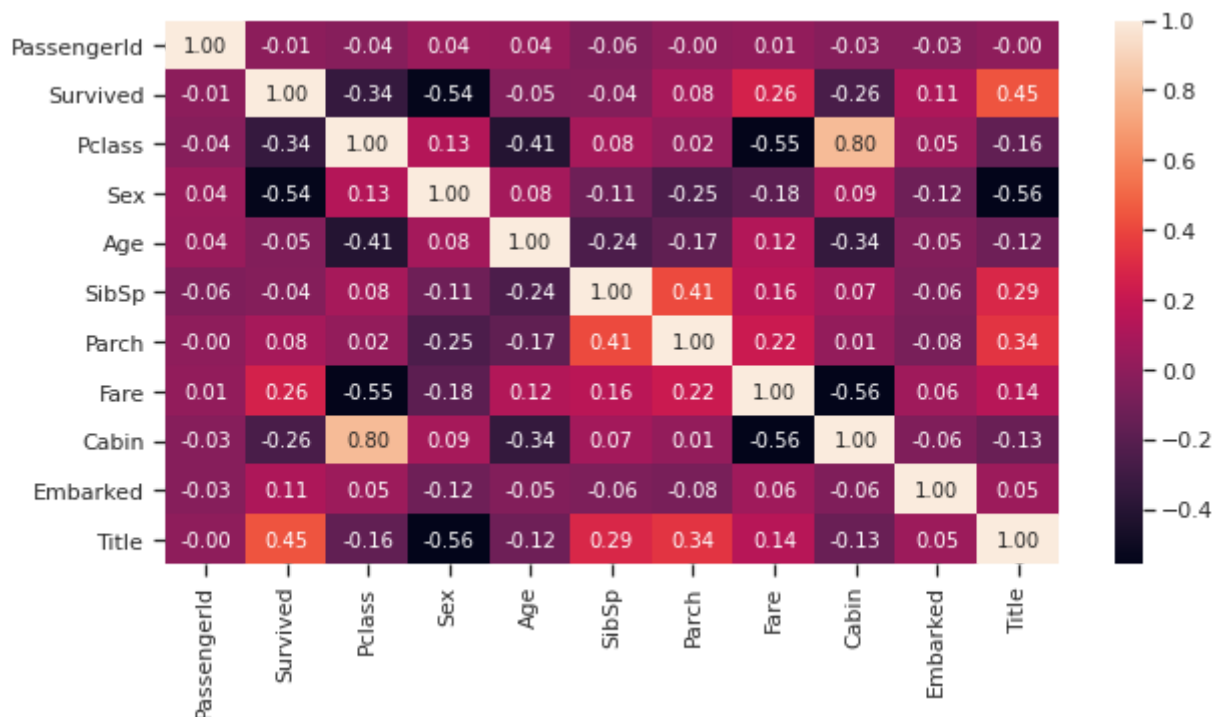
```
test=test.drop(['Name','Ticket'], axis=1)
test.head()
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	Title
0	892	3	1	34.5	0	0	24	71	2	0
1	893	3	0	47.0	1	0	5	71	0	2
2	894	2	1	62.0	0	0	41	74	2	0
3	895	3	1	27.0	0	0	34	71	0	0
4	896	3	0	22.0	1	1	46	71	0	2

## ▼ Проведение корреляционного анализа данных.

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data.corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb4be29f908>
```



Итак, мы видим, что идентификатор пассажира не должен быть включен в нашу модель, поскольку он не связан с переменной Survival.

Пол и заголовок (столбец, который мы добавили для некоторых признаков столбца «Имя») также Cabin, Pclass и Fare хорошо связаны с ним (поскольку он определяет класс пассажира). SibSp и Parch и возраст не очень связаны. Мы также можем пренебречь этими столбцами.



## ▼ Выбор метрик для последующей оценки качества моделей

Мы будем использовать метрику для точности, `f1_score`, отзыва и `roc_auc`. И мы увидим кривые.

```
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)
        # 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 признаков, подходящих для построения моделей

Как мы уже говорили, мы будем использовать Sex, Title, Pclass, Cabin и Fare за модель. Все с колонкой Survived.

```
X_train, X_test, y_train, y_test = train_test_split(data[['Pclass', 'Sex', 'Title',
    'Fare', 'Cabin']], data['Survived'], test_size=0.2, random_state=1)
```

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

```
↳ ((712, 5), (179, 5), (712,), (179,))
```

## ▼ Выбор моделей для решения задачи

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

```
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
clas_models = {'KNN_5':KNeighborsClassifier(n_neighbors=5),
    'Tree':DecisionTreeClassifier(),
    'RF':RandomForestClassifier(),
    'GB':GradientBoostingClassifier(),
    'Bag':BaggingClassifier(),
    'ExTree':ExtraTreesClassifier(),
    'Adab':AdaBoostClassifier()}
```

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

```
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()
```

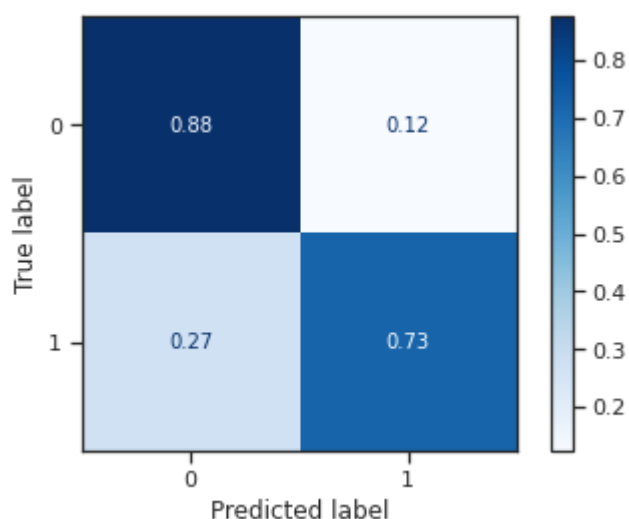
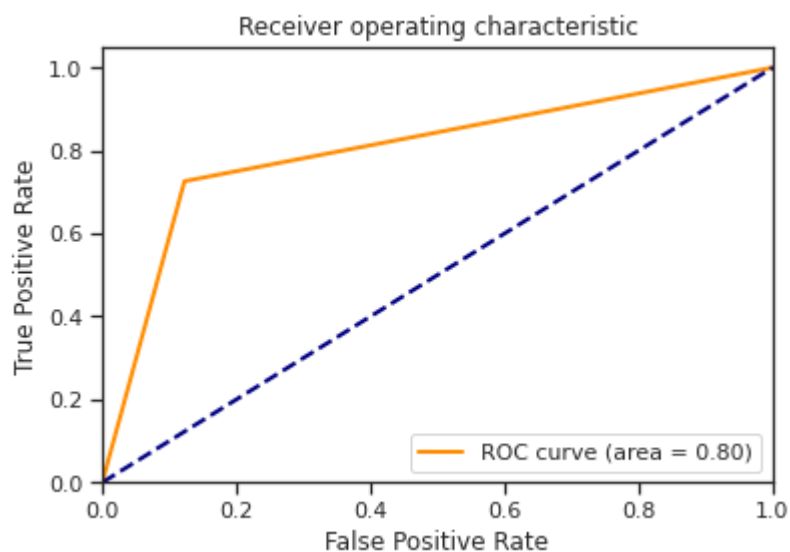
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```



```
*****
```

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

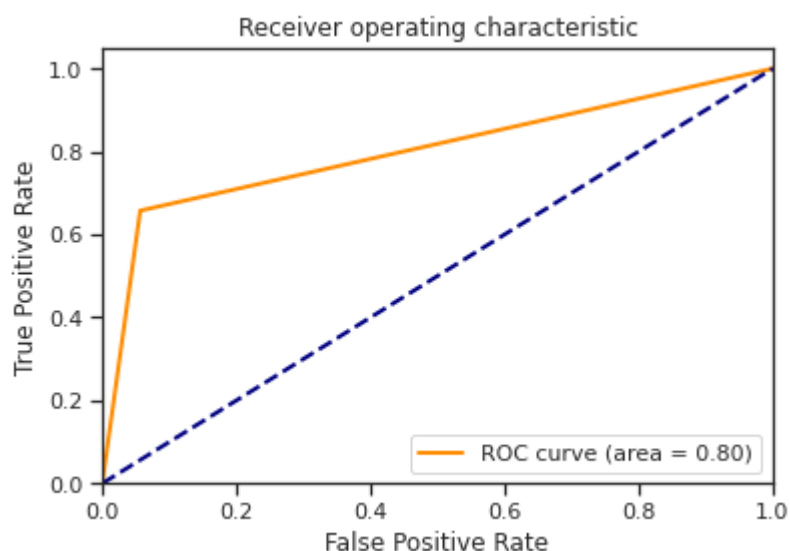
```
*****
```

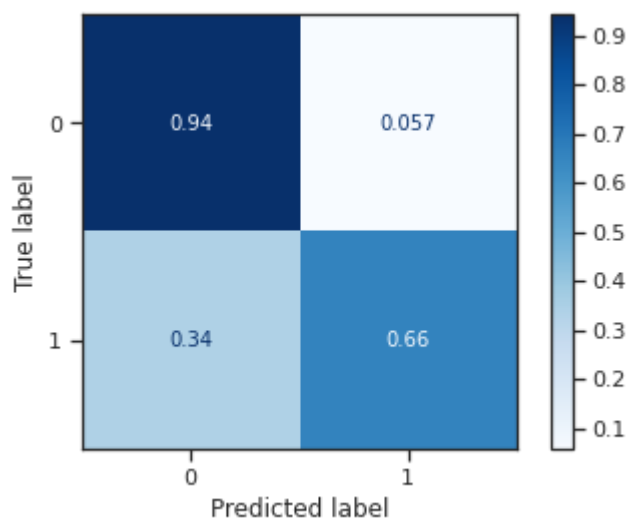


```
*****
```

```
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')
```

```
*****
```

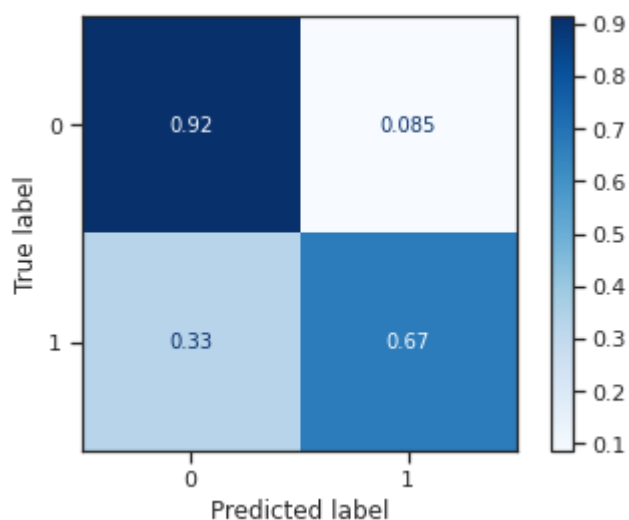
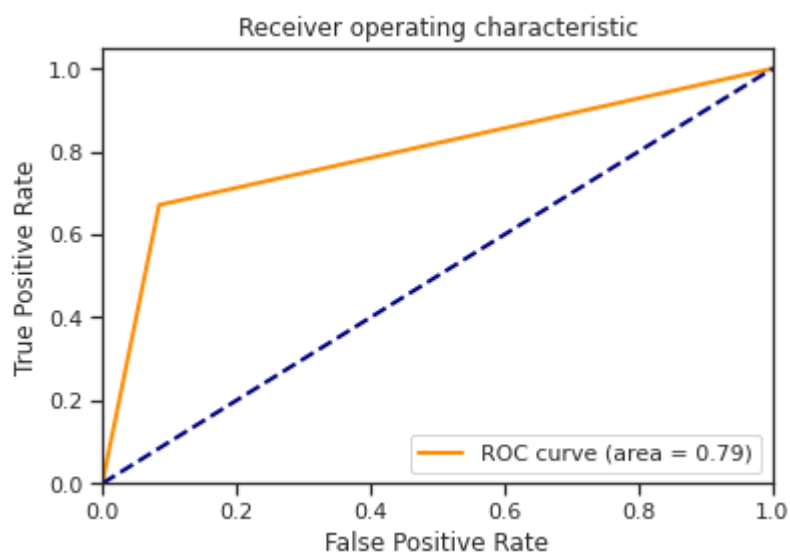




\*\*\*\*\*

```
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)
```

\*\*\*\*\*



\*\*\*\*\*

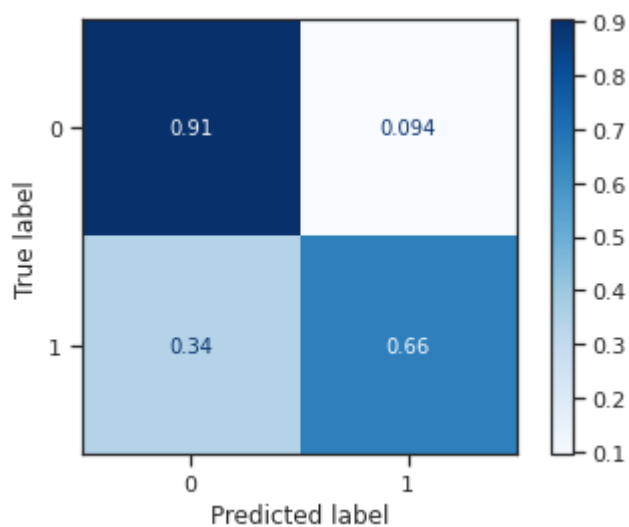
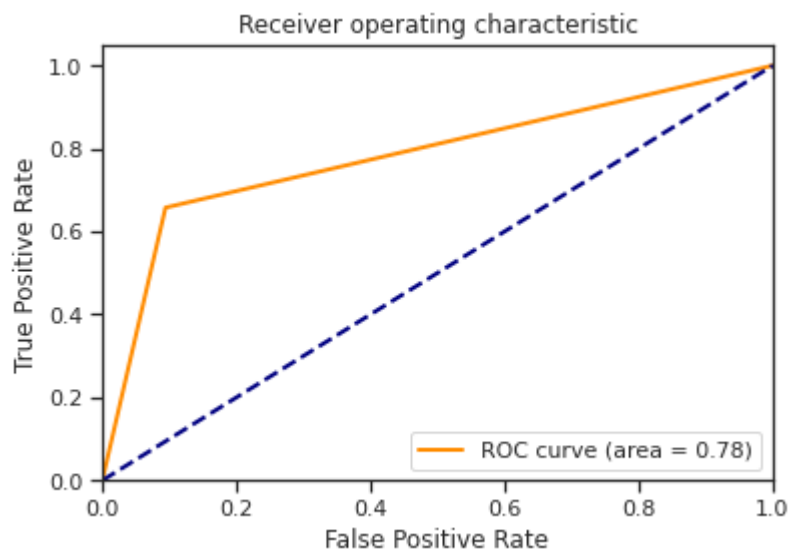
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
```

```

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)

```

\*\*\*\*\*



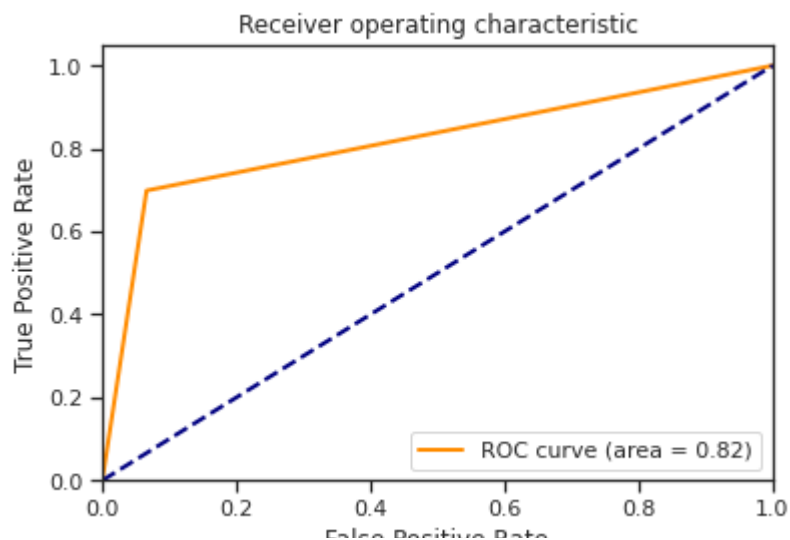
\*\*\*\*\*

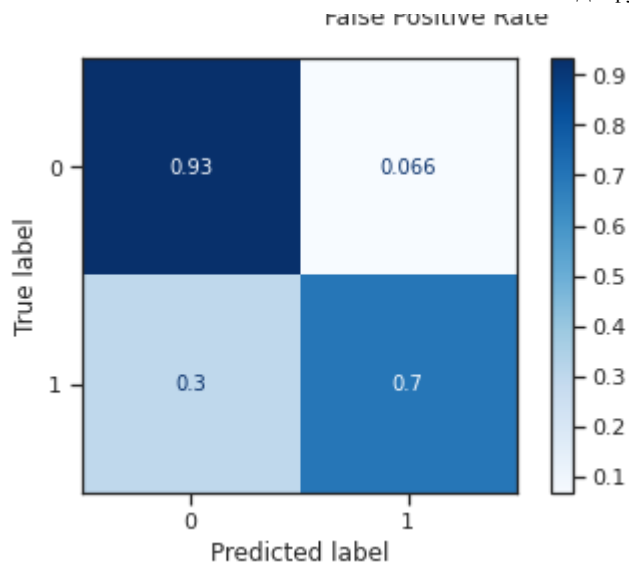
```

BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
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)

```

\*\*\*\*\*

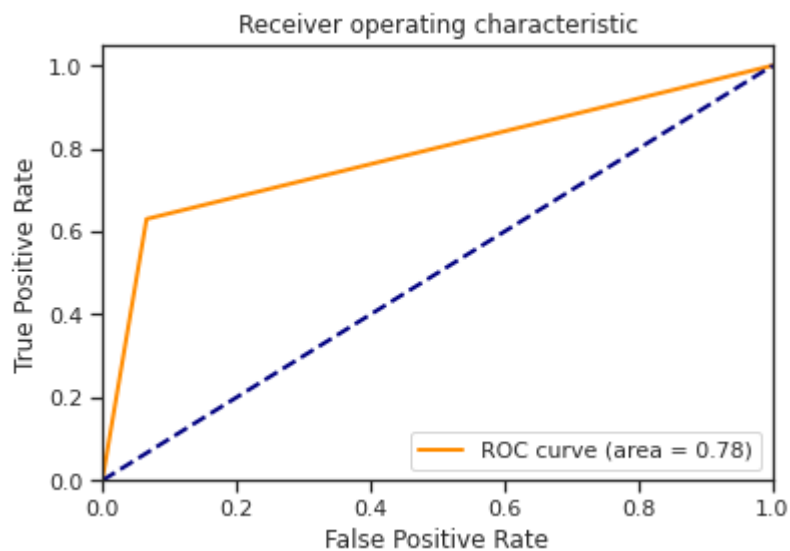




\*\*\*\*\*

```
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 классификатор мешков имеет наилучшую производительность в зависимости от других показателей по всем классификаторам, так что будет легче сравнивать

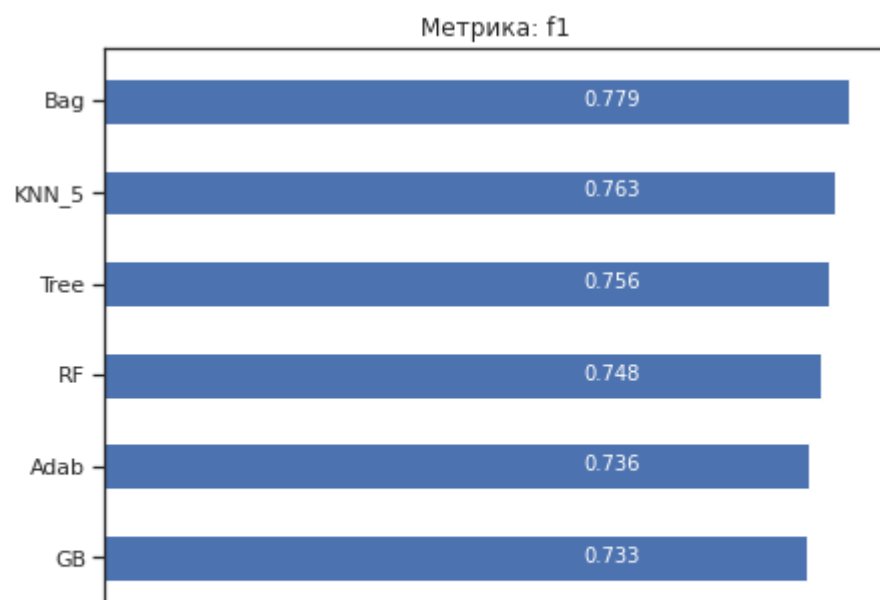
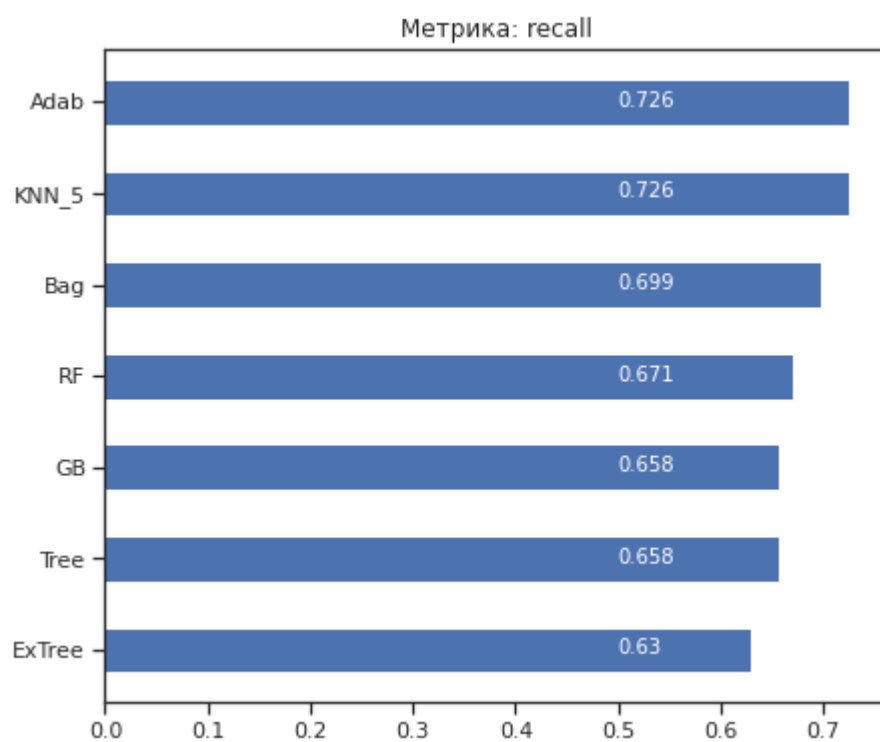
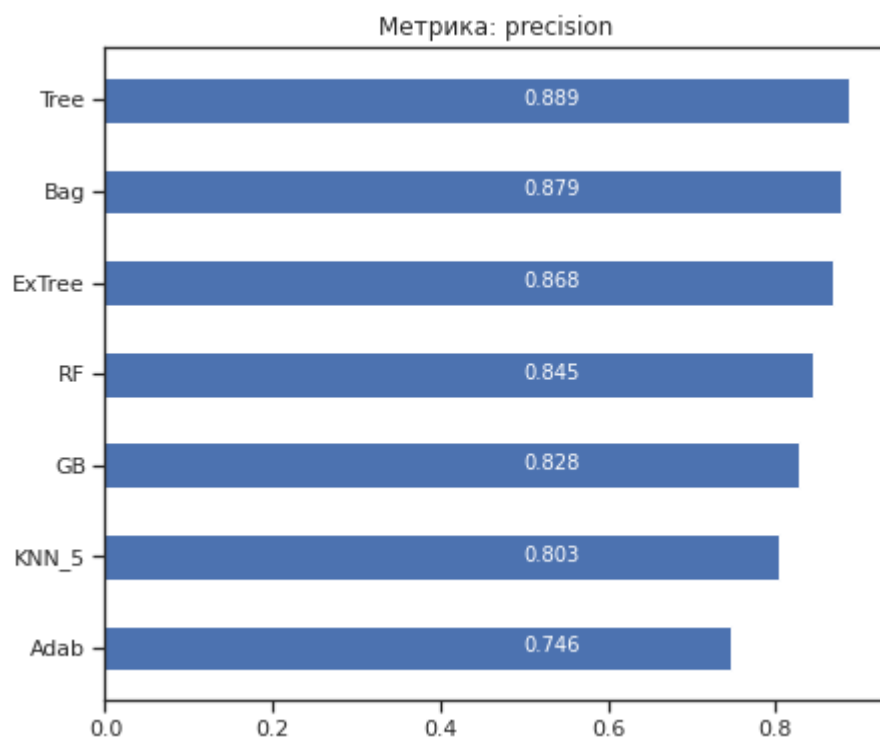


```
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

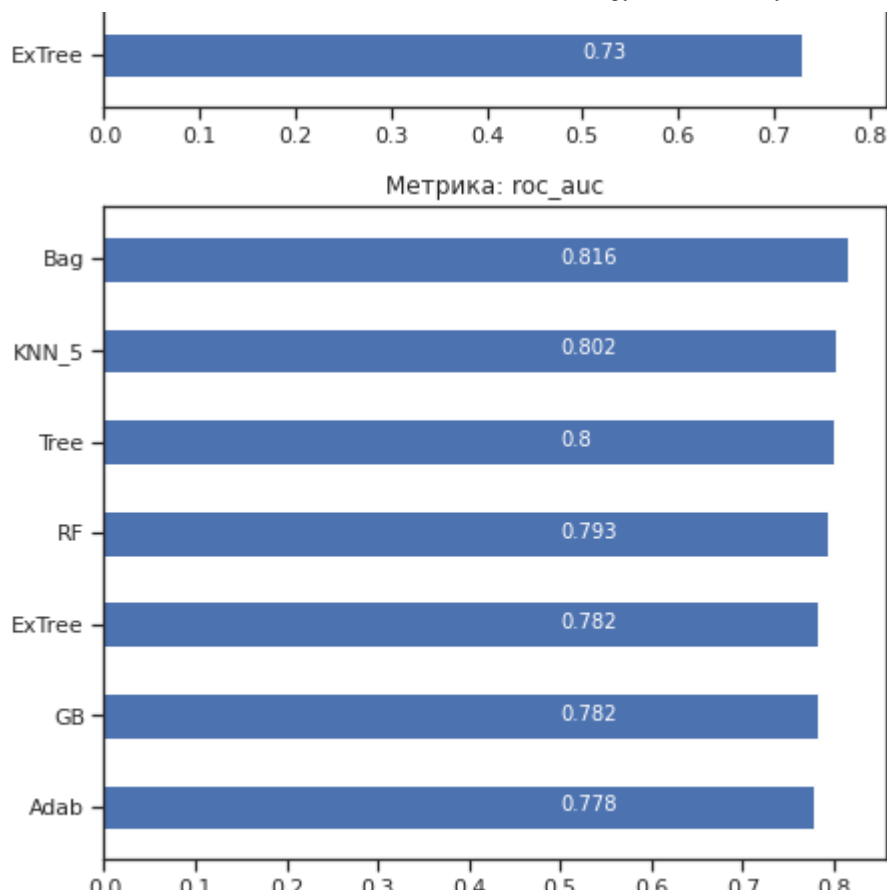
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

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

```
↳
```







В соответствии с precision: дерево решений и классификаторы мешков имеют наилучшие значения.  
 Согласно recall: классификатор адаптера и классификаторы k-соседей имеют лучшие значения.  
 Согласно метрике «f1-Score»: наилучший результат имеют классификатор мешков и классификаторы k-соседей.

## ▼ Подбор гиперпараметров для выбранных моделей.

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

```
x_train.shape
```

```
(712, 5)
```

```
n_range = np.array(range(1,500,4))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  5,  9, 13, 17, 21, 25, 29, 33, 37, 41,
                        53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97, 101,
                        105, 109, 113, 117, 121, 125, 129, 133, 137, 141, 145, 149, 153,
                        157, 161, 165, 169, 173, 177, 181, 185, 189, 193, 197, 201, 205,
                        209, 213, 217, 221, 225, 229, 233, 237, 241, 245, 249, 253, 257,
                        261, 265, 269, 273, 277, 281, 285, 289, 293, 297, 301, 305, 309,
                        313, 317, 321, 325, 329, 333, 337, 341, 345, 349, 353, 357, 361,
                        365, 369, 373, 377, 381, 385, 389, 393, 397, 401, 405, 409, 413,
                        417, 421, 425, 429, 433, 437, 441, 445, 449, 453, 457, 461, 465,
                        469, 473, 477, 481, 485, 489, 493, 497])}]
```

```
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,  5,  9, 13, 17, 21,
53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97, 101,
105, 109, 113, 117, 121, 125, 129, 1...
209, 213, 217, 221, 225, 229, 233, 237, 241, 245, 249, 253, 257,
261, 265, 269, 273, 277, 281, 285, 289, 293, 297, 301, 305, 309,
313, 317, 321, 325, 329, 333, 337, 341, 345, 349, 353, 357, 361,
365, 369, 373, 377, 381, 385, 389, 393, 397, 401, 405, 409, 413,
417, 421, 425, 429, 433, 437, 441, 445, 449, 453, 457, 461, 465,
469, 473, 477, 481, 485, 489, 493, 497])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
clf_gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=13, p=2,
                    weights='uniform')
```

```
clf_gs.best_params_
```

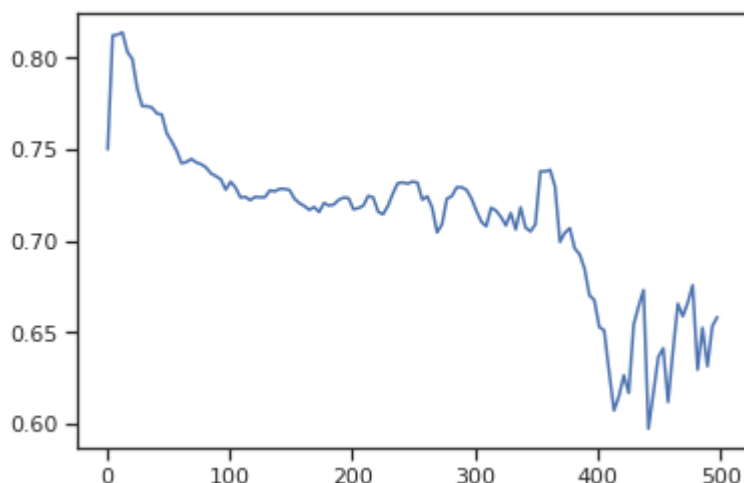
```
{'n_neighbors': 13}
```

Таким образом, наилучшее число для классификатора k-соседей равно 13. Мы снова вычислим, будут ли они лучше, чем производительность классификатора мешков.

[+ Code](#)
[+ Text](#)

```
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fb4bab79978>]
```



Double-click (or enter) to edit

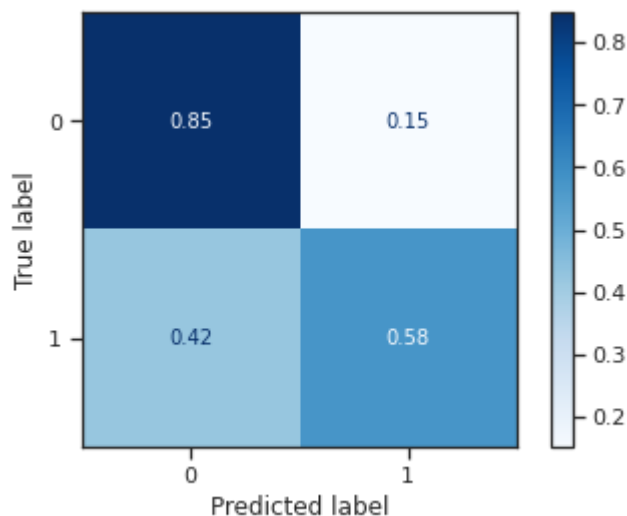
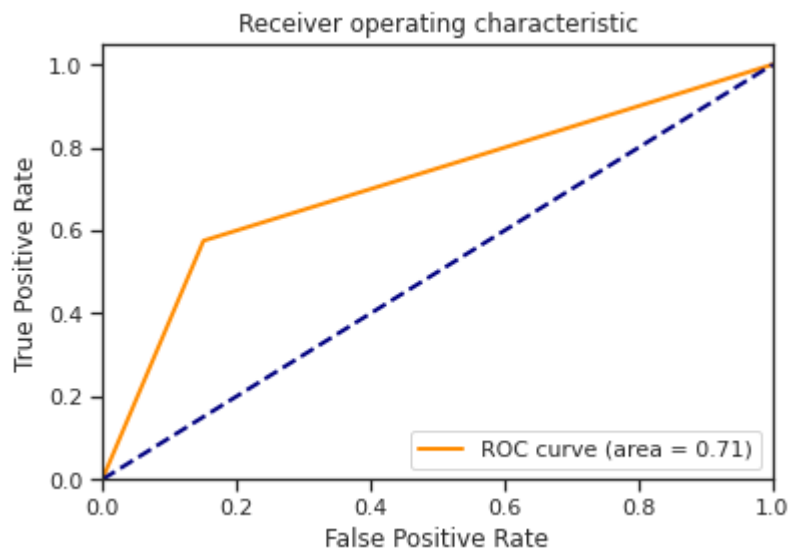
```
clas_models_grid = {'KNN_13':clf_gs.best_estimator_}
```

```
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=13, p=2,
                    weights='uniform')
*****

```



Таким образом, производительность классификатора Bagging лучше, чем у классификатора K-NN. Мы также увидели среднее значение его точности исполнения на тестовых данных.

```

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 = BaggingClassifier()
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)

```

```

[0.86111111 0.80555556 0.69014085 0.8028169 0.95774648 0.74647887
 0.73239437 0.81690141 0.78873239 0.81690141]
80.19

```

## Решение задачи классификации

Итак, наконец, мы используем классификатор Bagging и видим его производительность по данными.

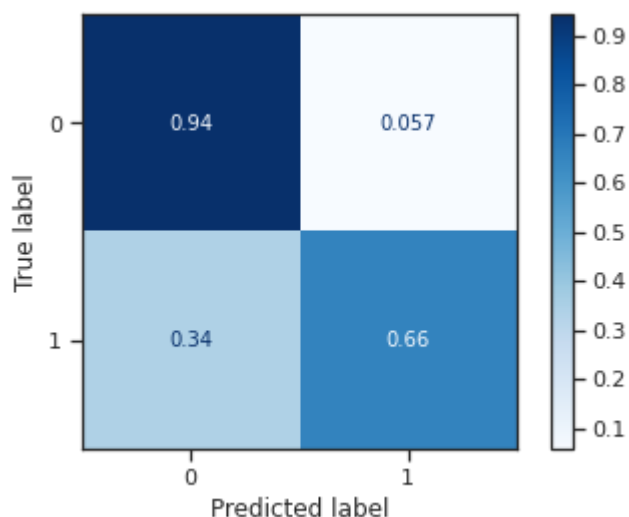
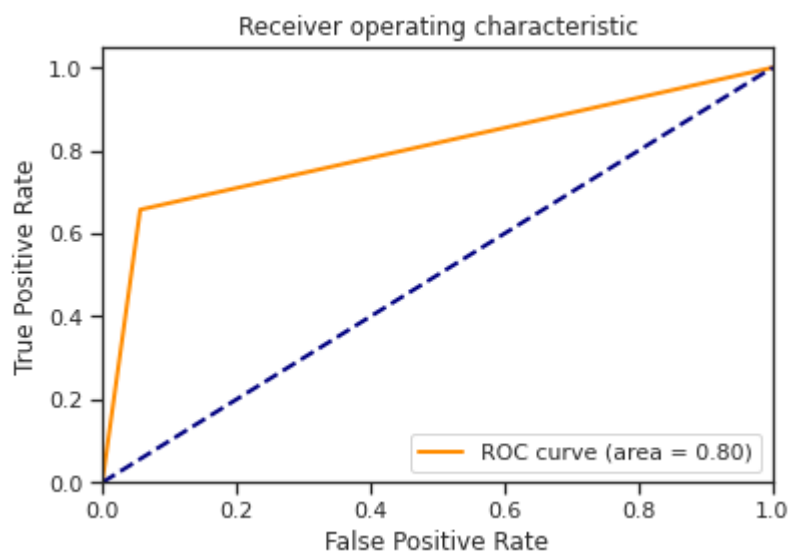
```
clas_models_grid = {'Bag':BaggingClassifier()}
```

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

```

*****
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  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)
*****

```



## Выводы

Таким образом, в нашей модели мы хорошо изучили данные и решили, какой из ее столбцов (после обработки всего объекта и пропущенных данных из него). Затем мы использовали оценки их работы с нашими данными. Наилучшая производительность была у классификатора, поэтому мы используем нашу модель для прогнозирования значений тестовых данных (данных без кросс-валидации). Это означает, что наша модель имеет точность приблизительно 80%.